***Design and Planning Document***

# **WithU**



**Team:** Sam Patterson, Joshua Chaimson, Pritesh Kalantri, Will Roberts, Mingren Shen, Shantanu Singhal, Jason Waner

\*Revisions are marked as highlighted\*

**Document Revision History**

*Rev. 1.0 <2017*-*10*-*08>*: initial version

*Rev. 2.0 <2017*-*10*-*13>*: 2.0 version

*Rev. 3.0 <2017-10-17>*: Changing iteration details

*Rev. 4.0 <2017-11-1>*: Iteration 1 revision

Rev.5.0 <2017-27-2017>: Iteration 2 revision

Table of Contents

[**WithU**](#_5cw8vmjmmdd6) **1**

[System Architecture](#_z9iy02vk7jnb) 3

[Design Details](#_vj4ale5gcv7h) 4

[Database Design](#_p3jgk2im7oei) 4

[Implementation Plan](#_nzof01k7e0nw) 20

[Potential Risks](#_f47on8lknbt1) 32

[Testing Plan](#_z1ztfp6icj0a) 33

## 

## 

## 

## 

## 

## System Architecture

We decided to base the architecture of our application off the MVC (Model-View-Controller) design pattern although there is some modifications. Android development naturally lends itself to this architecture due to the close relationship between Activities and the XML, such that the view and controller are almost grouped together as their own part. This is a noted drawback of the MVC architecture, however it still made the most sense for our application.

The model for our application will consist of the Firebase Real-time Database, the Firebase Authentication service, as well as Firebase Storage. Our backend classes will store various business logic needed to work with database. The Controller acts as a gatekeeper between the View and the Model. Information retrieved from the View is used by the controller when interacting with the Model, and information returned from the model is displayed properly to the View through the Controller.

The view/controller will be handled via the XML files and the Activities within Android Studio.

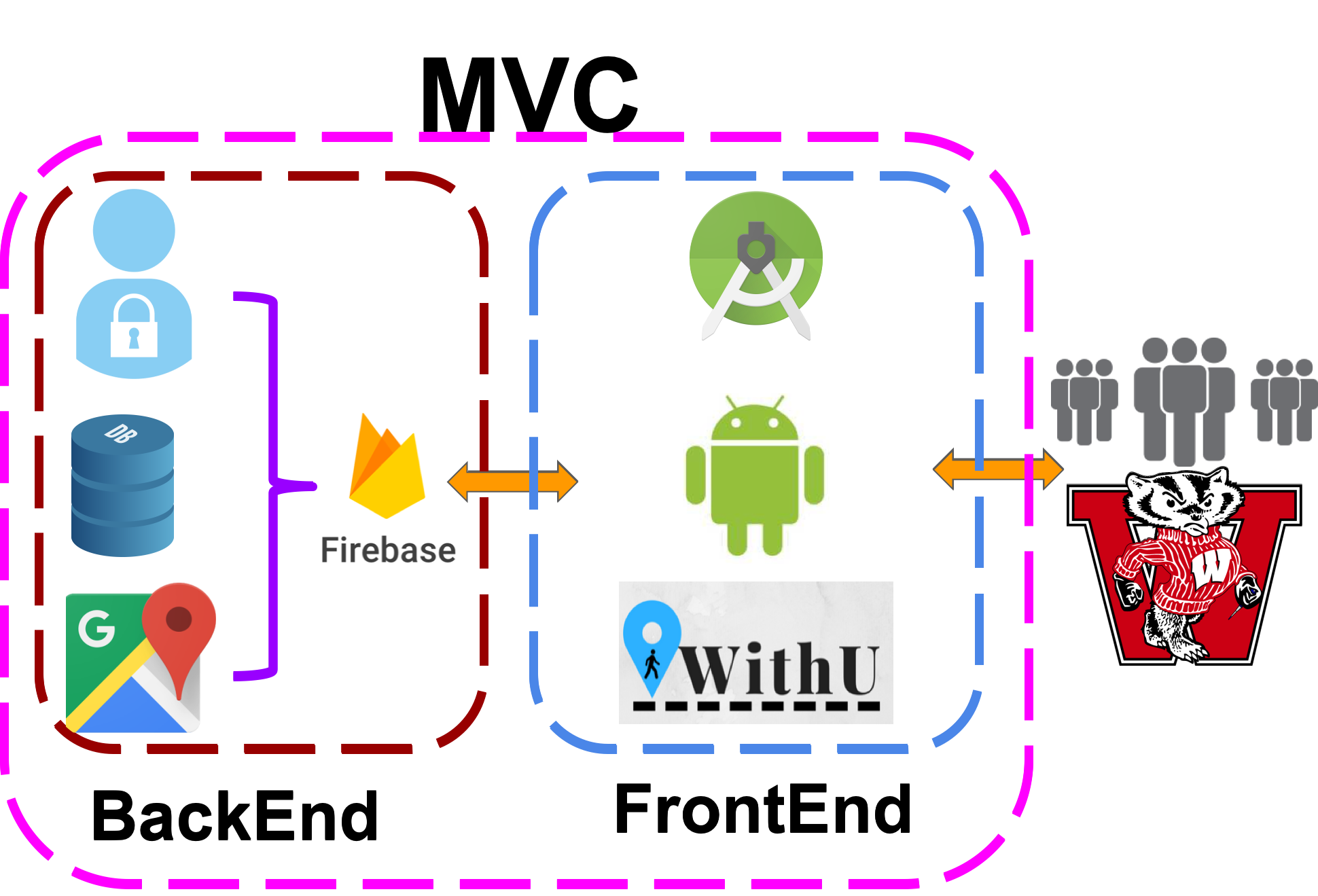
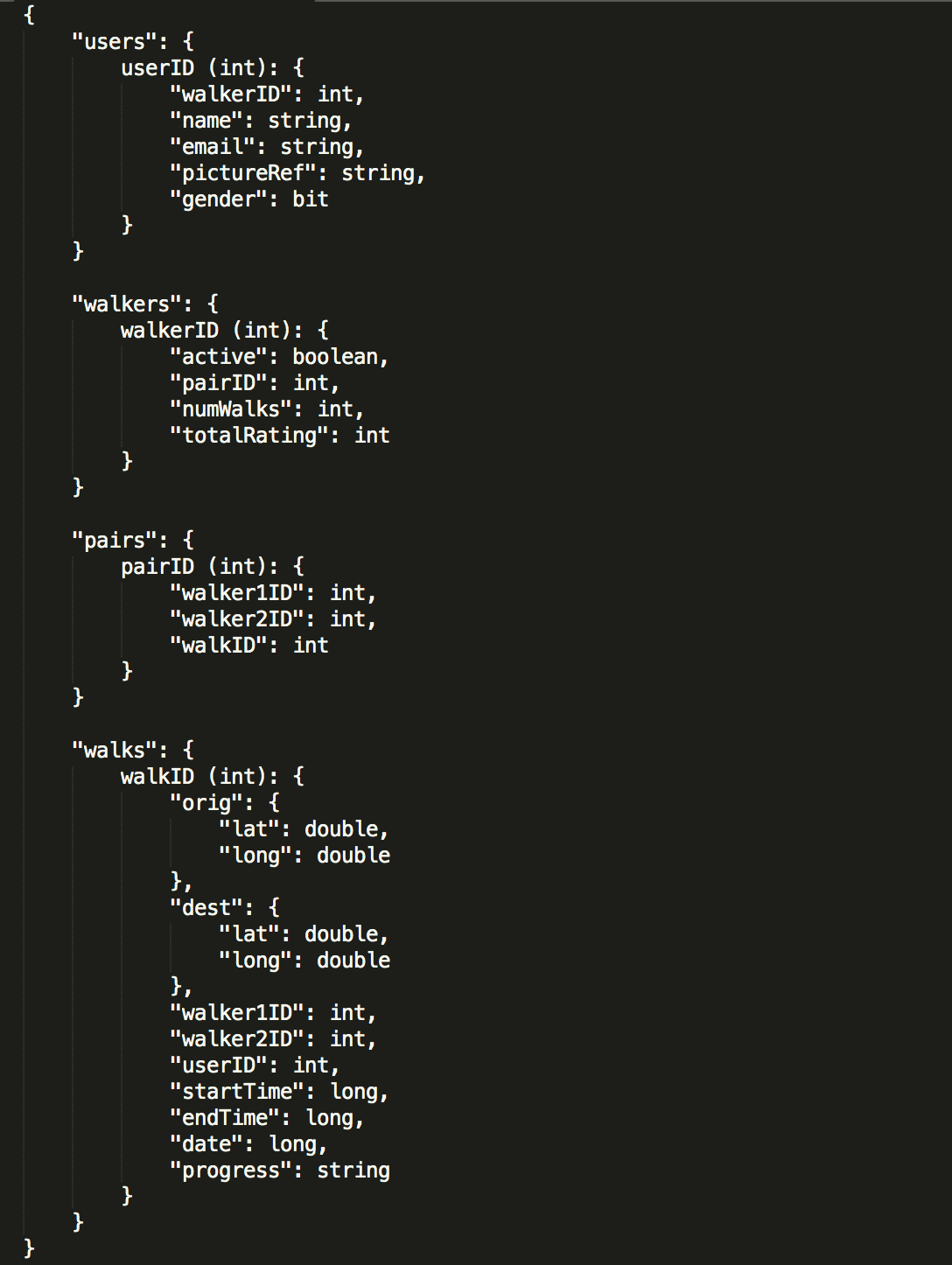


Figure 1. System layout of WithU

## Design Details

## Database Design



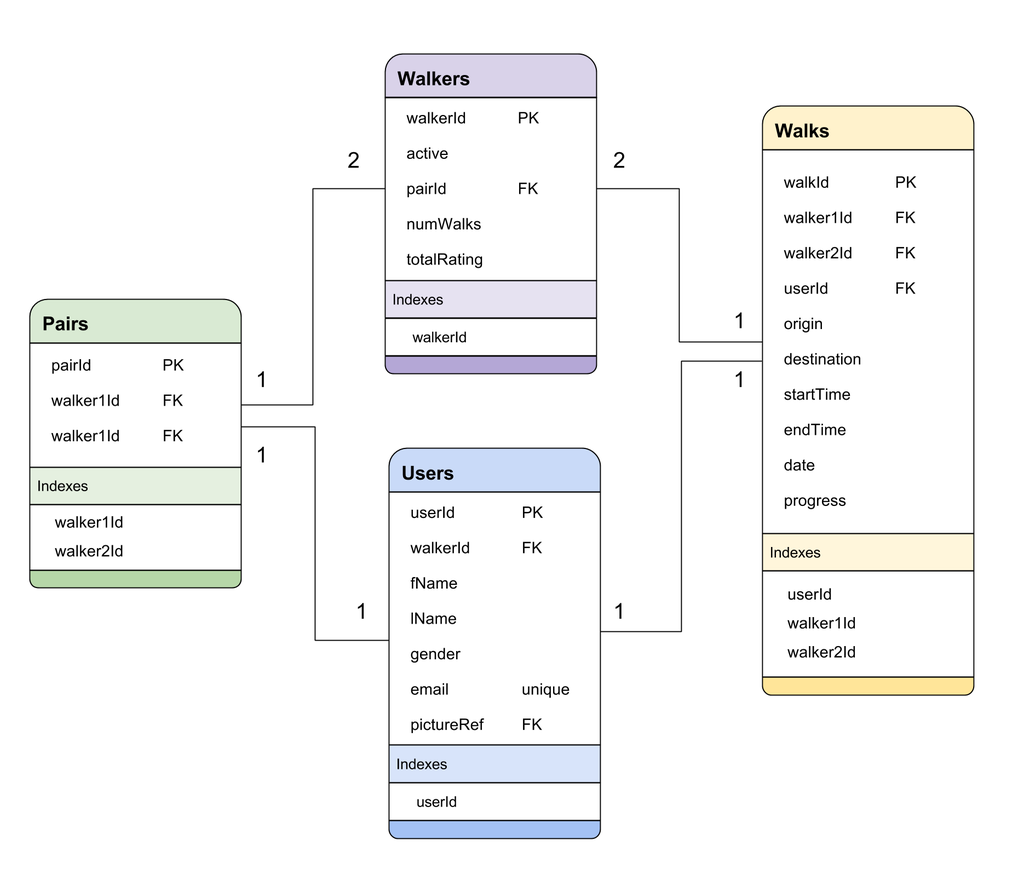


Figure 2. Database entity–relationship model

Model Object Classes

**User:** representation of the user data being stored in the database

|  |  |
| --- | --- |
| **Properties** | **Description** |
| userId | The unique ID value of the user |
| walkerId | The unique walker ID related to the user if the user is also a walker. |
| name | The first and last name of the user |
| email | The email address of the user. |
| pictureRef | The name of the file for retrieving the profile picture of the user. |

**Walker:** representation of the walker data being stored in the DB

|  |  |
| --- | --- |
| **Properties** | **Description** |
| walkerId | The unique ID value of the walker. |
| active | Boolean value representing whether or not the walker is online. |
| pairId | The unique ID of the pair to which the walker is assigned. |
| numWalks | The total number of walks that the walker has completed |
| totalRating | The accumulated rating value of the walker over all of their walks. |
| **Methods** | **Description** |
| averageRating(int numWalks, int totalRating) | Calculates the average rating of a walker from their total number of walks and their total rating. |

**Walk:** representation of the walk data being stored in the DB

|  |  |
| --- | --- |
| **Properties** | **Description** |
| walkId | Unique ID value of the walk |
| orig | Latitude and longitude of the walk origin. |
| dest | Latitude and longitude of the walk destination. |
| walker1Id | The unique ID of the first walker. |
| walker2Id | The unique ID of the second walker. |
| userId | The unique ID of the user who requested the walk. |
| startTime | The time, in milliseconds, of the start of the walk. |
| endTime | The time, in milliseconds, of the end of the walk. |
| progress | String value relating to the status of the walk. |
| **Methods** | **Description** |
| calculateDuration(startTime, endTime) | Calculates how long the walk took. |
| calculateDistance(orig, dest) | Calculates how far the walk was. |

**Pair:** representation of the pair data being stored in the DB

|  |  |
| --- | --- |
| **Properties** | **Description** |
| pairId | Unique ID value of the pair. |
| walker1Id | The unique ID value of the first walker. |
| walker2Id | The unique ID value of the second walker. |
| walkId | The unique ID value of the walk. When walkId is null, then the pair is idle. |

**Model Service Classes**

**WalkService:** class which handles all queries relating to the walk table

|  |  |
| --- | --- |
| **Methods** | **Description** |
| createWalk(Walk walk) | Add new walk to the database |
| retrieveWalksByProgress(String progress) | Retrieve all walks with the matching progress. |
| endWalk(int walkId, String progress) | Changes the walk “progress” and sets the “endTime” to the current time. |

**PairService:** class which handles all queries relating to the pair table

|  |  |
| --- | --- |
| **Methods** | **Descriptions** |
| createPair(int walkerId1, int walkerId2) | Add new pair to the database. |
| retrieveIdlePairs() | Returns a list of Pair objects where “walkId” = null |
| assignPairToWalk(int pairId, int walkId) | Sets the walkId field of the proper pair, linking the walk and the pair. |

**WalkerService:** class which handles all queries relating to the walker table

|  |  |
| --- | --- |
| **Methods** | **Descriptions** |
| createWalker(Walker walker) | Add new walker to the database. |
| retrieveWalker(int walkerId) | Returns Walker object from the database. |
| retrieveActiveWalkers() | Returns a list of Walker objects where “active” = true |
| retrieveSoloWalkers() | Returns list where “active” = true and “pairId” = null |

**UserService:** class which handles all queries relating to the user table

|  |  |
| --- | --- |
| **Methods** | **Description** |
| createUser(User user) | Add new user to the database. |
| retrieveUser(int userId) | Returns User object from the database |
| retrieveUserByWalkerId(int walkerId) | Returns User object from database |

**LoginManager:** class which handles all queries relating to the user table

|  |  |
| --- | --- |
| **Methods** | **Description** |
| signIn(String email, String password) | Makes call to Firebase Authentication service to try to sign in the user |
| createUser(String email, String password) | Makes call to Firebase Authentication service to try to create a new user |
| logout(String email) | Makes call to Firebase Authentication service to try to log the user out |

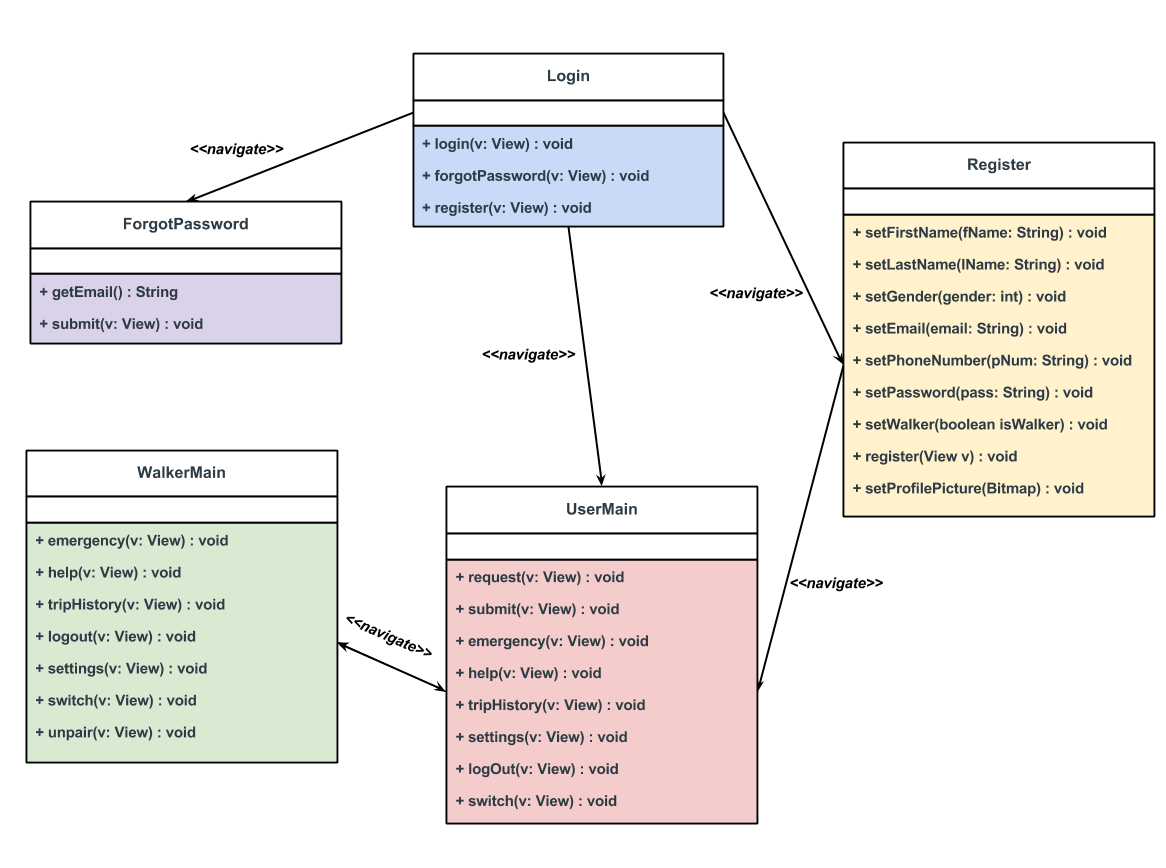


Fig 3. Overall System Class Relationship

**Controller Activity Classes（Frontend）**

**Login.java:** class that handles login page functionality

* public void login(View view)
  + Calls signIn(String email, String password) from the LoginManager class and directs the user to the user main page.
* public void forgotPassword(View view)
  + Directs the user to the forgot password page.
* public void register(View view)
  + Directs the user to the registration page

**Register.java:** class that handles registration functionality

* public void setFirstName(String firstName)
* public void setLastName(String lastName)
* public void setGender(int gender)
* public void setEmail(String email)
* public void setPhoneNumber(String phoneNumber)
* public void setPassword(String password)
* Public void setWalker(boolean isWalker)
* public void register(View view)
  + Directs the user to the user main page
* Public void setProfilePicture(Bitmap)

**ForgotPassword.java:** class that handles functionality of forgot password page

* public String getEmail()
* public void submit(View view)
  + Directs the user to the user main page

**UserMain.java**

* public void request(View view)
  + Directs the user to the submit request pop-up
* public void submit(View view)
  + Directs the user back to the user main page
* public void emergency(View view)
  + Directs the user to an emergency confirmation pop-up
* public void help(View view)
  + Directs the user to the help section of the application
* public void tripHistory(View view)
  + Directs the user to their trip history page
* public void settings(View view)
  + Directs the user to the account information page
* public void logOut(View view)
  + The user will be logged out of the application
* public void switch(View view)
  + Directs the user to the walker main page

**WalkerMain.java**

* public void emergency(View view)
  + Directs the walker to an emergency confirmation pop-up
* public void help(View view)
  + Directs the walker to the help section of the application
* public void tripHistory(View view)
  + Directs the walker to their trip history page
* public void logout(View view)
  + The walker will be logged out of the application
* public void settings(View view)
  + Directs the walker to the account information page
* public void switch(View view)
  + Directs the walker to the user main page
* public void unpair(View view)
  + Directs the walker to the confirm unpairing pop-up

**Start Up Screen**

|  |  |  |
| --- | --- | --- |
| Content | Component | Description |
| Register | Button | This button is used to take the user to the register screen. |
| Log In | Button | This button is used to take an already registered user to the Login screen. |

**Log In Screen**

|  |  |  |
| --- | --- | --- |
| Content | Component | Description |
| Email Address | Text Field | This field is for the user enters their **Wisc.edu email address.** |
| Password | Text Field | This field is for the user to enter their password to log in. |
| Remember Me | Check Box | This checkbox indicates whether the app saves the user’s account and password for auto-login |
| Log In | Button | When pressed, the app verifies that **Wisc.edu email address** and **Password** are valid and then goes to the users main page . If not valid, Invalid Information dialogue is displayed. |
| Forgot Password? | Button | When pressed, the server sends a recovery email to the registered **Wisc.edu** email address. |
| Register | Button | When pressed, goes to the corresponding page to create a new account. |

## 

## 

**Forgot Password**

|  |  |  |
| --- | --- | --- |
| Content | Component | Description |
| Email Address | Text Field | This field is for the user to enter their wisc.edu email address. |
| Submit | Button | When pressed, the email address will be verified to be a valid email address. Once verified, the password will be sent to the corresponding email address. |

## 

**Register**

|  |  |  |
| --- | --- | --- |
| Content | Component | Description |
| First Name | Text Field | This field is for the user to enter their first name. |
| Last Name | Text Field | This field is for the user to enter their last name. |
| Gender | Radio Button | This field is for the user to select their appropriate gender. |
| Email Address | Text Field | This field is for the user to enter their wisc.edu email address. This field must be a **Wisc.edu** email address**.** |
| Phone Number | Text Field | This field is for the user to enter their personal phone number. This number will be used to contact the user throughout the walking process. |
| Password | Text Field | This field is for the user to create a password that will later be used to verify the user upon logging in. |
| Re-enter Password | Text Field | This field is for the user to re-enter their desired password. This helps prevent against any errors in the desired password. |
| Register | Button | When pressed, the information entered by the user will be stored into the Firebase system. |
| Walker | Check Box | When checked, the user will also be registered to be a walker. |
| Terms of Use | Check Box | a walker must agree to in order to use our mobile app. |

**User Main Page**

|  |  |  |
| --- | --- | --- |
| Content | Component | Description |
| Map | Google Map Object | This object is used to show the user the map. The user can then interact with the map by moving it and zooming in and out.  For the walking, the map should display (1) tell the user where is the walker and how long should they wait.(2) Tell the walking team how to go to the client’s destination. |
| Request | Button | The user will select the request button and input their destination. |
| Search | Text Field | The user will input their desired destination. |
| Submit | Button | This button is used to send their desired location to the walkers. |
| Current Location | Button | The google map api indicates the current location of the **user.** |
| Where To | Search Bar | This is where the user can select their their desired destination. |
| Terminate Button | Button | The button is used for terminating the walk. |
| Let’s Go | Button | This button is used to send a notification to the walkers to start the walk. |
| Menu | Button(Toggle Switch) | Refer to **User Side Menu below.** |

**Walker Main Page**

|  |  |  |
| --- | --- | --- |
| Content | Component | Description |
| Map | Google Map Object | This object is used to show the user the map. The user can then interact with the map by moving it and zooming in and out.  For the walking, the map should display (1) guide the walker approaching their client (2) Tell the walking team how to go to the client’s destination. |
| Current Location | Button | The google map api indicates the current location of the **Walker.** |
| Menu | Button(Toggle Switch) | Refer to **Walker Side Menu below.** |

**Walker Invite Notification**

|  |  |  |
| --- | --- | --- |
| Content | Component | Description |
| Decline | Button | If pressed, the walker will have declined the request to pair with the requesting partner. This walker is required to pair up before able to accept any walk request. |
| Accept | Button | If pressed, the walker will have accepted to pair with the requesting partner. |

**User Request to Walker**

|  |  |  |
| --- | --- | --- |
| Content | Component | Description |
| Decline | Button | If pressed, the walker will have declined the request to walk with the requesting user. |
| Accept | Button | If pressed, the walker will have accepted to walk the requesting user to their desired destination. |

**Verify Meeting Notification**

|  |  |  |
| --- | --- | --- |
| Content | Component | Description |
| Decline | Button | If pressed, the walker is acknowledging that the user was not at their described location. The walker will then be prompted to confirm this decision. |
| Accept | Button | If pressed, the walker is acknowledging that the user was at their described location. |

**User Side Menu**

|  |  |  |
| --- | --- | --- |
| Content | Component | Description |
| Help | Menu List Button | Simplest tutorial of how to use our app for the user. |
| Your trip history | Menu List Button | Display the walking history by dates. |
| Settings | Menu List Button | Account Information that the user can update like uploading a new user image. |
| Log out | Menu List Button | Logout the app and close it. |
| Switch to Walker | Menu List Button | Change the role from the user to the walker and the screen goes to the walker main page. |
| Emergency | Button | When pressed, directs the user to the phone application to dial 911. |
| Report | Button | Allows user to report a walker |

**Walker Side Menu**

|  |  |  |
| --- | --- | --- |
| Content | Component | Description |
| Help | Menu List Button | Simplest tutorial of how to use our app for the walker. |
| Your trip history | Menu List Button | Display the walking history by dates. |
| Settings | Menu List Button | Account Information that the user can update like uploading a new user image. |
| Log out | Menu List Button | Logout the app and close it. |
| Switch to User | Menu List Button | Change the role from the walker to the user and the screen goes to the user main page. |
| Unpair | Menu List Button | When pressed, it will create a pop up button to confirm you want to unpair with your partner. |
| Emergency | Button | When pressed, directs the walker to the phone application to dial 911. |

**User Completed Walk**

|  |  |  |
| --- | --- | --- |
| Content | Component | Description |
| Rate | Button | This button is pressed after the user makes it to their destination |
| Complete | Button | Pressed when the user is done with their walk. Then they are taken back to the user main page. |

**Walker Completed Walk**

|  |  |  |
| --- | --- | --- |
| Content | Component | Description |
| Complete | Button | Pressed when the user is done with their walk. Then they are taken back to the walker main page. |

**Error Page**

|  |  |  |
| --- | --- | --- |
| Content | Component | Description |
| Error Hints | Image View | To simplify, we group all errors into one image description saying ”Something seems wrong, do not worry, we are working on it!”  And send out error message to the server and the server will email the administer and developing team. |

## 

## Implementation Plan

To implement and create our application we have split the tasks up and assigned them to everyone on the team. There are many tasks that have two people assigned to them, and this is because Sam and Jason are primarily focused on the backend of the application. Which means that they will take part in the tasks that deal with accessing data that will be used on the front end of the application.

The main plan for iteration 1 is to focus on the main core features to get the app functional. Some of the ones planned for this first go is the signup and login pages as those need to be done first in order to get people registered in our system. Then the other main part is getting the request walk/directions/etc. (i.e the overall walk functionality) minus the bells as whistles for part 1. In future iterations we will go into more depth for the walking functionality such as rating walkers, emergency buttons, reporting, etc. But obviously none of that can be implemented until the basic sign up and walk functionality works. Regarding limitations for our application, the biggest worry would be not being able to get the basic walk functionality working fast enough. As this is the backbone to our whole app we need to get this part working as fast as possible so we are able to work on other features that go off of that. Not being able to get this part done by the first iteration will limit our development going forward.

As for proposed risks there aren’t really many of them but the one main risk is making our UI too similar to that of Uber/other navigational apps. The goal is to take into consideration what those navigation apps do well but spin it in our own way at the same time. We also have risk of accessing to much location data from the user. This is always a topic of discussion on privacy. The application has to store and read constant location data from the user to get the application working. Another risk associated with the application is that currently we do not have a email verification setup. This could lead to fraud email accounts using the “@wisc.edu”. This is a security risk that will need to be set up in a later iteration.

|  |  |
| --- | --- |
| Task | Sign Up |
| Description | Create a screen that allows new users to type in profile information & login/password creation. |
| Difficulty | 2 |
| Dependencies | Log In Screen must be completed first |
| Time Units | 3 |
| Assigned To | Pritesh and Sam |
| Iteration | 1 |

|  |  |
| --- | --- |
| Task | Log In |
| Description | Create a screen that allows users to type in their login/password combination. Firebase Authentication will be used to check if the user exists in the system. |
| Difficulty | 2 |
| Dependencies | N/A |
| Time Units | 3 |
| Assigned To | Josh and Sam |
| Iteration | 1 |

|  |  |
| --- | --- |
| Task | Logout |
| Description | Button pressed that will sign out the current user |
| Difficulty | 2 |
| Dependencies | Log In screen must be completed |
| Time Units | 2 |
| Assigned To | Shantanu and San |
| Iteration | 1 |

|  |  |
| --- | --- |
| Task | Switch from Walker->User or User->Walker |
| Description | Create an option on the side menu bar where the user can select “Switch to Walker” or “Switch to User” |
| Difficulty | 3 |
| Dependencies | Side-Bar Window must be created first, and account setup for both sides on the backend. |
| Time Units | 3 |
| Assigned To | Myron and Jason |
| Iteration | 1 |

|  |  |
| --- | --- |
| Task | Request Walk |
| Description | Once the user logs in, they will search for their destination and request a walker pair. |
| Difficulty | 3 |
| Dependencies | Log in and main page need to be completed. Along with the backend data working. Google maps integration needs to be completed to find the destination and generate the route. Algorithm needs to be implemented in the controller which will select the most optimal route for the walk. |
| Time Units | 3 |
| Assigned To | Will and Jason |
| Iteration | 2 |

|  |  |
| --- | --- |
| Task | Accept Walk Request |
| Description | When a walker pair receives a request they will accept the request. |
| Difficulty | 2 |
| Dependencies | Users need to be set up (both walkers and users). Search for destination and route finding with google maps needs to be completed. Pair matching for the walkers needs to be completed. |
| Time Units | 3 |
| Assigned To | Pritesh and Jason |
| Iteration | 2 |

|  |  |
| --- | --- |
| Task | Decline Walk Request |
| Description | When the walkers receive a notification they will decline the request to go to a walker. |
| Difficulty | 2 |
| Dependencies | Users need to be set up (both walkers and users). Search for destination and route finding with google maps needs to be completed. Pair matching for the walkers needs to be completed. |
| Time Units | 3 |
| Assigned To | Josh and Jason |
| Iteration | 2 |

## 

|  |  |
| --- | --- |
| Task | Emergency Call |
| Description | Create a button the main walk screen that can prompt the android phone api to call 9-1-1 |
| Difficulty | 1 |
| Dependencies | Main Walk Screen must be completed for both walkers and users. |
| Time Units | 1 |
| Assigned To | Josh |
| Iteration | 3 |

|  |  |
| --- | --- |
| Task | Rate Walkers |
| Description | After the walker is complete the user can rate the walker on how well walker did at walking the user home. |
| Difficulty | 1 |
| Dependencies | The whole system of creating a user account, logging in, finding destination, requesting a walker, creating a route using google maps API, and completing the walk needs to be set up. |
| Time Units | 2 |
| Assigned To | Josh |
| Iteration | 3 |

|  |  |
| --- | --- |
| Task | Previous Walks |
| Description | Brings up a screen that shows the walk history for that user/walker. |
| Difficulty | 2 |
| Dependencies | The whole system of creating a user account, logging in, finding destination, requesting a walker, creating a route using google maps API, and completing the walk needs to be set up. As well as saving a history of walks to the user and the walkers. |
| Time Units | 3 |
| Assigned To | Jason and Mingren |
| Iteration | 3 |

## 

|  |  |
| --- | --- |
| Task | Request Partner |
| Description | When a walker signs into the application, they have to pair up with a partner before they can receive any requests for walks. |
| Difficulty | 3 |
| Dependencies | Sign up, log in, switching from user to walker, walker main page,google maps integration and backend all have to be set up before we can do this task. |
| Time Units | 3 |
| Assigned To | Will and Sam |
| Iteration | 3 |

|  |  |
| --- | --- |
| Task | Finish Walk Notification |
| Description | When a walker confirms that the user has reached their destination, the user will receive a notification that their walk is complete |
| Difficulty | 2 |
| Dependencies | Sign up, log in,main screens, walker pairing, request a walker, and google maps integration need to be set up. |
| Time Units | 2 |
| Assigned To | Shantanu and Jason |
| Iteration | 2 |

|  |  |
| --- | --- |
| Task | Start Walk Notification |
| Description | When the walker and the user meet a push notification will be sent to the walker confirming to start the walk. |
| Difficulty | 1 |
| Dependencies | Sign up, Log in, google maps integration, request a walker all need to be set up before this task can be started. |
| Time Units | 1 |
| Assigned To | Myron and Sam |
| Iteration | 2 |

|  |  |
| --- | --- |
| Task | Cancel Walk |
| Description | Create a Button where a confirmation screen is brought up asking if the walker wants to cancel the walk. |
| Difficulty | 2 |
| Dependencies | Main walk screen must be created, google maps integration, request a walk, pair the walkers, and the back end need to be completed. |
| Time Units | 3 |
| Assigned To | Josh and Jason |
| Iteration | 2 |

|  |  |
| --- | --- |
| Task | Out of Bounds Request |
| Description | If the user or the walker deviates from the guided path, the app will push a notification telling the user they have deviated from their path. |
| Difficulty | 3 |
| Dependencies | Sign up, log in, main screen, google maps integration, pair walker, request walker, guided route, and the back end need to be completed. |
| Time Units | 3 |
| Assigned To | Shantanu |
| Iteration | 3 |

|  |  |
| --- | --- |
| Task | Side-Window Pullout |
| Description | Create a side-menu with various options to navigate user to other parts of the application |
| Difficulty | 2 |
| Dependencies | The main page navigation needs to be completed. |
| Time Units | 2 |
| Assigned To | Will and Pritesh |
| Iteration | 1 |

|  |  |
| --- | --- |
| Task | Account Info |
| Description | Create screen that shows the logged in user’s account information |
| Difficulty | 1 |
| Dependencies | Sign up, log in, main window, and side window. |
| Time Units | 2 |
| Assigned To | Pritesh |
| Iteration | 2 |

|  |  |
| --- | --- |
| Task | Report User/Walker |
| Description | This will allow the user and the walker to report each other. This can be used to report bad behavior, bad service, etc... |
| Difficulty | 2 |
| Dependencies | Sign up, log in, main page, google maps integration, pair walker's, request walker, and back end need to be completed. |
| Time Units | 2 |
| Assigned To | Will and Jason |
| Iteration | 3 |

|  |  |
| --- | --- |
| Task | Auto-Center of Map/Current Location |
| Description | This button on the main page will allow the user to center on their position on the map. |
| Difficulty | 1 |
| Dependencies | Sign up, log in, main application screen, and google maps integration. |
| Time Units | 1 |
| Assigned To | Josh |
| Iteration | 1 |

|  |  |
| --- | --- |
| Task | Partner Meetup |
| Description | When the partners agree to meetup, our app will lead the male walker to meet the female walker. Then a map will shown on the main screen with detailed direction guidings to help two walkers meet. |
| Difficulty | 3 |
| Dependencies | Sign up, log in, switch from user to walker, google maps integration, partner pairing, and backend. |
| Time Units | 2 |
| Assigned To | Shantanu and Sam |
| Iteration | 2 |

|  |  |
| --- | --- |
| Task | Go to Help link |
| Description | A small 4 pane slideshow with mockups which will help the user to understand how the app is used for both walkers and users. |
| Difficulty | 1 |
| Dependencies | Sign Up, Log In, User main page, side menu all need to be completed. This is an additional functionality. |
| Time Units | 2 |
| Assigned To | Pritesh and Shantanu |
| Iteration | 3 |

|  |  |
| --- | --- |
| Task | Call the user |
| Description | The walker can’t find the user at the desired location, so the walkers can call the user. |
| Difficulty | 2 |
| Dependencies | Sign Up, Log In, User main page, side menu all need to be completed. This is an additional functionality. |
| Time Units | 2 |
| Assigned To | Josh |
| Iteration | 2 |

|  |  |
| --- | --- |
| Task | Terms of Use |
| Description | All rules by which one must agree to abide in order to use WithU. |
| Difficulty | 2 |
| Dependencies | Sign Up is needed to be completed. This is an additional functionality. |
| Time Units | 2 |
| Assigned To | Mingren |
| Iteration | 2 |

## Potential Risks

Some of the risks that our group sees with this design are scalability, platform support, and usage. For scalability, our application needs to be ready in case our user base grows exponentially. Whenever creating a new application, it is impossible to judge whether your application will grow at a steady rate or simply take off. When an application grows at a fast rate, this increases the risk for errors and potentially crashing the application. Our group needs to make sure that the application is ready to support a large user base without losing its functionality.

For platform support, our group is taking a risk by supporting back to Android 4.0 OS (ice cream sandwich). Having an application that supports more platforms improves in its chances for success, but it comes with a much higher cost. With this note, another problem is that our application only supports Android and this time. With plans for supporting IOS in the future, this is putting the application at risk for a poor launch.

In addition, for usage, is the concern that UW students will not make use of the application. The overall goal is to replace SAFEwalks’ lengthy process for requesting a walker. However, since this service is rarely used, it increases the risk that our application goes unnoticed. Also, branching off of the previous risk of platform support, our usage will be significantly less knowing that the IOS operating system is widely used throughout our target users.

Another risk comes from the service itself. To to be simple, we are planning to implement a safewalk version of Uber. So we have encountered the same risks encountered by Uber. For example, walker’s background check is very important for a service like WithU. We must ensure the safety of our users. By now, we just plan to make sure this app does not grow outside UW-Madison, so that we can better monitor the walker’s background. So we require all users and walkers must use Wisc Email to register WithU. Also, we make sure that two walkers must be grouped before they can accept request from a user.

The problem also links with liability issues. Should WithU accepts responsibility when the walkers misbehave? We plan to add a legal terms of use into WithU. When you download our app, you agree to terms and conditions by default, including the fact that WithU absolves itself of anything that happens to you—be it an accident, injury, theft, physical attack, rape or death. Another potential risk may be that we store our user’s walking history which requires us to better protect the privacy of our users. By now, we use Google Firebase custom defined Security & Rules to develop our data security features.

If the service grows fast enough, We plan to further seek help from UW-Madison and The Wisconsin Alumni Research Foundation (WARF) as they can provide more professional legal service for us.

## Testing Plan

When we are creating our application we will be carrying out many tests to make sure our application functions correctly. Because our application deals with human interaction through location services, having a way to automatically tests these situations is crucial for our success of our application. This will save time and help improve our process of completing the application. In the end, these automated tests will have to simulate the whole process of application because it would take a lot of man hours to fully test out the system without these automated tests.

**Unit Testing**

Throughout the process of developing our application we are going to use the unit tests we created before to make sure our application is functioning correctly. Josh, Will, and Jason will create the unit tests for the front and back of the applications. Android studio recommends that we run our unit tests on our local development machine because it saves us time not having to load our application on a device or an emulator. As we start testing our specific code blocks, we are able to use mocking frameworks in android studio to make sure we are testing specific areas of code without any other dependencies in the code. Unit testing will allow us to fix bugs right away, and make our code work as intended.

**Integration Testing**

After we are done with our specific unit tests, we will start to integrate the separate units and make sure they work together. Mingren, Pritesh, Shantanu and Sam will lead and create tests for the integration phase. Our starting point for our integration tests will be if we can get the backend working with the frontend of the application. From there we are going to make sure all the units we have planned for iteration 1 are going to work together. For example, we will have to make sure that when a user requests walkers that we return an active pair of walkers. Integration testing will take a while to complete, but it is very important that we are able to successfully test and integrate the many methods and classes we will have.

**System Testing**

Once our integration testing is completed, we will start to combine all of our units, and do system testing. Android studio has many tools to help us with system testing. To start testing with android studio we can load our application on a device or use the built in emulator. Everyone in the group will help and develop the different tests for system testing. We can build system tests by using android studio espresso test recorder to run through our application using preset set of commands. This is a great tool because we won’t have to write any test code to interact with the UI of our application, we can simulate how a user would interact with the system. Another great aspect of espresso test recorder is that we can use firebase test lab for android. This will allow us to test our application across many different types of android devices and configurations. This is a great feature of firebase because it allows us as developers to test on devices we wouldn’t be able to get our hands on. This will help us build a stable and reliable application for our users. Another tool we are able to use is called UI/Application Exerciser Monkey. This tools stress tests our UI/Application in a random way. All of these tools android studio provides will make sure we are developing an application where all of our units work together.

**Regression Testing**

After we finish our first iteration, we have to make sure when adding new features we do not break our application. To make sure this doesn’t happen, android studio has a monkeyrunner tool that will allow us to compare output screen shots with ones we know are correct. This will allow us to organize the different features of our application and add them as we see necessary when creating our application.