MyWay

Phase 3: Design Documentation

Donglai Wei, Shidan Xu, Neil Gurram

Please see below for wireframes, and the Phase 1 and Phase 2 Design Documentation. We indicated clearly when the start of Phase 1 and Phase 2 occurs.

Programming

First, we need to talk about the programming aspects of our project. Everything that was true regarding programming in Phase 2 still holds in Phase 3. Throughout the code, we made sure we appropriately labeled methods. For example, we made methods with descriptive names like MyWay.SearchWayByInput found in api/client/js/api.js to indicate clearly that we want to look for a way given an input list of data. In addition, we properly used external code so that it wouldn't take away extra time to do something already done. Namely, rather than coding up Johnson's Algorithm to find the all-pairs shortest paths, we used online sources to code it to prevent us from getting bogged up in the details of implementing this somewhat tricky algorithm. We included citations for such external code. Further, we provide comments throughout the code so that it is enough to understand what is happening without there being a distraction. Finally, there is very little duplication in the code (as can be seen in the tests where we use helper methods rather than writing the same lines over and over again), and we made sure we removed all files that are not necessary for certain phases.

In addition, modularity is demonstrated properly with our code. For example, just like we did in phase 2, we maintained a ruby-on-rails style of code management:

- /app
    - server side's MVC components (/models, /controllers)
    - no viewers/, as we are doing REST API
- /client
    - /client/test: QUnit Tests
- /public
    - assets are accessible from the html (/txt, /img, /csv)
- /preprocess
    - offline-codes to help prepare the data for finding the ways

And then we added the client side code to phase 3 in /client:

- /client/html (Deals with all client functionality)
    - *client/html/traveller.html:* focuses on the methods that allow the client to input a list of valid gallery rooms/artwork name/artwork author, and then visualize a way to go through these items.
    - *client/html/mfa.html*: REST login page without authentication (we didn't have time to do authentication)
    - *client/html/mfaDB.html*: after MFA user login, he can change the database with the API calls

- client/js: Provides libraries for graphics, tag input, and smooth scrolling on the pages for the traveller and MFA.
- client/css: Provides the layout

This demonstrate a separation of concerns and shows how the code is divided properly into modules of files. Furthermore, our functions do not have too many arguments. For example, going back to MyWay.SearchWay, we only have two arguments, a list of gallery rooms to see and a function to output the ways, which are both reasonable parameters to have.

Finally, we need to talk about the verification. Firstly, we took the chance to add more tests to augment to our test suite before so that we can be sure that our product is continuing to work as planned. In addition, the graphics behavior seems to be working appropriately. For example, when we click buttons, we do get the appropriate output. Also, we display the ways on the layout whenever the appropriate button is clicked (like “Search” or “Feeling Lucky” on the Welcome Page or Map Page)

Design Challenges

We also had some design challenges to focus on in our project. One challenge in the User Interface was whether to employ a more Google-like Design or a Google Maps Design. Namely, the Old Google approach has a search box initially, and once a user clicks enter, he goes to the second search page; the Google approach has a search box initially, and once we input values in the search box it goes to another page of searching results; and the Google Maps approach has one full page, where the user inputs the data, hits enter, and then gets the destination on the background. We decided to choose the old Google approach. Our understanding is that we are not sure which floor of MFA we are going to be on, so it is more appropriate to wait for the user to decide on which galleries he’s visiting, and then display the map of the corresponding floor with the ways.

A second challenge that we faced was whether we needed to have a Road. The functionality of the Road is for dealing with the case where the path connecting two adjacent gallery chambers are not connected by a straight line, but rather have obstacles in between. However, given the scope of the project, Road seems to be redundant. In addition, to serve our purpose of connecting the paths, we find that Road is less important.

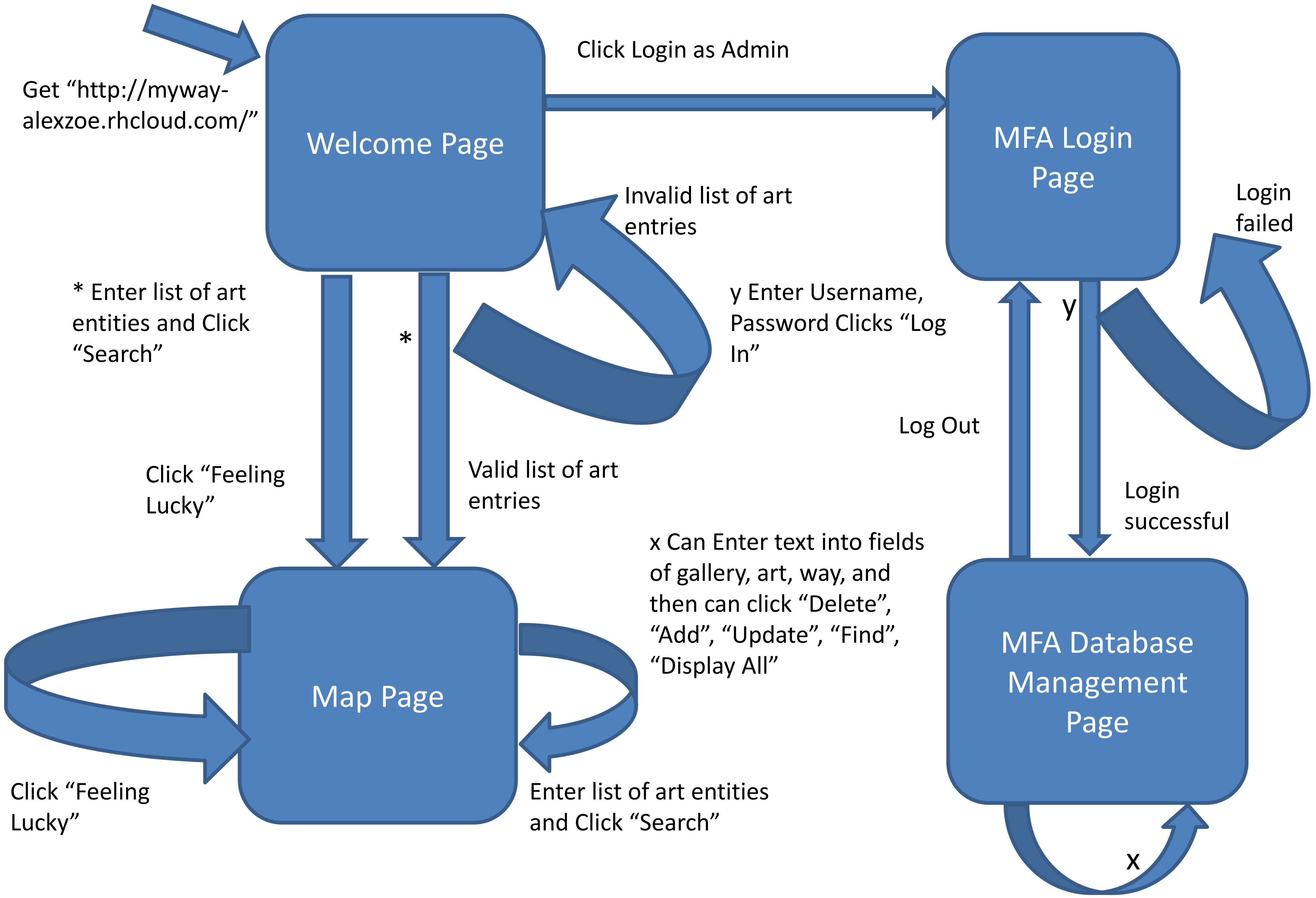
Design Changes

Now we discuss some other changes that we made from Phase 2 to Phase 3. First, we decided that it would be pretty neat to search by artwork or by artist (which we call as “author” for our project), so we decided that we should add to artwork a gallery field, so that we would then display the corresponding gallery after hitting the submit button. We also remove the concept of a “road”, which connects the neighboring galleries. We here directly connect the center of two neighboring galleries as the path, instead of asking for the road that in between. Secondly, because we only have one layout that will be displaying the ways through the museum, we decided to get rid of the map component that we had from phase two. Finally, in order to implement some more of the APIs to make our project a little more complex, we decided to add an MFA username and password for the MFA users to manage artworks.

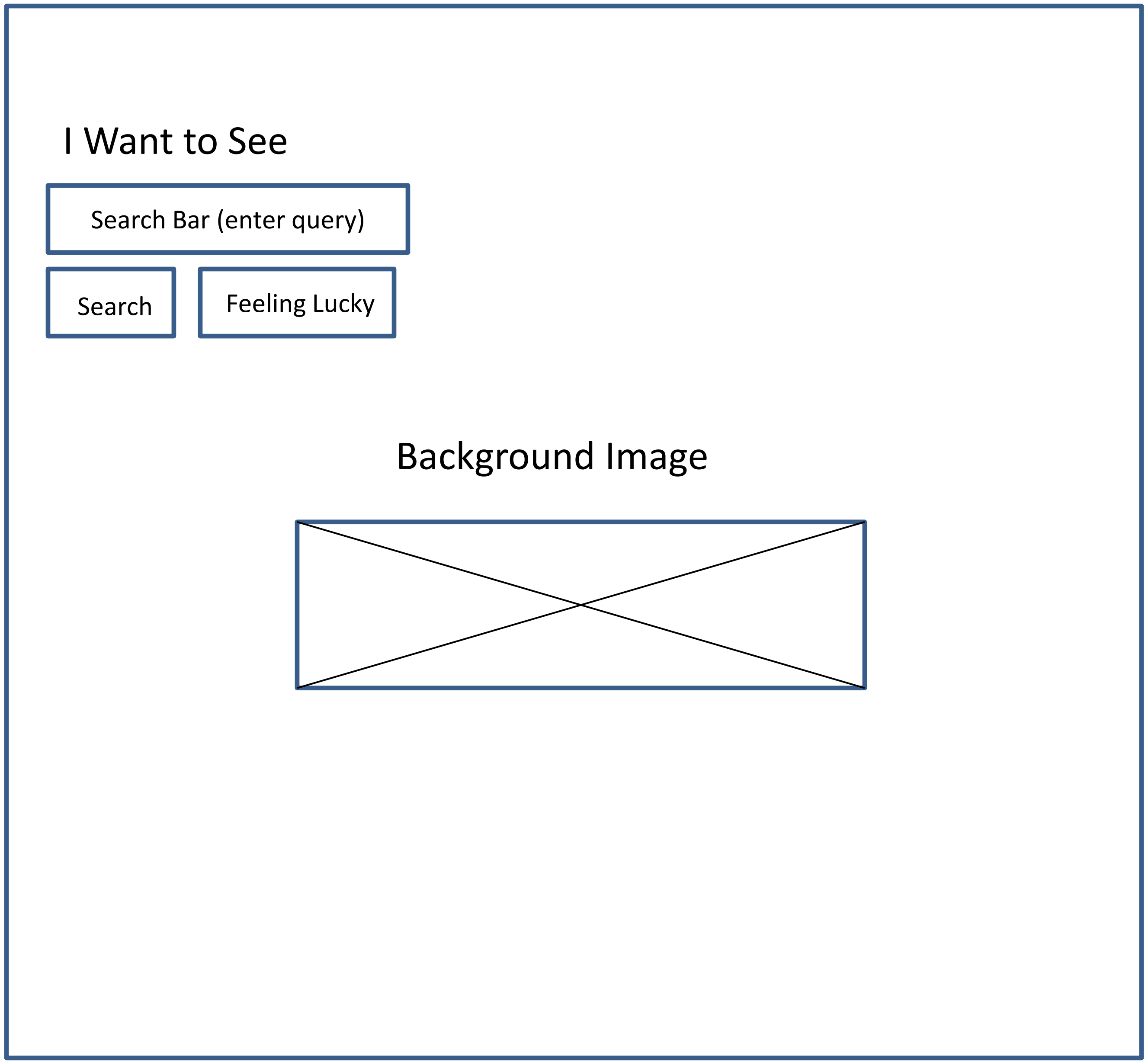
Teamwork

Finally, it is important to talk about the lessons learned in the teamwork aspect of the project. We learned to spend more time discussing about the design rather than coding up things which were later found to be off topic. In addition, we divided the tasks into detailed and “checkable” milestones for all three members. Furthermore, as a team we were able to discuss what unnecessary features we should cut off from our first Minimum-Viable-Product (MVP). Finally, when it seemed like we got stuck on our part of the project, we were more than happy to ask others how should we fix this problem. We realized getting stuck on our own was not a good thing, and we were thankful for being able to get help from both our fellow team members and the T.A.

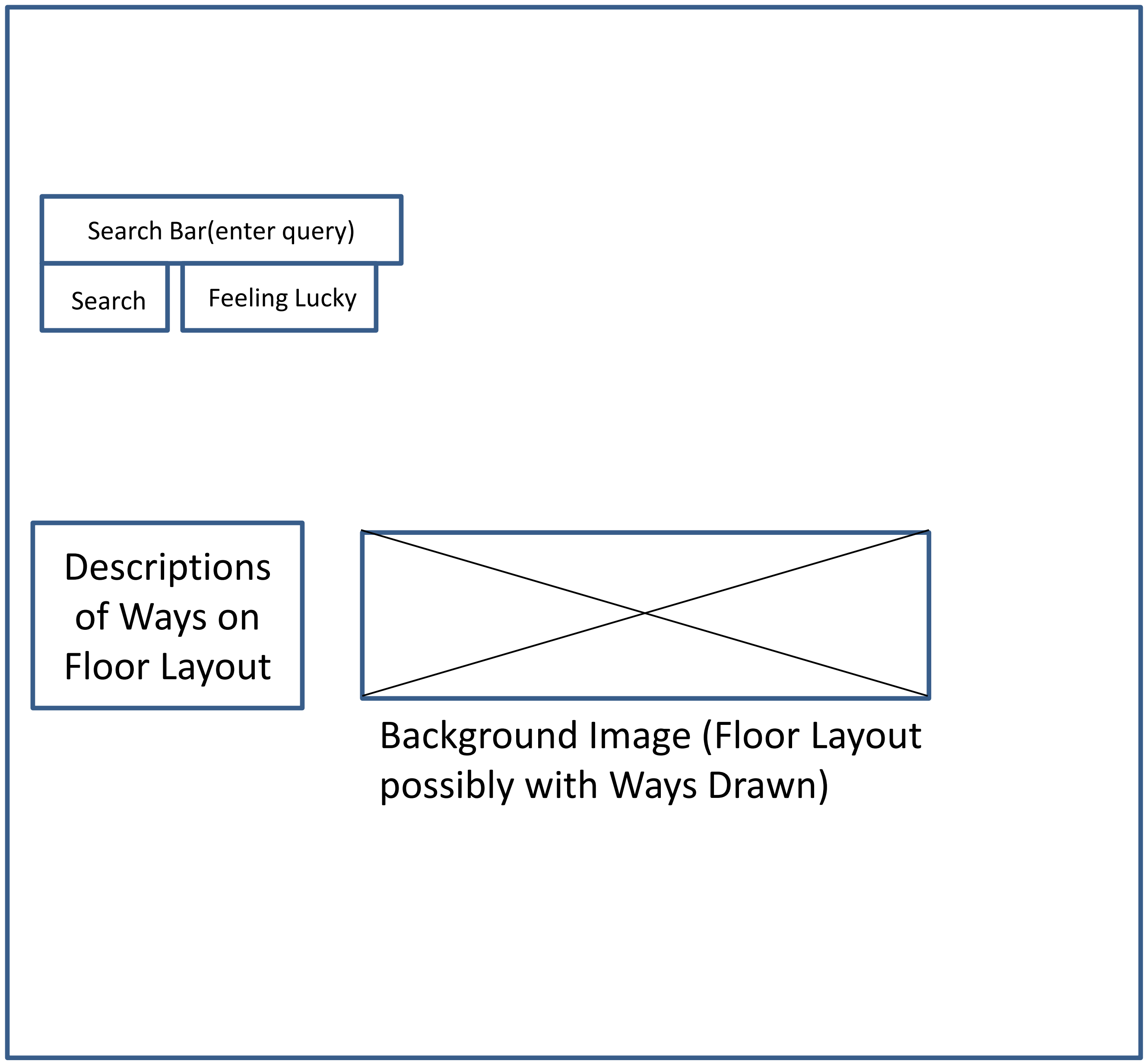
State Diagram (used symbols in case we couldn't fit)


Get "http://myway-alexzoe.rhcloud.com/"
Welcome Page
Click Login as Admin
MFA Login Page
Invalid list of art entries
Login failed
* Enter list of art entities and Click "Search"
*
y Enter Username, Password Clicks "Log In"
y
Log Out
Click "Feeling Lucky"
Valid list of art entries
Login successful
x Can Enter text into fields of gallery, art, way, and then can click "Delete", "Add", "Update", "Find", "Display All"
Map Page
MFA Database Management Page
Click "Feeling Lucky"
Enter list of art entities and Click "Search"
x

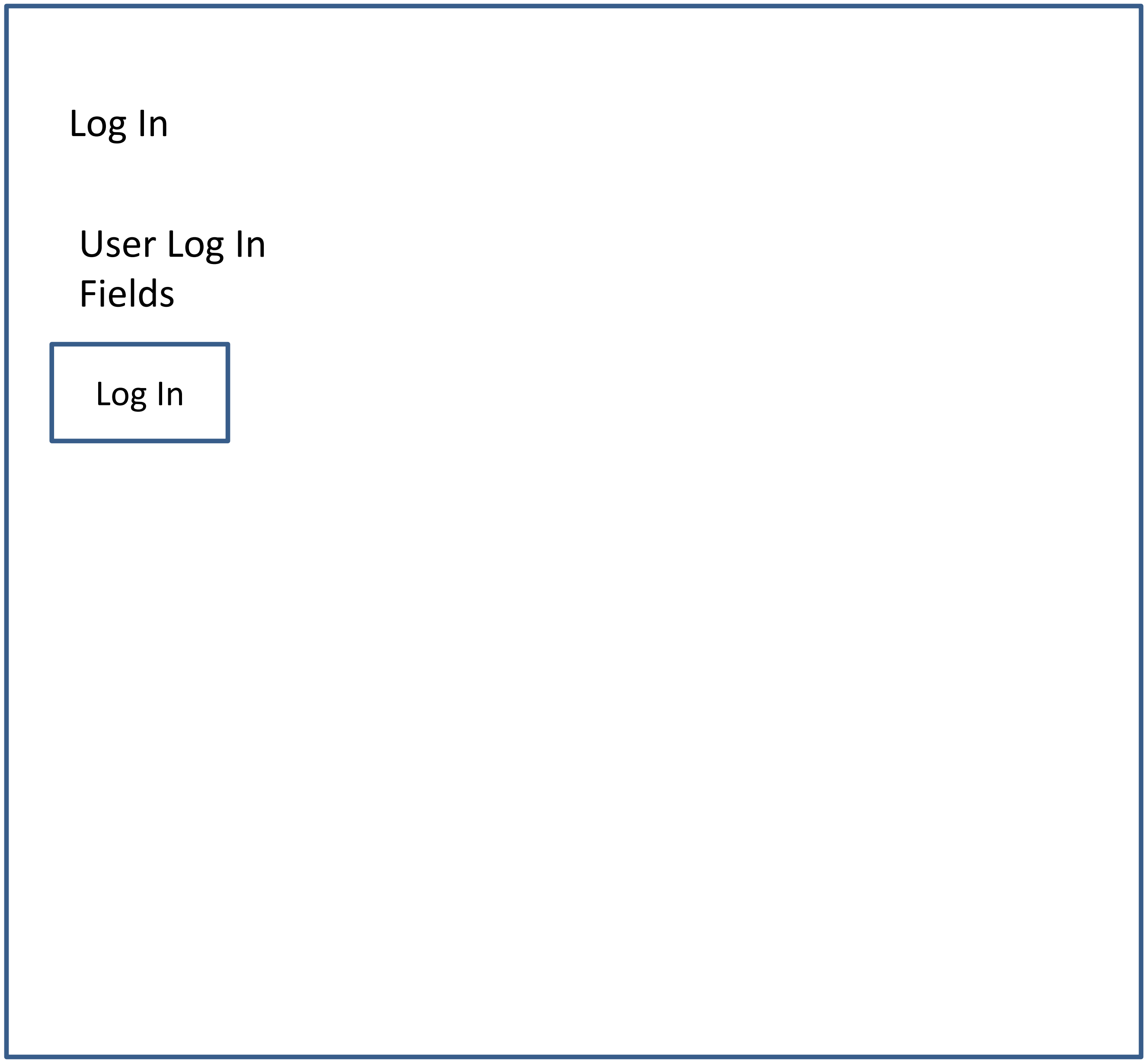
Welcome Page



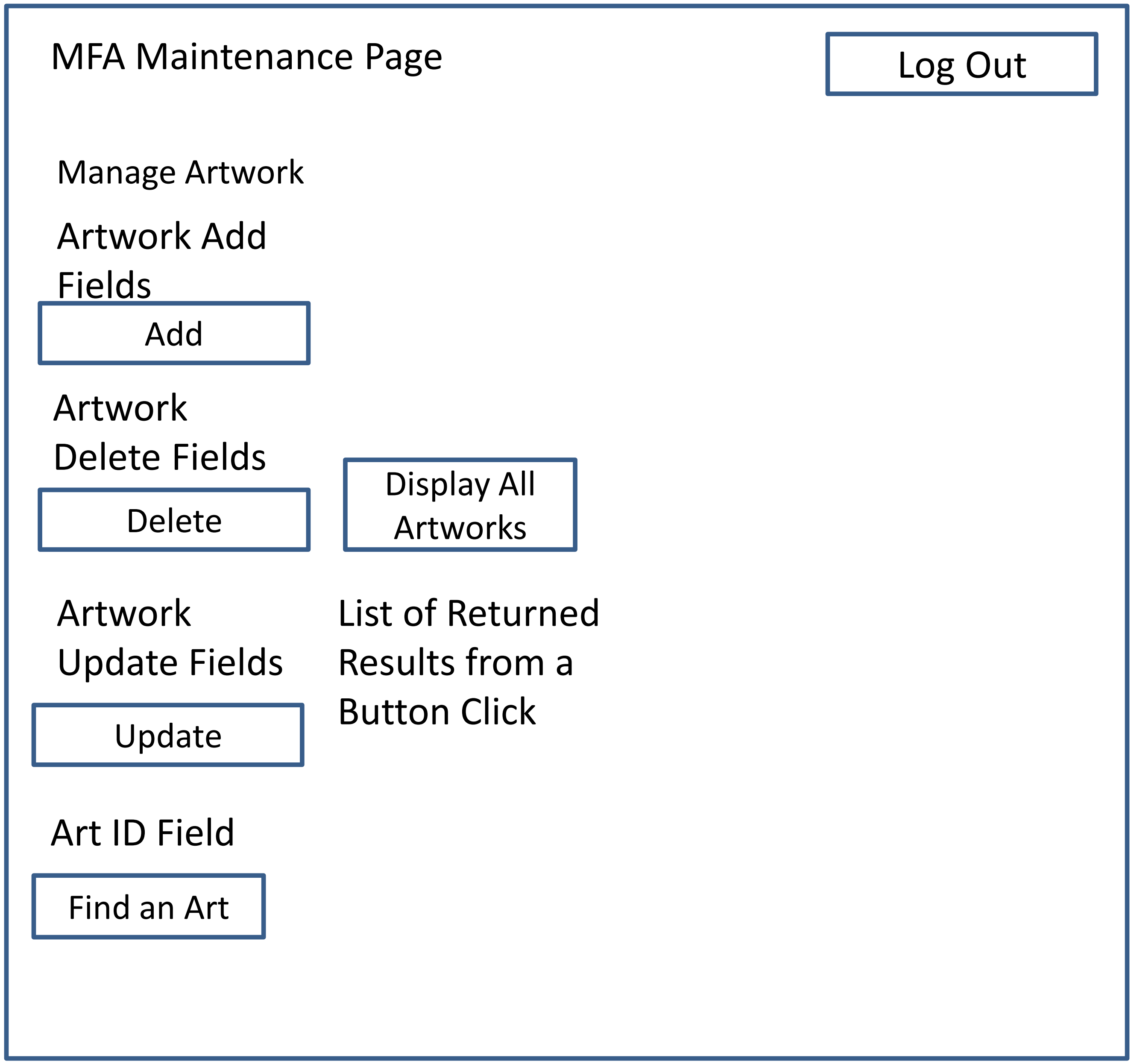
Map Page


Search Bar(enter query)
Search
Feeling Lucky
Descriptions of Ways on Floor Layout
Background Image (Floor Layout possibly with Ways Drawn)

MFA Login Page

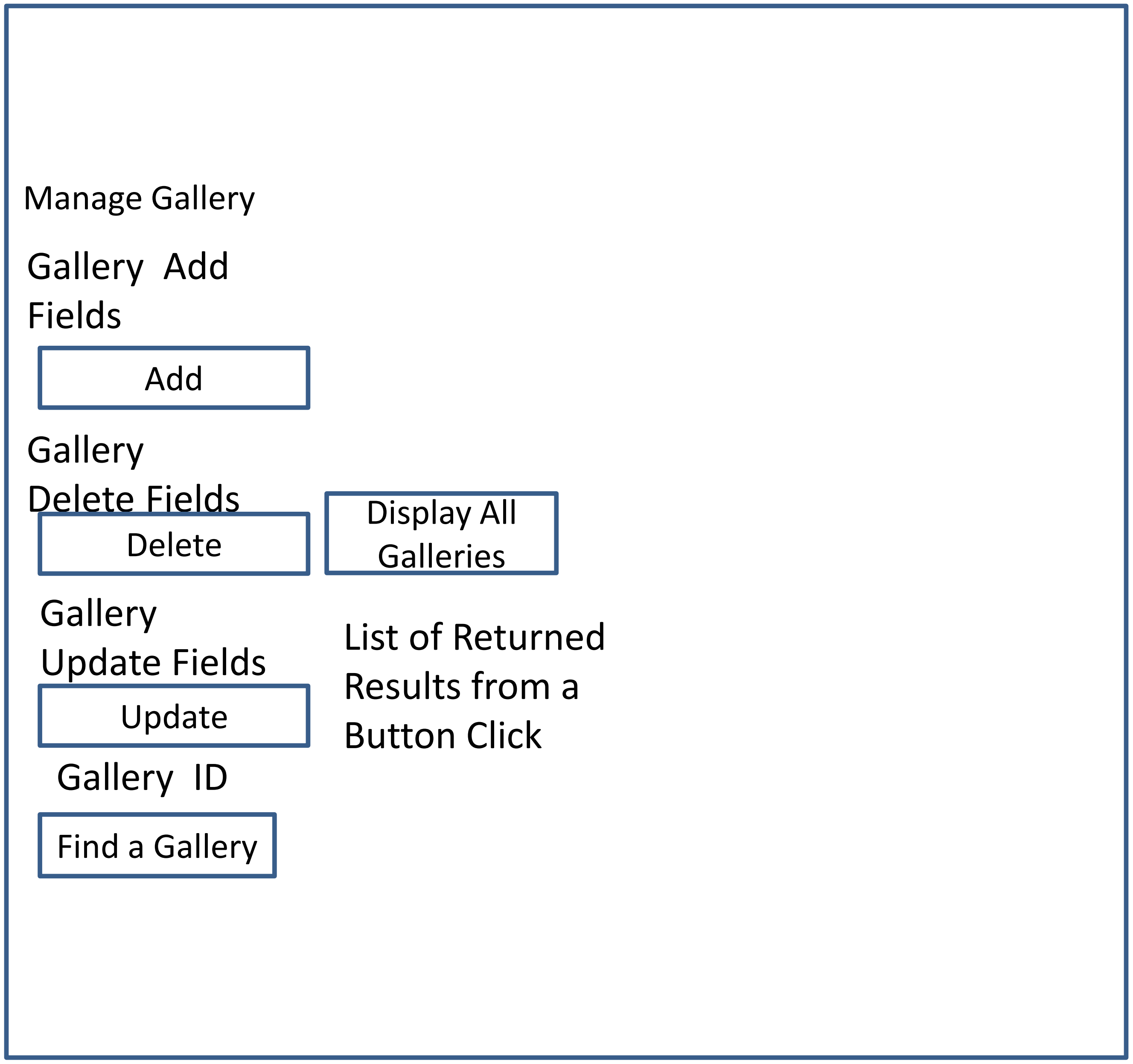


MFA Database Management Page (1/3)



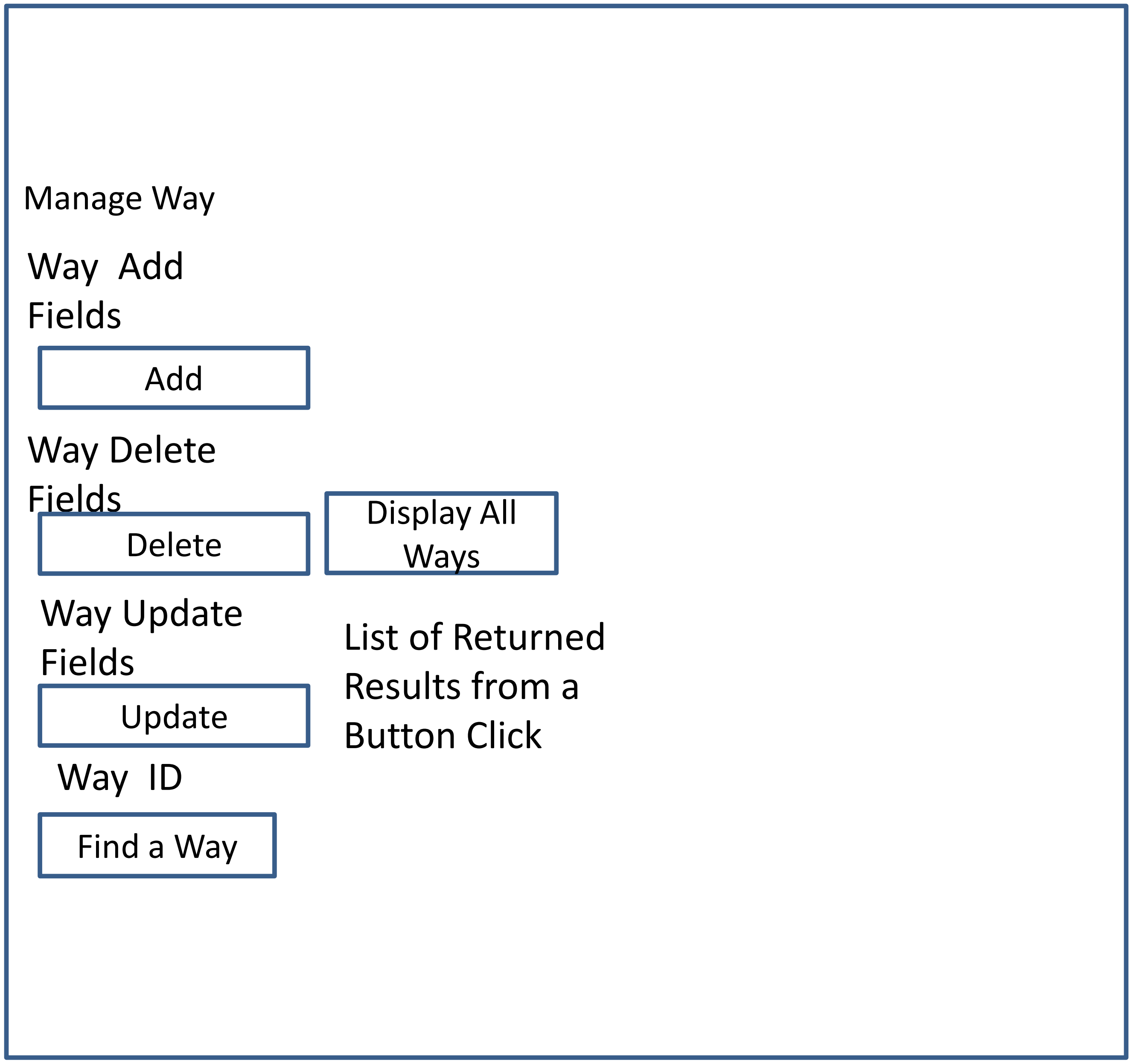
Scroll Down to See Next Part of Page

MFA Database Management Page (2/3)



Scroll Down to See Next Part of Page

MFA Database Management Page (2/3)



Neil Gurram, Donglai Wei, Shidan Xu
6.170 Project 3 Part 1
10/14/14

Team Design for MyWay

We each made our own individual submissions as a contribution for the team design. Then, we went to office hours and took their advice and we collaborated as a team to get a new team design that was quite different from what the three of us had initially proposed.

So, here are the main points needed for Phase 1, all of which come from the **Design Component**.

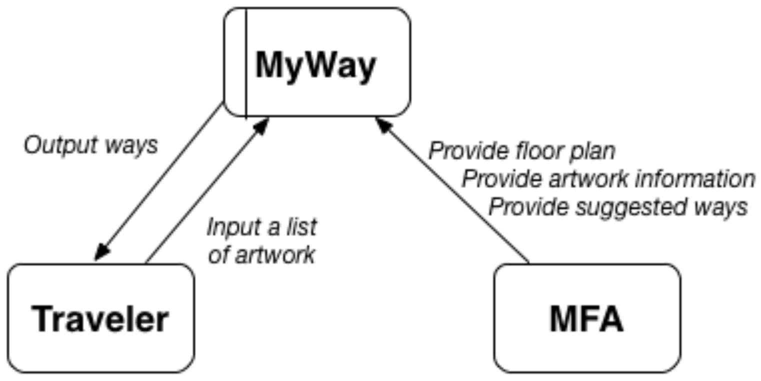
Overview:

We want to realize the following goal for our app: given a list of artworks that one wants to see, how can we create a visual representation for the user to check out the works in the Museum of Fine Arts (MFA) of Boston. This ties in strongly with our purposes:

- help a **traveler** find the best **way** to check out **artworks** in **MFA**
- serve as an advertisement platform for **MFA**
- help a **traveler** organize their interaction with **MFA**.

So far, there isn't an app that can do exactly what we want. All bold faced terms will be talked about later. If anything, our app hopes to be an extension of the three apps of MOMA, Yelp, and TripAdvisor.

Below you will see the context diagram to show the various roles that will interact with our app.



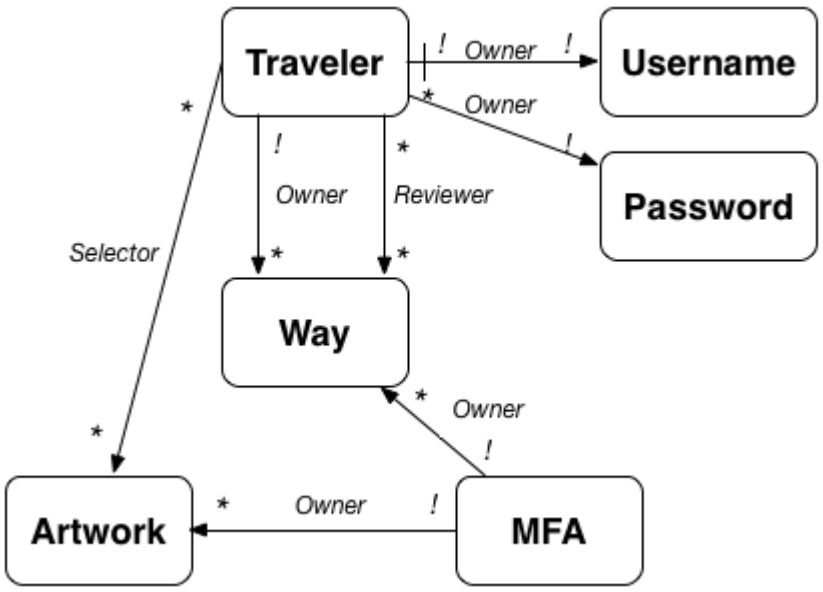
Design Model:

Then, we focus on our design model, and we start by touching base with the concepts.

- The first concept deals with the **way**. A way is just a graphical representation of how to view artworks; more specifically, given that a traveler provides a list of art, the website processes the list and returns a visualization of the routes that the traveler should take, so that we can minimize some characteristic (either trip time or distance, for now). This concept is motivated by the purpose of needing a traveler to organize their interaction with MFA.

- The second concept concerns with the **traveler**. A traveler is just a user who can search for artwork to see in MFA. This concept is motivated by the purpose of wanting a traveler to find the best ways to check out artworks. Another concept motivated by this same purpose is **artwork**, which is an abstract representation of the paintings, sculptures, and other works of art, that are in the MFA. The artwork is the basis of the ways that people can create and search for.
- A final concept under consideration is **MFA** itself. The MFA acts as a service provider who gives floor plans, provides recommended ways, and will update artworks as needed. This concept is motivated by the purpose of wanting to provide an advertisement platform for MFA.

Below you will see the data model to show all our concepts as well as essential data elements and relationships for our app.



Challenges:

This project did not come without its challenges.

- First, we wondered whether we should have our concepts focus on a social aspect in our app (such as becoming friends with people who share a common way). We initially wanted to put this in our app, but then we decided better of it, as we wanted to have our focus be narrowed, and having a social aspect only deviates from our purpose.
- Secondly, we wondered whether we should have a parent class for MFA and a user. This is a good point as both MFA and user can provide ways. However, we decided that they

perform different functions and their difference is too large to necessitate a same parent class to occur, and thus we avoided this so that our app did not become overcomplicated.

- Lastly, we had to choose between using a normalized data model vs and embedded data model. We chose a normalized data model as we have a lot of many-to-many relationships (e.g. multiple travelers can review multiple ways). We want to query from both sides; for example, we want to know how many ways does user A favor, and who favors way W. Through the replicating of the data, normalized data models provide faster query time, which led us to our choice.

MyWay

Phase 2: Design Documentation
Donglai Wei, Shidan Xu, Neil Gurram

Programming

1. Basic coding
   - functionality: We implemented two main functionalities
     - Finding a way based on the query
     - Manipulating (get/add/update/del) objects in the database
2. Modularity
   - We used a ruby-on-rail style of code management:
     - /api:
       - /api/app: MVC components (/models, /views, /controllers)
       - /api/config: configuration files for app parameters
     - /test: Qunit tests
     - /public: assets accessible from the html (/js, /stylesheets, /txt, /img, /csv)
     - /preprocess: off-line codes to prepare the data for way finding
3. Verification:
   - Unit tests: We provide reproducible Quint tests in /test folder

Design

1. Design model (Data design):
   - Applied design moves to the old Data Model from proj 3.1:
     - removing object (Figure 1): We removed the “Traveller” object, as we decided to focus on the “way-finding” functionality and we removed the social-network functionality (which included review a way, save a way)

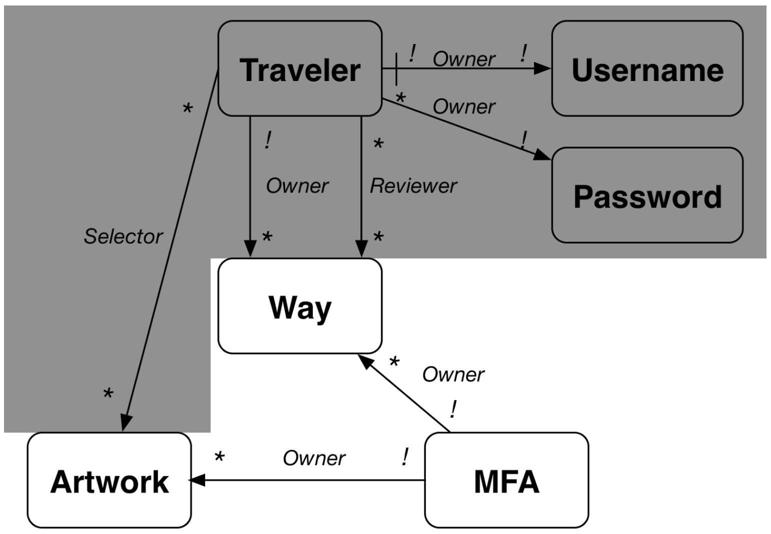


Figure 1. design move: removing objects.

- renaming objects: We renamed our “MFA” object into “Map” object, as we wanted to simplify the problem of finding ways on one floor of “MFA” object.
- adding objects: We added the “gallery” and “road” object.
    - “gallery”: Previously, we designed “artwork” objects to have their own physical locations. To simplify the “Traveling Salesman Problem” to find a way (to be implemented later), we only cared about the location of the gallery that the artworks are in. Also, with the new “gallery” object, we can easily add/update/del all the “artwork” objects that are in the same physical gallery
    - “road”: Not all “gallery” objects are connected to each other. In order to find a feasible way, we need the “road” object to record the connection between pairs of galleries.

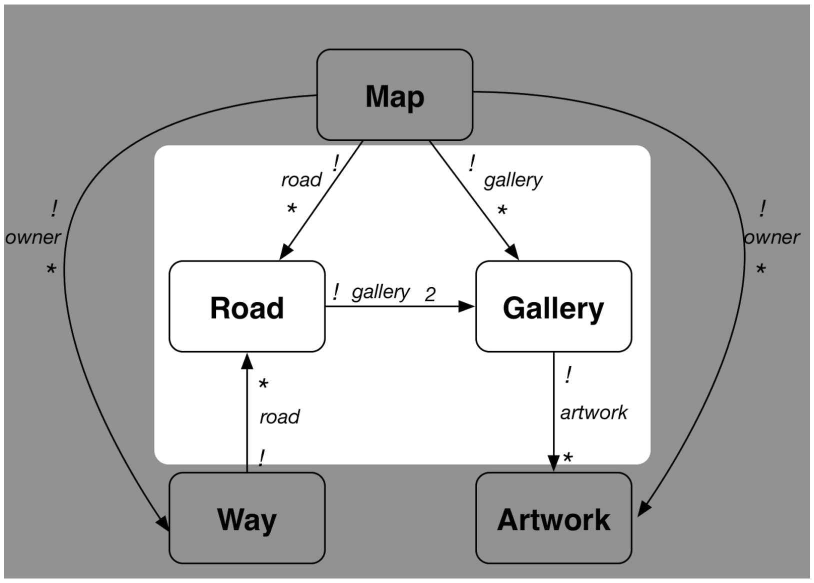


Figure 2. design move: adding objects.

- Contours of the Data Model:
    - Gallery: a gallery contains a list of artwork ids
    - Road: a road contains two gallery ids
    - Way: a way contains a list of road ids
    - Map: a map contains a list of road ids and a list of gallery ids

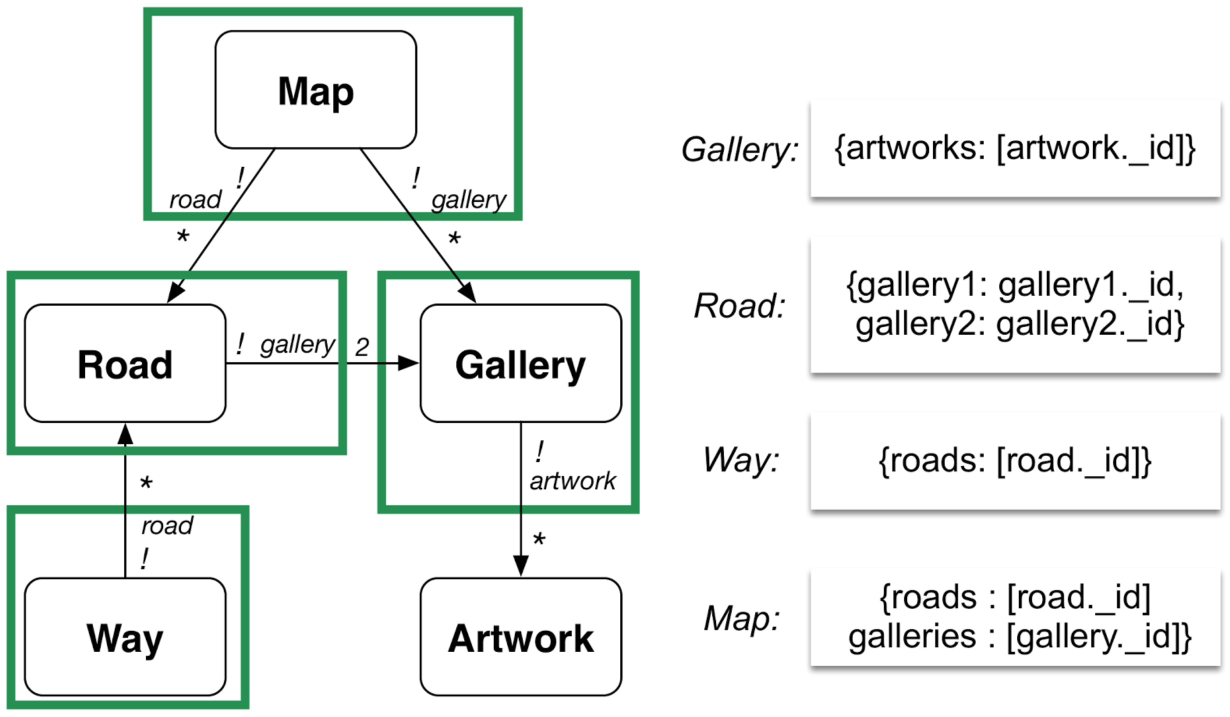


Figure 3. contours on the final data model and the implementation.

2. Design challenges:

- Challenge 1: How to design objects to enable route planning?
    - our choice: We needed to represent the graph of artworks, so that we can run the use “Traveling Salesman Problem” solver to output the shortest path to cover the artworks in the query. Thus, we add “gallery” (node in the graph) and “road” (edge in the graph) object into our data model.
- Challenge 2: How to represent the location information of the artworks?
    - choice 1: every “artwork” object has the location information
    - choice 2: every ”artwork” object is contained in a “gallery” object that has the location information
    - our choice: We choose option 1 to simplify the “Traveling Salesman Problem” to find a way (to be implemented later)
- Challenge 3: How to design the a “map/way/road” object?
    - choice 1: light-weight (normalized), only contains references to the contained objects
    - choice 2: heavy-weight (embedded), contains the associated objects themselves
    - our choice: We choose option 1, as we want a normalized data model where we can easily update individual objects without much side effects.
- Challenge 4: Whether to include the “Traveler” object?
    - choice 1: yes. implement save/review “way” functionality for travellers

        - choice 2: no.
        - our choice: We choose option 2, as we want to focus on the purpose of the app, which is to provide an interface for users to find a way based on the artworks that they are interested in.

Application Programmer's Interface:

1. Find a way based on the url query artworks
    1.1. Currently implemented as a dummy method, will return just a way regardless of input.
    1.2. This method can also be found in the more detailed API below in Way.

- **Way**
    - **POST**:
        - **/mfa/way/search/artworks_id/**
            - Request body:
                - - ids: the list of artwork ids
            - Response:
                - success: true if successfully found a way connecting all artworks minimizing total distance
                - content: json file of the way
                - err: error message

2. Manipulate (get/add/update/del) objects in the database. There are 2 show methods, 2 delete methods, 1 update method, and 1 delete method for Artwork, Gallery, Way, Map. When there are two methods, one refers to the specific (show a particular artwork) and the other refers to the general (show all artworks). Road has all but the update method, as updating the edge involves a remove and an add, which feels more natural to us to be called separately).

- **Artwork**:
    - **GET**:
        - **/mfa/artwork/id=...**
            - Request parameters:
                - id: a string representation of the desired artwork's _id
            - Response:
                - success: true if this artwork is in database
                - content: json file of the requested artwork
                - err: error message
        - **/mfa/artwork/**
            - Request parameters:
                - null
            - Response:
                - success: displays all artworks in the database
                - content: json file of all artworks

            - err: error message
        - **/mfa/artwork/delete**
            - Request body:
                - null
            - Response:
                - success: true if successfully deleted all artworks
                - content: null
                - err: error message

    - **POST**:
        - **/mfa/artwork/new/**
            - Request body:
                - id: the string representation of the ID of the artwork
                - access_number: the access number of the artwork
                - gallery: the room number of the gallery the artwork is in
            - Response:
                - success: true if new artwork created
                - content: null
                - err: error message
        - **/mfa/artwork/update/update**
            - Request body:
                - content: content of the fields to be changed (other fields can be null, id is required)
                    - id: the _id of the artwork in question
                    - name: the new name for the artwork
                    - access_number: the new access number
            - Response:
                - success: true if successfully updated the artwork
                - content: json file of the modified artwork
                - err: error message
        - **/mfa/artwork/delete/id**
            - Request body:
                - id: the _id of the artwork to be deleted
            - Response:
                - success: true if successfully deleted artwork with given id in database
                - content: null
                - err: error message

- **Gallery**:
    - **GET**
        - **/mfa/gallery/**
            - Request parameters:

            - null
        - Response:
            - success: true if galleries in the database
            - content: json file of all galleries
            - err: error message
    - **/mfa/gallery/delete**
        - Request parameters:
            - null
        - Response:
            - success: true if successfully deleted all galleries
            - content: null
            - err: error message

    - **/mfa/gallery/id=...**
        - Request parameters:
            - id: a string representation of the desired gallery's _id
        - Response:
            - success: true if this gallery is in database
            - content: json file of the requested gallery
            - err: error message

- **POST**
    - **/mfa/gallery/new**
        - Request body:
            - name: the room number of the gallery
            - x: the x coordinate of the gallery chamber
            - y: the y coordinate of the gallery chamber
            - artworks: a list of artwork in the gallery, separated by ','
        - Response:
            - success: true if new gallery created
            - content: a JSON file for the gallery created
            - err: error message
    - **/mfa/gallery/delete/id**
        - Request body:
            - id: the _id of the gallery to be deleted
        - Response:
            - success: true if successfully deleted gallery with given id in database
            - content: null
            - err: error message
    - **/mfa/gallery/update/**
        - Request body:

            - content: content of the fields to be changed (other fields can be null, id is required)
                - id: the _id of the gallery in question
                - x: the new x position of the gallery
                - y: the new y position of the gallery
                - artworks: the new list of artworks in the gallery, overwrites old one
        - Response:
            - success: true if successfully updated the gallery
            - content: json file of the modified gallery
            - err: error message

- **Road**:
    - **GET**:
        - **/mfa/road/id=...**
            - Request parameters:
                - id: a string representation of the desired gallery's _id
            - Response:
                - success: true if this gallery is in database
                - content: json file of the requested gallery
                - err: error message
        - **/mfa/road/**
            - Request parameters:
                - null
            - Response:
                - success: true if roads in the database
                - content: json file of roads
                - err: error message
        - **/mfa/road/delete**
            - Request body:
                - null
            - Response:
                - success: true if successfully deleted all roads
                - content: null
                - err: error message
    - **POST**:
        - **/mfa/road/delete/id**
            - Request body:
                - id: the _id of the road to be deleted
            - Response:
                - success: true if successfully deleted road with given id in database

                - content: null
                - err: error message
        - **/mfa/road/new**
            - Request body:
                - gallery1: the id of the first gallery chamber
                - gallery2: the id of the second gallery chamber
            - Response:
                - success: true if new road created
                - content: a JSON file for the road created
                - err: error message
- **Way:**
    - **GET:**
        - **/mfa/way/id=...**
            - Request parameters:
                - id: a string representation of the desired way's _id
            - Response:
                - success: true if this way is in database
                - content: json file of the requested artwork
                - err: error message
        - **/mfa/way/delete**
            - Request body:
                - null
            - Response:
                - success: true if successfully deleted all ways
                - content: null
                - err: error message
        - **/mfa/way/**
            - Request parameters:
                - null
            - Response:
                - success: true if ways in the database
                - content: json file of artworks
                - err: error message
    - **POST**:
        - **/mfa/way/search/artworks_id/**
            - Request body:
                - ids: the list of artwork ids
            - Response:
                - success: true if successfully found a way connecting all artworks minimizing total distance
                - content: json file of the way
                - err: error message
        - **/mfa/way/update/**

      - Request body:
        - content: content of the fields to be changed (other fields can be null, id is required)
        - name: the name of the way
        - description: Some explanation of the way
        - roads: a list of edges, in order, of the way
      - Response:
        - success: true if successfully updated the artwork
        - content: json file of the modified artwork
        - err: error message
    - **/mfa/way/delete/id**
      - Request body:
        - id: the _id of the way to be deleted
      - Response:
        - success: true if successfully deleted way with given id in database
        - content: null
        - err: error message
    - **/mfa/way/new**
      - Request body:
        - name: the name of the way
        - description: Some explanation of the way
        - roads: a list of edges, in order, of the way
      - Response:
        - success: true if new way created
        - content: null
        - err: error message
- **Map**:
  - **POST**
    - **/mfa/map/new**
      - Request body:
        - image: the image for the map
        - name: the name of the map
        - description: some description of the map
        - roads: list of roads in the map
        - galleries: list of galleries in the map
      - Response:
        - success: true if new map created
        - content: a JSON file for the map created
        - err: error message
    - **/mfa/gallery/delete/id**
      - Request body:
        - id: the _id of the gallery to be deleted

        - Response:
            - success: true if successfully deleted gallery with given id in database
            - content: null
            - err: error message
    - **/mfa/map/update/**
        - Request body:
            - content: content of the fields to be changed (other fields can be null, id is required)
                - id: the _id of the map in question
                - image: the new image for the map, overwrites old one
                - name: the name of the map
                - description: some description of the map
                - roads: list of roads in the map
                - galleries: list of galleries in the map
        - Response:
            - success: true if successfully updated the map
            - content: json file of the modified map
            - err: error message