2011

GVSU Art Gallery Meets iOS: How To Cram 10K+ Works of Art Into Your Pocket

Andres Solano
Grand Valley State University

GVSU Art Gallery Meets iOS: How To Cram 10K+ Works of Art Into Your Pocket

By
Andres Solano
December, 2011

GVSU Art Gallery Meets iOS: How To Cram 10K+ Works of Art Into Your Pocket

By
Andres Solano

A project submitted in partial fulfillment of the requirements for the degree of

Master of Science in

Computer Information Systems

at

Grand Valley State University

December, 2011

Jonathan Engelsma **Date**

Table of Contents

List of figures

Abstract
The Grand Valley State University Art Gallery has over ten thousand works of original art, including paintings, sculptures, ceramics, and more. A large portion of this collection is displayed across GVSU's campuses. These pieces of art are used by students for courses and research and are also simply enjoyed by people each day as they encounter them on campus. Although each individual work has a placard describing it and its artist, these do not help people locate specific works of art around the different campuses and buildings. Furthermore, the amount of information that can be displayed on a work of art is limited to the size of the placard. Another difficulty is that there is no way for a person to share or comment on a work of art that they really appreciate. This project addressed these problems by developing and deploying a mobile application that allows users to browse and locate works of art using their iPhone and its network connection and GPS capabilities. Furthermore, users can use the application to go on virtual art tours, and they can interact with their friends around a piece of art via integration with Facebook, Twitter, and other social networks, all from their iPhone.

Introduction

When we are walking around the GVSU campuses, people will have the opportunity to see more than just buildings and grass. Outside of some of the buildings people may find sculptures or some other kinds of art belonging to the University. Inside of the structures they'll see paintings that are also part of the collection of art GVSU has to offer to the University community and visitors from other places. Viewing art can be enjoyable, but when you have ten thousand works of art in a collection, remembering where they are and all the information relevant to each work becomes complicated. While each work has a placard containing some information about the piece, this is minimal information and many people would like to know more. The Art Gallery at Grand Valley State University is in charge of this collection and a companion digital website that provides a digital view of its collection. While the website is useful, often when actually viewing the art people do not have a computer and hence cannot access the website. , In addition, sometimes people might want to be able to find where the artwork is physically located so they can go view it. It is problems like these that motivated the GVSU Art Gallery's decision to implement a mobile solution that allows people use the iPhone, iPod or iPad to share GVSU's art collection with the rest of the world. These devices have networking and GPS capabilities to help students, faculty members and visitors to find their way around the campuses and locate the work of art they are searching.

Background and Related Work

Museums like the Museum of Modern Art (MoMA)[1] in New York and the Brooklyn Museum (BKMuseum)[3] have developed their own applications to share their collections, events, works of art, and other services the museums offer to general public.

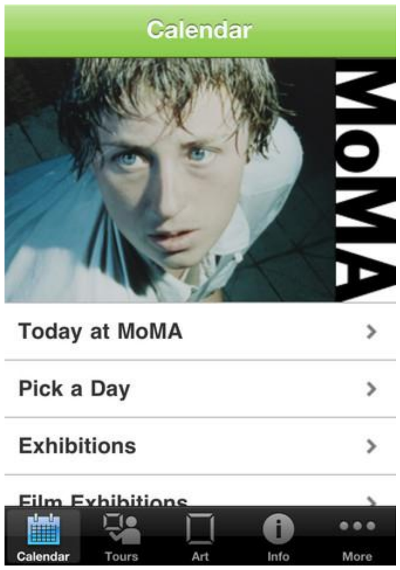


Figure 1: MoMA's application for iPhone

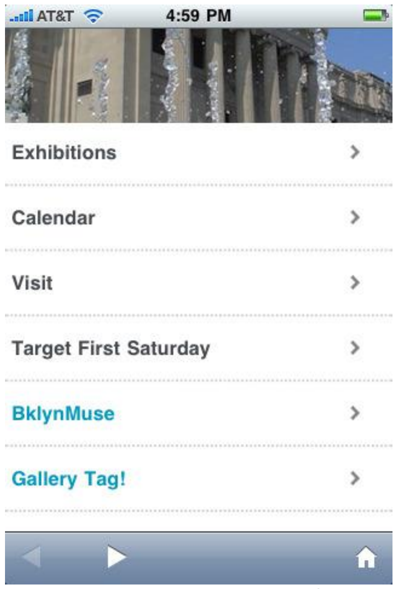


Figure 2: BKmuseum's application for iPhone

MoMA’s application[1] can be found in the AppStore and Android Market; furthermore, in the AppStore users will find the app running on both iPhone (iPod) and iPad. For the iPhone version the application will display the calendar, tours, art and information as the main functions, but will show also other options like Podcasts, Tracks, Tell a friend, Suggestions among other menus. The app requires iOS 3.1 or later. On the other hand, the iPad application will change a in the way it displays the content and the content itself will change, the name of the App is AB|EX|NY (Abstract Expressionist New York)[2], the menu will show the user a Home button a browse menu, a map, a videos list, the list of art terms, the about and the buy section. The app will work on devices that run iOS 4.2 or later.

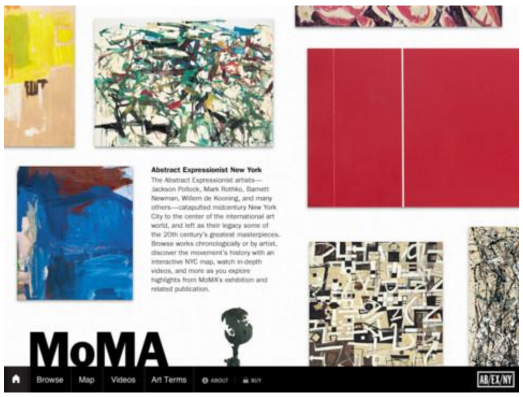
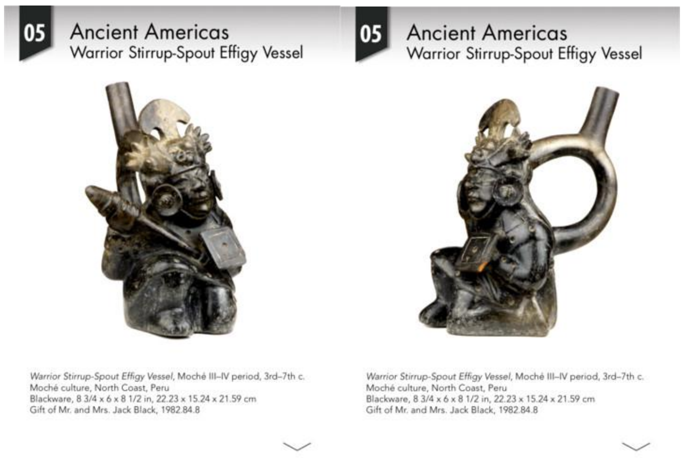

Abstract Expressionist New York
The Abstract Expressionist artists—Jackson Pollock, Mark Rothko, Barnett Newman, Willem de Kooning, and many others—catapulted midcentury New York City to the center of the international art world, and left as their legacy some of the 20th century's greatest masterpieces. Browse works chronologically or by artist, discover the movement's history with an interactive NYC map, watch in-depth videos, and more as you explore highlights from MoMA's exhibition and related publication.

Figure 3: MoMA's application for iPad

The strength of this application lies in the amount of information it shows to the user and the multimedia related to the works of art the museum exhibits to the public. The location of the piece inside the building is another interesting feature to note. The app will tell the user in what floor and room the work of art will be placed. The app does not need to show the geo location of the art work given that all the pieces are inside the building. If the person using the app needs to go there, the only thing he has to do is get the location of the building from any other map application the device can offer. This app is solid and has a lot of content. The UVaM (University of Virginia Art Museum)[4] application takes yet a different direction. This app is capable of displaying 3D images of objects and rotating them over the Y axis giving a better appreciation of these art pieces. Although is not possible to rotate them in all axes, it does give the sensation to the user that they can interact with the piece - something they would not be allowed to do in the museum with the actual work itself.


05
Ancient Americas
Warrior Stirrup-Spout Effigy Vessel
05
Ancient Americas
Warrior Stirrup-Spout Effigy Vessel

Warrior Stirrup-Spout Effigy Vessel, Moché III–IV period, 3rd–7th c.
Moché culture, North Coast, Peru
Blackware, 8 3/4 x 6 x 8 1/2 in, 22.23 x 15.24 x 21.59 cm
Gift of Mr. and Mrs. Jack Black, 1982.84.8

Warrior Stirrup-Spout Effigy Vessel, Moché III–IV period, 3rd–7th c.
Moché culture, North Coast, Peru
Blackware, 8 3/4 x 6 x 8 1/2 in, 22.23 x 15.24 x 21.59 cm
Gift of Mr. and Mrs. Jack Black, 1982.84.8

Figure 4: UVaM's application for iPad

The iPad version has an additional feature, besides the ones mentioned in the iPhone version as well as a different look. In this version a Map menu will be shown to the user and he'll be able to see places where the artists used to live or some of their works around Midtown, Long Island, Downtown and Manhattan. The amount of art works displayed seems to be less than the other app, but only because it is different content. The way to show it on the iPad screen changes with respect to the iPhone and iPod devices.

The BKMuseum application[3] has a lot similarities with the MoMA's app[1]. However, the way it displays the information is more like a web site embedded page for mobile devices. In its main menu the user will be able to see the "Exhibitions", "Calendar", "Collection", "Visit", "Target full web site" and the "Full web site" options; in order to run on iPhone the app will need at least the iOS 3.1. The advantages of this application are not many, except for it can run easily just having the network connection due to its lack of location capabilities and the way the app display the information to the user.

Program Requirements

At the outset of the project the following questions were explored:

- What do we need to show to the user?
- How are we going to get the information the app needs to show?

- How are we going to show that information to the user?

The “what”, it was more related with what kind of menus to divide the information we were going to show to the user. Those menus has some similarities with the other apps mentioned in the related work session. They are “Tours”, “Browse”, “Search” and “Favorites”. For each of these menus the questions remains making this some how recursive until we get to the work of art the user is looking for.

Once we defined this, the next step was to obtain the information we want to display. For that purpose the Art Gallery utilizes a web based catalog platform called CollectiveAccess (CA)[8]. CollectiveAccess is for cataloguing and archive collections. It is also highly configurable and with minimal programming effort you can use external data resources, repositories and support a variety of metadata standards. It also supports a Web service framework[11] to request information from the server for the different kind of objects the catalogue offers. This is the way the mobile app for the iPhone and iPod will request its content from CollectiveAccess[8] for “Tours”, “Browse” and “Search” modules. The “Favorites” module is going to use a different approach. Even knowing that CA does have a way to support content information added by the users, the CA software does not have a way to create a list of favorites in its model. Therefore the works of art the user selects are going to be stored in a database inside the device. The problem with this is the list would be only in that device and if you try to see that information in another device or platform you are not going to be able to get it no matter if the other device belongs to the same user.

Now that we know from where we are going to get the information for the application it is time to define how this content will be displayed on the screen for each module.

The “Tours” will be the first thing the user is going to see and it will show the list of available tours the user has access to. Once he selects a tour a map the tour stops or art works will appear and a list of images of each work of art will also appear at the top of the map. If the user wants to find where the work of art is located, he only needs to click the images or the pins displayed by the map. This will activate a pop up that is going to show him the title and a button he can press to get all the information regarding that piece.

The “Browse” is a list constructed hierarchically. At the moment the list will show the campuses at the very beginning, then after one of the campuses is touched it will display the list of buildings that campus has. Once the user selects a building, if the building has sub levels it will show those levels otherwise the user will see the list of works of art that building or level has. If he touches one of these works of art it will be displayed with the information about that piece.

In the “Search” module, the user could look for artist or identification number (ID), whether is an ID or an artist name he’ll receive a list of IDs or artists depends on what he was looking for or a single object in case the search parameter was accurate enough. Given that the result does not exist then he won’t see anything in the list. Once he gets the results if the object he was trying to find is an artist and he touches it, the next thing the application will display is a list of works created by the artist selected. Otherwise if the object was an ID, the works of art or the single piece will be displayed by the list and as in any other work of art if the user clicks on it, the user will end up viewing the information related to that work of art.

For the "Favorites", if there are works of art marked as favorites in the detail view, then the Favorites module will have elements to show otherwise it will be empty. If it isn't empty the works the list contains will be viewable and after a work in the list is touched it will take the user to the detail view of that artwork. This is going to be the only list where the user can remove objects in the entire application.

The detail view is where the user can see the information about the work of art. This is the heart of this application, because every single module will take the users to this specific view at the very end. In this view the user can check things like the title of the piece, the author, a physical description, a historical narrative, the medium, the date, the location, the identifier or ID number and a list of additional works by this artist. The fields which have a gray arrow at the right are clickable and will display the remaining part of that particular data that the user can't see in this view. Furthermore, this view contains an image to give the user a quick preview[6] of this piece and if the magnifier glass button is touched a view with a bigger image is displayed with zoom in/out capability. There is yet more. Touching the pin button the user will be able to locate that specific work in the map or pressing the share button can can share it with his Facebook friends, email it and also tweet about it. The last button will allow the user to designate the work of art as a favorite and save it to a list of favorites on the device.

Implementation

Having enumerated the requirements of the application, now we need to define what tools to use to implement this solution and how it will be the communication with the CollectiveAccess server.

Platform

The first step was to decide how the application would be implemented. Given the fact that the application needs to work on iPhone and iPod, which are Apple products, our options to work with the Apple's Objective-C[14] programming language and Xcode IDE.

After make this decision, I had to get used to the Objective-C syntax and the model it uses to build applications. The model used is widely popular and known as the MVC (Model-View-Controller) Architecture[5]. The other thing would be to start learning the iOS Framework to be able to make network connections, String processing and many other common things you can do in any programming language.

Server Communication

As I said in the requirements session, the communication will be done using the web services provided by CollectiveAccess[8]. The web services supports a broad range of functionalities provided by the software and they can be accessed using any programming language. They are divided in 4 mayor groups:

1. Cataloguing: Change data.
2. ItemInfo: getting detailed information on an object, entity, occurrence, etc.
3. UserContent: Creating and viewing user-contributed content like comments, ratings, tags and sets.
4. Search: Searching the dataset using the CollectiveAccess search engine.

All these groups can be accessed using either SOAP (Simple Object Access Protocol)[13] or through RESTful[12] URLs. At the beginning the app started using SOAP[13] to communicate with the server, but this approach wasn't very efficient given that the protocol returns too much information and sometimes using SOAP the communication with the server never returned data and at other times it worked. The problems could be related in some way with the Art Gallery server. Besides processing these SOAP responses from the server on the iPhone is not as simple as processing simple XML or JSON representations.

The RESTful[12] approach was adopted, but the Web services don't return JSON as I expected at the beginning. The responses the server would give us would still have to be parsed by the client.

The URL format the application will use to make the request to the server looks like this:

```
HTTP://LOCALHOST/SERVICE.PHP/<MODULE>/<SERVICE>/<ACTION>?METHOD=<METHODN
AME>&<ARG1>=<ARGV1>&<ARG2>=<ARGV2>&<ARG3>=<ARG3>
```

Where:

- localhost: It will be the Art Gallery Server.
- <module>: One of the groups using no CAPS; for example: iteminfo.
- <service>: One of the groups with the first letter of each word in the name CAPS; example: ItemInfo.
- <action>: If it is RESTful or SOAP, in our case "rest".
- method: The function to use when the service is called.

As I said earlier, given that only two of the four groups were used, then the methods and its parameter would vary. The table below will show what methods were called from the ItemInfo web service marked with "X" in the right column.

Return type	Method & parameters	Used by the Application
Int	auth (string $username, string $password)	
Array	get(string $type, int $item_id, array $bundles, array $options)	X
Array	getAttributes (string $type, int $item_id)	
Array	getAttributesByElement (string $type, int $item_id, string $attribute_code_or_id)	
Array	getApplicableElementCodes (string $type, int $type_id, boolean $include_sub_element_codes)	
Array	getItem (string $type, int $item_id)	
Array	getLabels (string $type, int $item_id, string $mode)	
Array	getObjectRepresentations (int $object_id, array $versions)	
Array	getPrimaryObjectRepresentation (int $object_id, array $versions)	
Array	getRelationships (string $type, int $item_id, string $related_type)	X
Array	getRelationshipTypes (string $type, int $sub_type_id, string $related_type, int $related_sub_type_id)	
Array	getSets()	
Array	getSet (int $set_id)	
Array	getSetItems (int $set_id, array $options)	
Array	getSetsForItem (string $type, int $item_id)	
Array	getLastChangedItems(string $type, int $timestamp)	

The other group of Web Services used was Search[10], the following table shows the methods in the service definition and which ones were used by the application also marked with “X” in the right column.

Return type	Method & parameters	Used by the Application
`DOMDocument`	`querySoap(string $type, string $query)`	
`Array`	`queryRest(string $type, string $query)`	X

Thanks to some changes made in the original web services calls performed by the CollectiveAccess Developer Team, we could avoid the use of some of these methods to get the information we need to display by using more than one request to the server. For example,in the "queryRest" method when you were going to perform a search, once you gave it the parameters the XML response you were going to receive from the server would contain the object ID, the name with which is known in the system and the ID number. The problem with this was the application needed more data like the artist name in case the user was looking for artist and the images of the works of art if the search was by ID number and the server could retrieve none of them. The XML answer will be the same no matter how you configure the query and change the type of object you were looking in the system. Therefore, if the method wasn't modified the procedure to get this information would require the following steps for searching by ID number were required:

1. Use the queryRest method to get the list of works of art or the work of art the user was looking for.
2. Once the response was parsed by the application, then uses the object ID to make a call for every element in the list to get its description to a different method in the ItemInfo service.
3. In order to get the icon the list would display we need to do the same in the step 2 also for every element, but calling a different method in the ItemInfo service and this call will return only the URL where we can get the image. Therefore, a last call for every element in the list needed to be done in order to get the image.

The problems this posed for the application was making 4n+1 network calls, where n is the number of objects in the list. This would incur in performance issues (latency) and high battery consumption. One way to avoid this problem would be limiting the amount of elements we can call. However, this should be done in the first call and the queryRest method did not support pagination and neither did the ItemInfo methods.

For these reasons, CollectiveAccess modified the queryRest method to get more information about a specific work of art in a single call, and then using this new version we can skip the URL and the description calls. Therefore, the amount of network calls would be reduced to n+1 better than 4n+1.

The new call looks like this:

```
http://localhost/service.php/search/Search/rest?method=queryRest&type=ca_objects&query=idno:%@*&additional_bundles[work_description]&additional_bundles[access]&additional_bundles[ca_object_representations.media.icon][returnURL]=1
```

And the server response will look like this:

```
<CaSearchResult>
```

```
        <ca_objects object_id="">
                <displayLabel></displayLabel>
                <idno></idno>
                <work_description>
                </work_description>
                <access></access>
                <ca_object_representations.media.icon>
                </ca_object_representations.media.icon>
        </ca_objects>
</CaSearchResult>
```

This kind of changes also had to be made in "Tours" and "Browse". When you were going to get the list of works of art the app could limit the amount of pieces to retrieve from the server in the subsequent calls. This was necessary to boost the performance in cellular networks, then the application request only 25 pieces at a time. Furthermore, to be able to get more information related with the piece the "get" method in ItemInfo service was also modified.

One thing I haven't mentioned so far is the parameter "type" that you can watch in the service's methods, this parameter refers to one of the following objects the CA[9] manage:

Item	Table name
Objects	ca_objects
Object lots	ca_object_lots
Entities	ca_entities
Places	ca_places
Occurrences	ca_occurrences
Collections	ca_collections
Storage locations	ca_storage_locations
Loans	ca_loans
Movements	ca_movements

The application uses ca_objects, ca_entities and ca_storage_locations. The ca_objects are the works of art in CA, the entities are the artists' representations inside CA and the ca_storage_locations is the way to represent the buildings where the pieces are located. The other items in the table haven't been used by the app until now, so if you want to know more about them you can refer to the CA wiki for more information[8].

Results

The result of this process can only be shown; the first version published on the AppStore will look as follows:

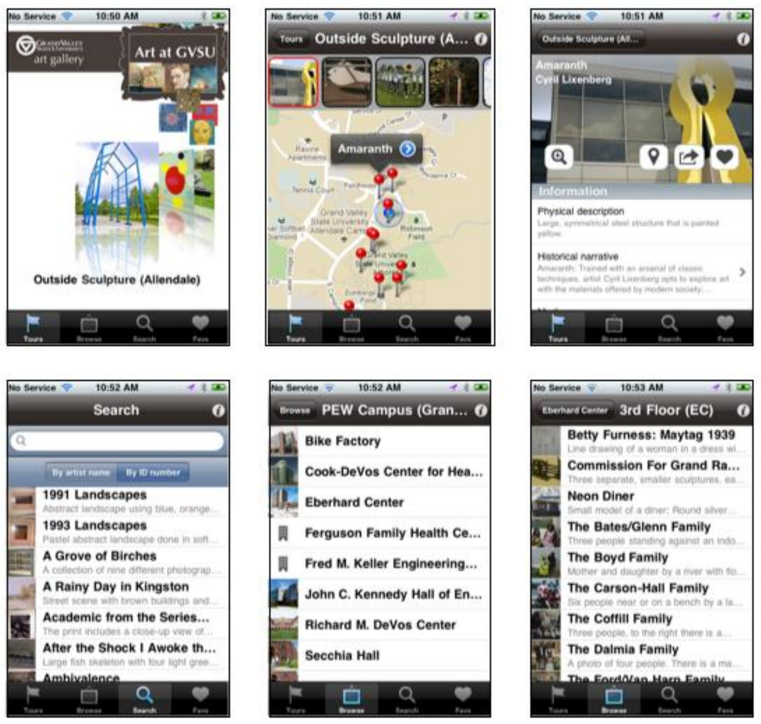


Figure 5: Art at GVSU for iPhone, version 1.0.

The image above shows the final product in its version 1.0. From left to right in the first row the Tours module, the next image is the map with the stops of the selected tour and at the end of the first row of images you'll see the detail view where the information for the work of art is displayed. In the second row of images in the left you'll see the search module performing a partial search by ID number and at its right the browse module showing the buildings in one of the campuses in this case the PEW campus. At the very end of the second row of images you'll see how the list of works of art looks. If you see the search by ID number and the list of artworks are almost identical, there is only one difference visually speaking and it's the search controls at the top of the search view, the content displayed should be the same.

Evaluation and Reflection

Evaluating the app at this point is a bit difficult, because it has only been published on the AppStore for one week and very little analytics data has been gathered thus far. Therefore, the only way we can present the results would be qualitatively after use the application.

1. The accuracy of the application is based on the information that is presented to user the geo location as well as the artist and the piece information. All this information comes from the CollectiveAccess software the Art Gallery has implemented for this task.
2. The app's efficiency can be measured by considering how fast the user will receive the information on the screen. Considering some of the limitations we had at the beginning doing more network calls per element in the list retrieved by the server, I think every module had an improvement at least using WIFI the app is relatively fast. Testing in the iPhone4 and 3G, the response was acceptable.
3. Another thing we can measure would be if users can find what they are looking for using the app, I did a test looking if I could find the Seymour and Esther Padnos painting by David LaClaire located in the Kennedy Hall of Engineering building, 3rd floor. It worked being very accurate, but when I used the search to get there again for some reasons worked sometimes and some others didn't work. But as long as the user can find his objective with the app, he is going to be pleased.

At this point the only think I can say without having enough data is the application is doing its job. Regardless, some issues that affect the performance and the memory usage in some devices like my iPod, which is running iOS 3.1.3 and sometimes sends me this message, perhaps is the control managing the images in the "Tours".

For the next version, the application requires some fixes for some strange behaviors known as bugs,. So far, users have reported the following issues:

- The photo zoom-in view acts strangely. It seems to gravitate to the bottom of the screen. Take a look at item 2003.238.1 for example.
- Random crashes which will need to be resolved by analyzing the stack traces..
- You cannot cancel out of some of the ShareKit screens, and have to manually kill the app from task manager in order to get back into main app.

Conclusions

Summarizing the work done so far. The platform of choice for implementing the Art Gallery app was to write a native iPhone app. To do this I had to learn Xcode and Objective-C.
Now that we have the platform and the tool defined, then we needed to see how the app was going to get the information to display. Therefore, it was necessary for the mobile client to interface with the CollectiveAccess server used by the Art Gallery. Here is where the web service interfaces of Collective Access were important. Some of the web services interfaces required modifications to improve the performance and decrease the battery consumption due to excessive network calls. We also had to abandon

original attempts to use CollectiveAccess's SOAP web services as they simply are not appropriate for mobile apps. Once we got past the web server interacing issues we were able to complete the implementation and publish the app to the Apple iTunes AppStore. In the future analytics data will be gathered to try to understand how users are using the app.

Future Work

For future releases of the application the staff of the Art Gallery have as do people in the School of Computing. The following ideas are currently on the table:

- Barcode search: All the works of art have adjacent to them a placard with information about the work. The idea is place QR Codes on the placards. That way if the device has a camera the user will be able to use the search module and scan the Code and be taken to the artwork's detail.
- Game Mechanics: In order to encourage more app usage, some sort of mechanism has to be implemented. Gamification is a good way to engage app users.
- User content: Besides the idea of games, the CollectiveAccess has a module where the user can add content to the objects in the software like comments and ratings.
- Favorites: Change the way the app is doing this in the moment instead of storing the list on the device in a Database, save the list in the server.
- 3D Objects: In order to increase the interaction with the works of art, some of them could have 3D objects the users would be able to rotate doing things than they are not allow to do with the pieces in the real world.
- Framework: Another idea that has been around is turn this into a framework to build museum applications for iOS mobile devices; the framework will be built in conjunction with CollectiveAccess.

Bibliography

[1] MoMA iPhone App
http://itunes.apple.com/us/app/moma/id383990455?mt=8

[2] MoMA AB|EX|NY iPad App
http://itunes.apple.com/us/app/moma-ab-ex-ny/id398432441?mt=8

[3] BKmuseum iPhone App
http://itunes.apple.com/us/app/brooklyn-museum-mobile/id378356586?mt=8

[4] UVaM iPad App
http://itunes.apple.com/us/app/uvam/id478915134?mt=8

[5] Conway, Joe, and Aaron Hillegass. iPhone Programming: The Big Nerd Ranch Guide. Indianapolis: Pearson Technology Group, 2010.

[6] Nahavandipoor, Vandad. Graphics and Animation on iOS. California: O'Reilly Media, 2011, pp 137-154.

[7] Chisnall, David. Cocoa® Programming Developer's Handbook. New York. Addison-Wesley, 2010.

[8] CollectiveAccess. [Online] Available, http://collectiveaccess.org/support/developers, 2011.

[9] CollectiveAccess. "Getting_Data". [Online] Available, http://wiki.collectiveaccess.org/index.php?title=API:Getting_Data, 2011.

[10] CollectiveAccess. "Search Syntax". [Online] Available, http://wiki.collectiveaccess.org/index.php?title=Search_Syntax, 2011.

[11] CollectiveAccess. "Web Services". [Online] Available, http://wiki.collectiveaccess.org/index.php?title=Web_services, 2011.

[12] WikipediA. "Representational State Transfer". http://en.wikipedia.org/wiki/Representational_state_transfer, December 11, 2011.

[13] WikipediA. "SOAP". http://en.wikipedia.org/wiki/SOAP, December 07, 2011.

[14] Apple Inc. "iOS Developer Library" http://developer.apple.com/library/ios/navigation/, 2010.