KASPER – ID1 PROJECT DOCUMENTATION

**Software Development Team:**

Project Manager: Tushita Patel

Dev Lead: Dylan Prefontaine

Test Lead: Jeremy Liau

Build Manager: Christopher Mykota-Reid

Developers: Gaurav Arora, Arianne Butler, Haotian Ma, Kristof Mercier, Melody Zhao

Test Team: Christopher May, Ryan Tetland

Contents

[1.0 Requirements Document 3](#_Toc474085170)

[1.1 Requirements ID-1: 3](#_Toc474085171)

[1.2 Platform Description (iOS, Android, and Web Browser) 4](#_Toc474085172)

[1.3 Mini Milestones for ID1 4](#_Toc474085173)

[1.4 GUI Mock Ups: 5](#_Toc474085174)

[1.5 Use Cases: 7](#_Toc474085175)

[1.6 Sequence Flow Diagrams: 11](#_Toc474085176)

[2.0 Design 13](#_Toc474085177)

[3.0 Testing Plan 14](#_Toc474085178)

[4.0 Test Report 15](#_Toc474085179)

[5.0 Coding Style Guide 15](#_Toc474085180)

[6.0 Build Report 15](#_Toc474085181)

[7.0 Defect Report 17](#_Toc474085182)

[8.0 Upcoming Requirements 17](#_Toc474085183)

[8.1 ID2 User Requirements: 17](#_Toc474085184)

[8.2 Back-end System Requirements 19](#_Toc474085185)

[8.3 Future Requirements (Nice to Haves) 19](#_Toc474085186)

[9.0 Triage Meeting: 20](#_Toc474085187)

# Requirements Document

## 1.1 Requirements ID-1:

* Basic User Interface
  + Buttons
  + Text boxes
  + Range sliders
  + Menus
  + Links between screens
  + Image swipe feature
* Generate Dummy Data Structures
  + User Structure:
    - Integer 🡪 ID
    - String 🡪First Name
    - String 🡪 Last Name
    - String 🡪 Email Address
    - String 🡪 Password
    - String 🡪 Salt
    - String 🡪 PhoneNum (x3)
    - The ID 2 stuff looks good
    - Future stuff looks good as well
    - Location 🡪 Location
    - DateTime 🡪 Registered
    - DateTime 🡪 LastSeen
  + Listing Structure Fields:
    - Integer 🡪 ListingID
    - Integer 🡪 ListerID
    - Float 🡪 Price
    - Integer 🡪 SquareFeet
    - Integer 🡪 Bedrooms
    - Float 🡪 Bathrooms
    - Location 🡪 Location
    - String 🡪 Description
    - Boolean 🡪 isPublished
    - DateTime 🡪 dateCreated
    - DateTime 🡪 lastModified
    - Array 🡪 Thumbnail Images
    - Array 🡪 Images
  + Location Structure (to be determined):
    - Province 🡪 Province
    - String 🡪 City
    - String 🡪 Address
    - String 🡪 Postal Code
    - Double 🡪 Latitude
    - Double 🡪 Longitude
  + Province Structure:
    - String 🡪 Name
    - Char[2] 🡪 Abbrev

## 1.2 Platform Description (iOS, Android, and Web Browser)

The Kasper Home App will be functional on Android, iOS, Windows phones, and web browsers.

For ID-1, we will be focusing on building the basic structure for mobile systems only.

## 1.3 Mini Milestones for ID1

**Dev Team:**

* Every developer familiar with Ionic2, CSS, HTML, typed Script and Angular2 ✔
* Each developer has completed at least one App Page ✔
* Unit Testing of GUI’s where possible (not much to test in ID-1) ✔
* Decide on System Architecture ✔
* Implement GUI Designs ✔

**Test Team:**

* Visual testing of the GUI’s implemented by Dev Team ✔
* All testers familiar with testing frameworks ✔
* Testing reports complete ✔

**Build:**

* Explored Travis with Ionic ✔
* Researched Django, goDaddy and server ✔
* Verify code style ✔

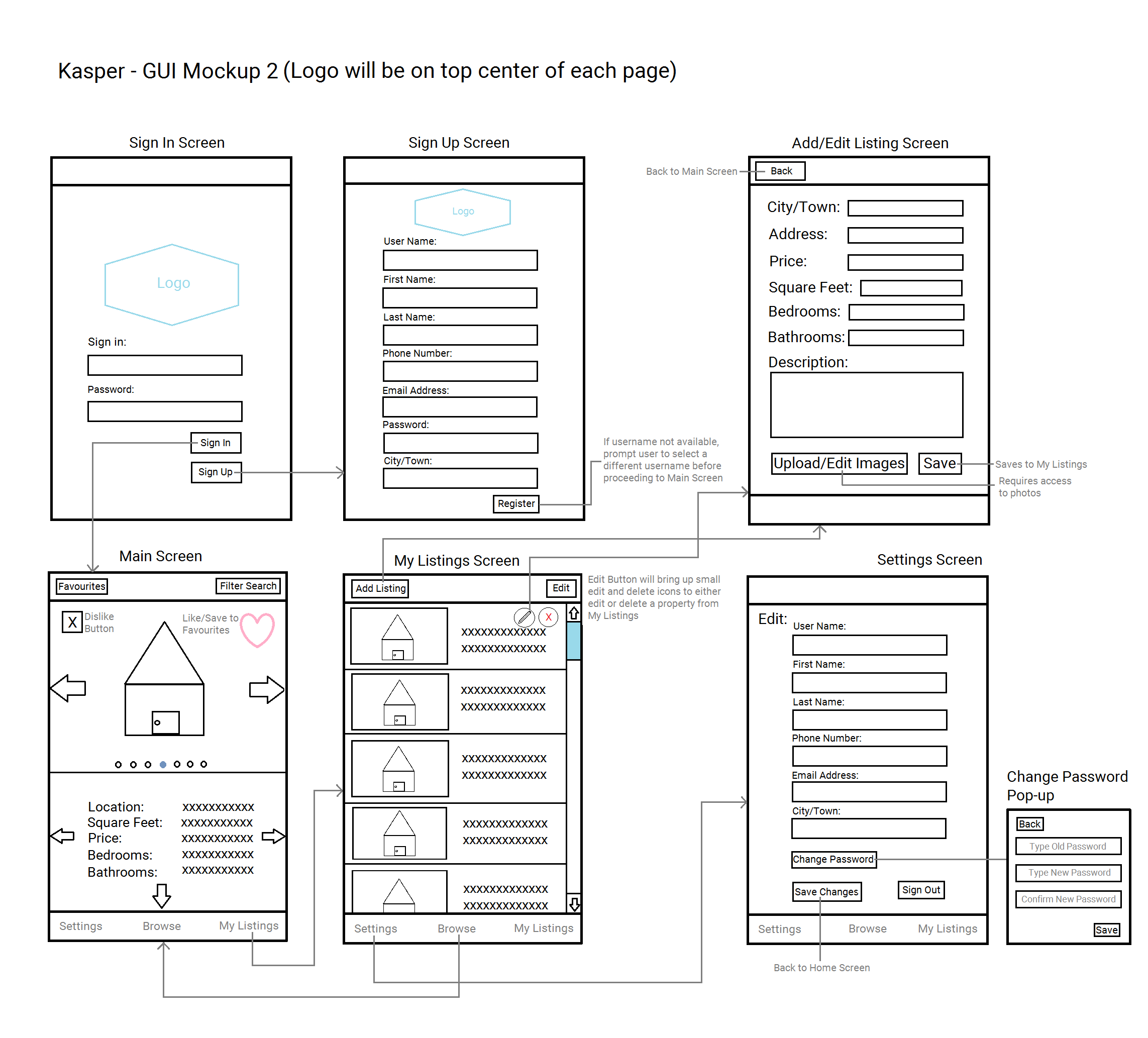
**Documents:**

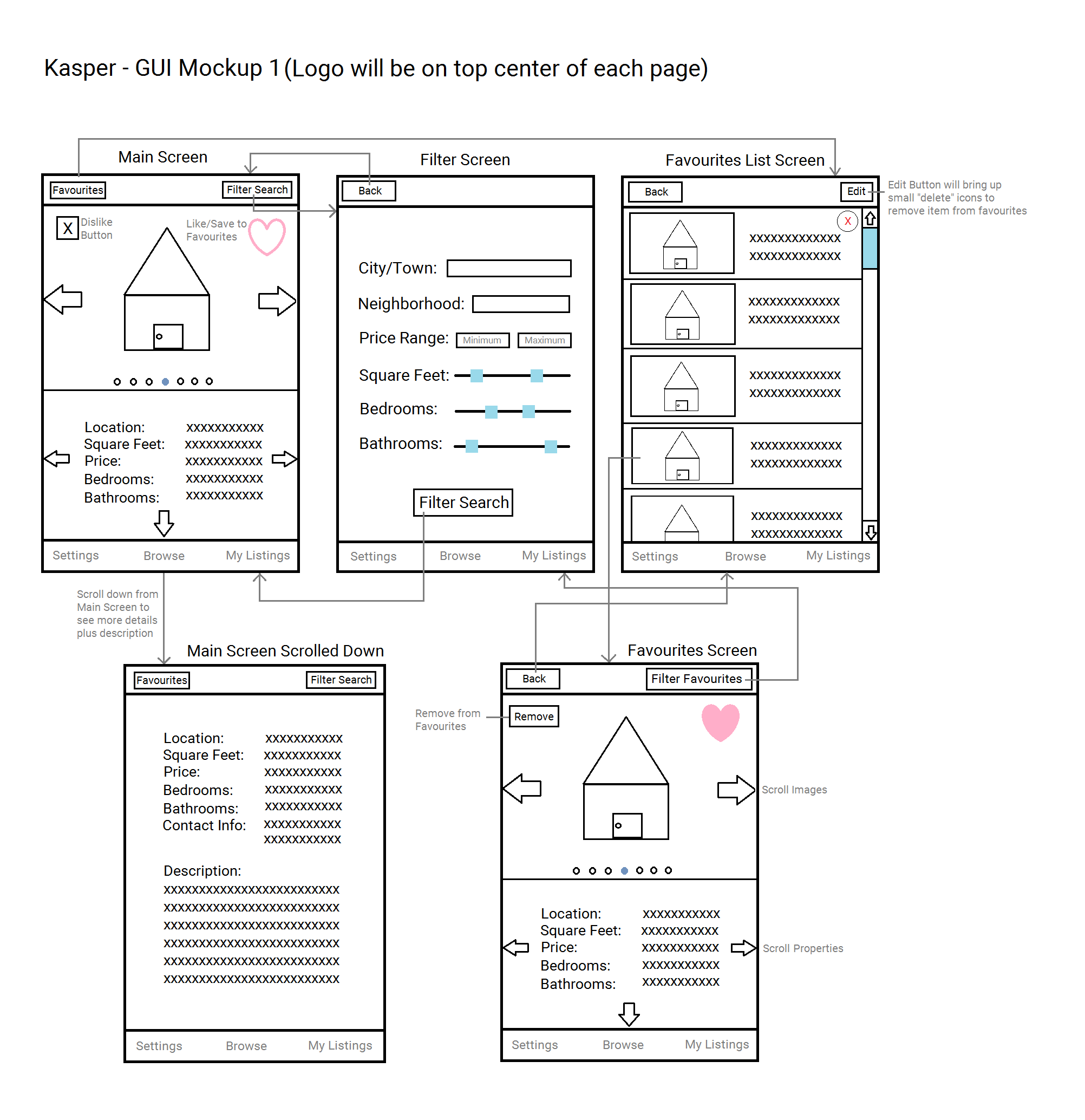
* Testing Documents ✔
* Risk Report ✔
* Requirements Document ✔
* Design Document ✔
* All documents are compiled and thoroughly edited ✔

**Other:**

* Each Developer and Tester has completed a pair programming session ✔
* All group member’s familiar with Trello, GitHub, Slack, and Google Documents ✔
* Finish GUI Design and Mock-ups for ID1 ✔
* Triage meeting ✔
* Meet with Upper Management
* Meet with client weekly ✔
* Every member familiar with the Schedule as given by project manager ✔

## 1.4 GUI Mock Ups:

GUI Mockup showing Sign In, Sign Up, Main Screen, Listings, Add/Edit Listings, and Settings

****GUI Mockup showing Main Screen (with scroll down), Filter Search, Favourites List, and Favourites

## 1.5 Use Cases:

Sign-in:

The user must enter the correct e-mail and password to enter the app. They can also choose the option to register.

* Actor(s):
  + User, Database
* Precondition(s):
  + app is open
* Basic Flow:
  1. User enters Username and password
  2. User clicks Sign-in
  3. User’s info is verified with the Database
  4. If password is correct the User is sent to Main Screen
* Alternate Flow:

4\*. Username/password is incorrect and User re-enters

* Post Condition(s):
  + The app is on the Main Screen

Sign-up:

The user must enter their information to create an account. They also have the option of going to the Sign-in page.

* Actor(s):
  + User, Database
* Precondition(s):
  + App is open
* Basic Flow:
  1. User selects sign up button
  2. User enters information and selects register
  3. The User’s information is checked if correct and added to the Database
  4. User signs in
* Alternate Flow:

4\*. There is an issue with the info and User is prompted to fix issue

5\*. User fixes info and submits info

* Post Condition(s):
  + User has an account
  + The app is on the Main Screen

Filter:

The user will click on the Filter Button to enter a specialized Listing search. They must choose which constraints to use, and the Listings will be updated accordingly.

* Actor(s):
  + User, Database
* Precondition(s):
  + User is either on the Main Screen or Favourites
* Basic Flow:
  1. User selects Filter Button
  2. User selects specific filters and clicks filter search button
  3. The database is called and the list of viewable Listings is populated.
* Alternate Flow:

2\*. User selects back and returns to Main Screen

* Post Condition(s):
  + Entries change based on filter input

Swipe through Listings:

The user will swipe right on images to view the next image for the current Listing. To view the next Listing the user will swipe the text portion of the screen. Swiping right and left will move forward and backward through the Listings respectively.

* Actor(s):
  + User, Database
* Precondition(s):
  + User is on the Main Screen
  + Listings Buffer contains five Listings loaded from the Database (relevant to design in ID-2)
* Basic Flow:
  + User swipes Left or Right
  + Next Listing is loaded from the Database
* Alternate Flow: none
* Post Condition(s):
  + Next Listing is visible on the Main Screen

Disliking or Add Listing to Favourites:

While navigating the main page, the user can indicate if they would like to not see a property again, or they can add one to their favourites list.

* Actor(s):
  + User, Database
* Precondition(s):
  + User is on the Main Screen
* Basic Flow:
  1. User selects Favourite Button on a Listing
  2. The Listing is added to user Favourites

Alternate Flow:

1.\*User selects or deselects the dislike button, changing the “disliked” field on a Listing.

* Post Condition(s):
  + Listing has either been added to favourites or the “dislike” field has been altered.

Change Settings:

To change their personal settings, the user must navigate to the Settings. They can make the necessary changes and then save them, or they can log out.

* Actor(s):
  + User, Database
* Precondition(s):
  + User is on the Main Screen
* Basic Flow:
  1. User selects Settings Button
  2. User edits Settings and clicks the Save Changes button
  3. The User’s new info is confirmed and updated in the Database
* Alternate Flow:

2\*. User clicks Sign-out button and is sent to the Sign-in Screen

* Post Condition(s):
  + Users info is successfully updated

Changing User Password:

The user may change their password by entering their old password, and two instances of their new password. This is done from the settings page.

* Actor(s):
  + User, Database
* Precondition(s):
  + User is on the Settings page
* Basic Flow:

1. User selects the Change Password button

2. User enters the old password, new password, and confirmed new password

3. User clicks submit

Post Condition(s):

* + The user has changed their password

Add Listing:

To add a Listing, a user must upload their information including a description of the house and images. They can then choose to publish a Listing once they have finished creating its profile.

* Actor(s):
  + User, Photo Directory, Camera, Database
* Precondition(s):
  + User is on the Main Screen
* Basic Flow:
  1. User selects the Add Listing Button
  2. User fills in the Listing information and may upload photos
  3. User clicks the Save Button
* Alternate Flow: none
* Post Condition(s):
  + Listing is added to My Listings and the Data Base and is made available for other users to see

Edit Listing:

While looking through their Listings, a user can choose to edit the current Listing. At this point they can then choose to publish or un-publish a Listing, or alter the Listing’s images or information.

* Actor(s):
  + User, Photo Directory, Camera, Database
* Precondition(s):
  + User is on the Main Screen and has at least one Listing in My Listings
* Basic Flow:
  1. User selects Listings button
  2. User selects Edit from the Listings Screen
  3. User submits the edited info
  4. The new Listing info is verified and then updated in the entries Database
* Alternate Flow:

3\*. User toggles the publish button

* Post Condition(s):
  + Listing modified in My Listings and the Database

Viewing and Removing Listing from Favourites:

The user can view their Favourites List by navigating to the Favourites List page.

* Actor(s):
* Precondition(s):
  + User is on the Main Screen
* Basic Flow:

1. User selects the Favourites button
2. Scrolls through their favourited properties

* Alternate Flow:

1. The user swipes left on a property and presses the delete button.

* Post Condition(s):
  + Listings are removed from a User’s favourites if applicable.

Selecting an Item from Favourites Listing:

The user may change the Main View to reflect the contents of their Favourites List. At this point the user may proceed in a similar way to the Main Page, except that Listings can now only be removed from Favourites, and not added.

* Actor(s):
  + User, Database
* Precondition(s):
  + User is on the Main Screen
* Basic Flow:
  + User selects a Listing from the Favourites List
* Alternate Flow:

1. User selects the Edit Button

2. User selects the Remove Button and confirms the action.

Post Condition(s):

* + The user is on the Favourites Screen of the Listing

## 1.6 Sequence Flow Diagrams:

C:\Users\Jeremy Liau\Desktop\NewDiags\KasperUIFlowDiag1.png

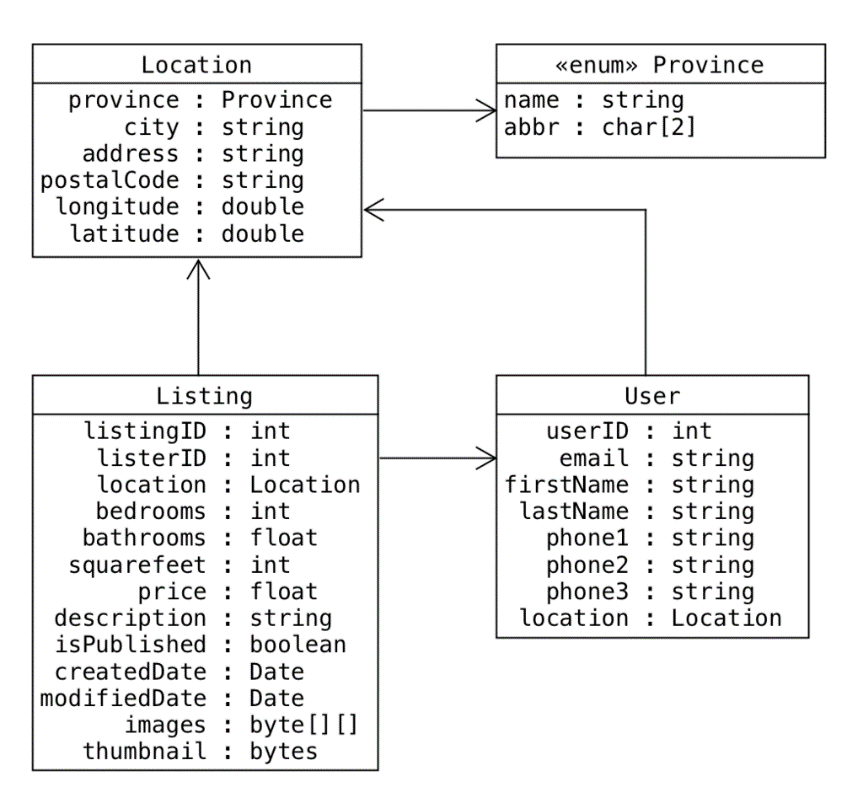
The above diagram depicts the Main, Filter, and Favorites screens, as well as their current states of implementation. Implemented and tested routes are highlighted in green. Non-implemented and inaccessible routes are highlighted in red.

C:\Users\Jeremy Liau\Desktop\NewDiags\KasperUIFlowDiag2.png

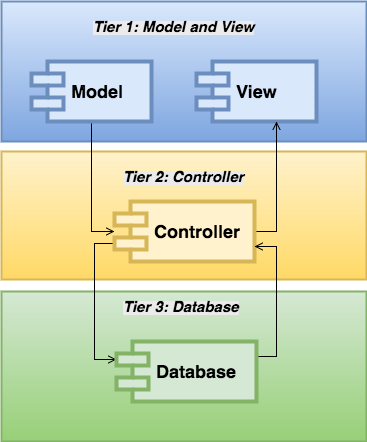
The above diagram depicts the Sign-In, Sign-Up, My Listings, and Settings screens, and their current states of implementation. Implemented and tested routes are highlighted in green. Non-implemented and inaccessible routes are highlighted in red.

# 2.0 Design

Data Structures UML Diagram ID-1



Architecture

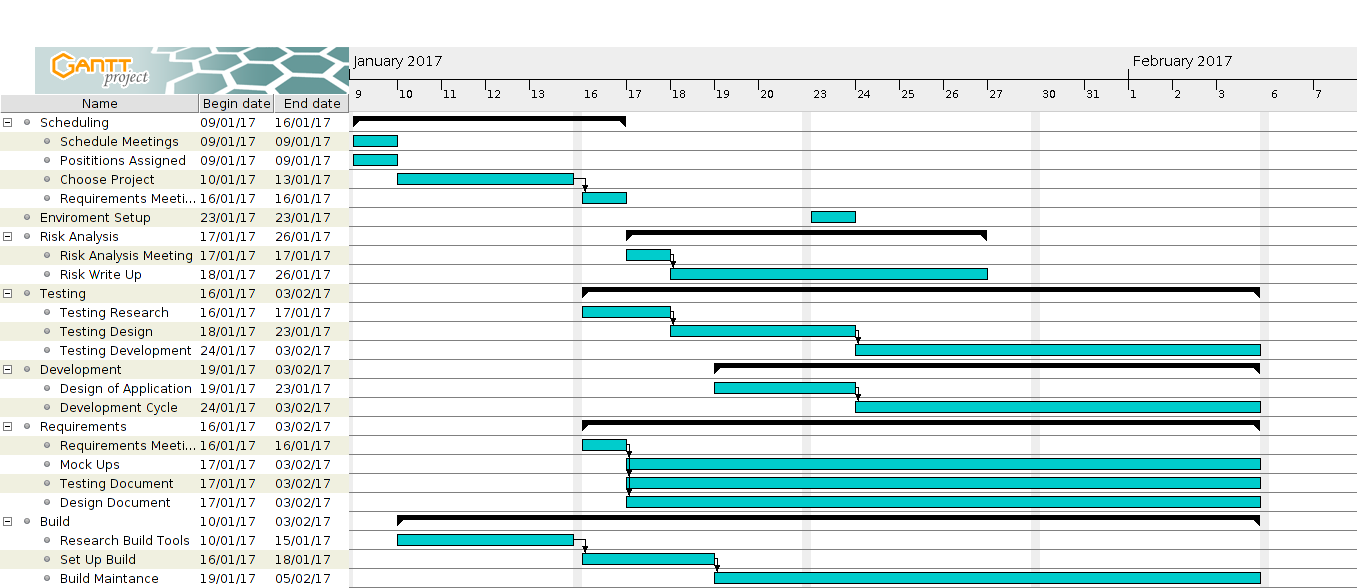
The Ionic framework uses a model view controller based architecture which allows for separation of concerns. Because the client requires the use of this framework, its native structure must be accounted for upon deciding the overall architecture of the system. The view, consisting of HTML and SCSS, provides the structure and style for the user interface. The model handles components and interactions, and is limited to a small number of typescript files. Most of the work is done within the controller component, which also handles database interactions. Systems which are centered around a database generally adhere to an n-tiered architecture, where each layer communicates only with its adjacent layers. Because Ionic is built on MVC architecture, and because the Controller is the only component interfacing with the database, the View and Model components can occupy their own layer, away from database interactions. For the reasons stated, our system will be three-tiered – tier 1 includes the Model and View components; tier 2, the Controller component; and tier 3, the database.

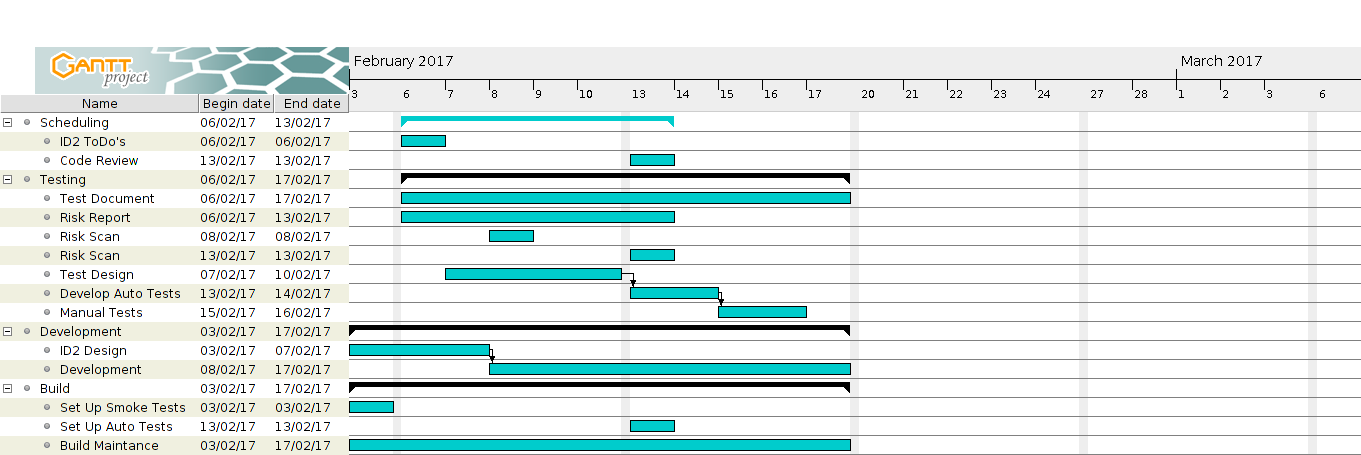
This architecture is well suited to our system because it allows for the clean

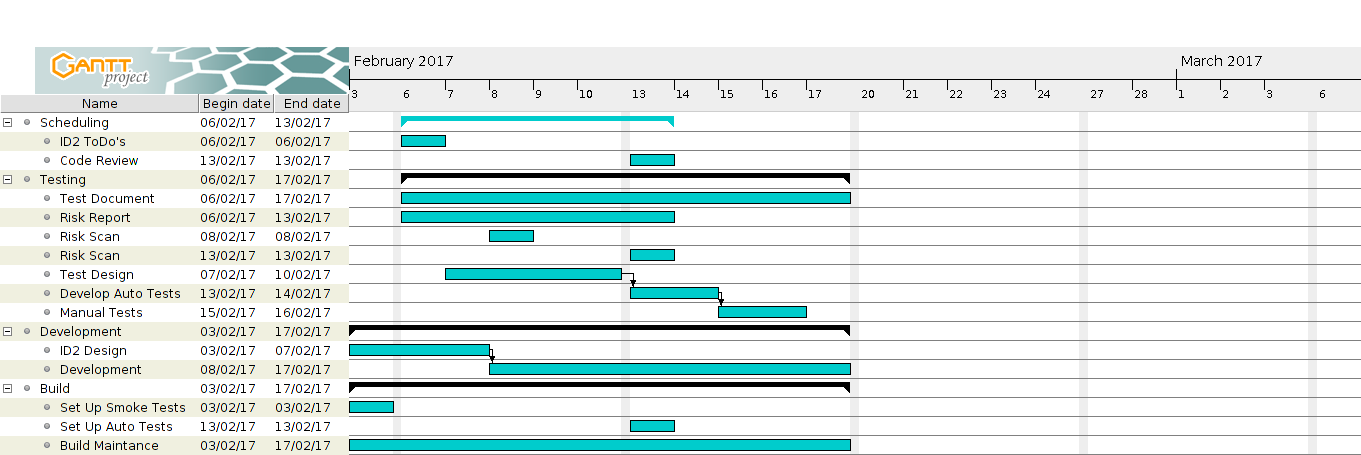
separation of our components.

# 3.0 Testing Plan

For this ID, most of the testing will be done on the GUI. This will be done with manual tests along with an automated end-to-end test using Protractor, all of which will interact with the UI. The automated test with Protractor is meant to re-enact a user’s experience from start to finish. The test will enter both valid and invalid data into the fields, just as a user would. It will also test all of the buttons that have been implemented by the Development team. The results of our manual tests will be recorded in a test matrix. During the manual tests we will record: what is being tested, the data that will be entered, the expected result, and the actual results. Documenting these tests will ensure that the defaults can be tracked. Because ID1 is the earliest stage of implementation, there is not a lot of functionality to be tested. Future testing will largely be focused on acceptable and unacceptable filter input, and continued implementation of automated end-to-end testing.

Gantt Diagram ID-1:



Gantt Diagram ID-2:

Test Matrix:

<https://github.com/CMPT371Team1/Documentation/blob/master/ID1-Documentation/TestingDocuments/testMatrix.xlsx>

# 4.0 Test Report

Manual testing has been done in the chrome browser. As the system is incomplete there are many unimplemented features. Tests done on input fields have been limited to comparison tests on password confirmation/changing, valid/invalid credentials upon log-in, and allowable data for input fields (characters/symbols in number-only fields). Some of these input fields are working as intended, but others are not (see test matrix at the link above).

Unimplemented features: Signing out does not redirect to the Sign-in screen, filters are not saved, cannot upload images for listings, cannot favourite/dislike, and settings cannot be saved.

The end-to-end test is incomplete. Currently, it allows traversal between screens, but not writing to fields. Research is being conducted on locating field elements with Protractor without the use of ng-models in HTML.

# 5.0 Coding Style Guide

Initially, our team decided to follow the Google JavaScript Guidelines:

<https://google.github.io/styleguide/jsguide.html>

However, because this resource is detailed and extensive, and because we will also be using TypeScript, we decided to create our own set of guidelines to serve our purposes. The guidelines highlight the salient features of the coding style to be followed by all developers, along with useful examples for quick referencing.

The summarized guide and sample can be found on the following wiki page:

<https://github.com/CMPT371Team1/Documentation/blob/master/ID1-Documentation/OtherDocuments/CodingStyleGuide>

# 6.0 Build Report

Smoke Test Status:

The smoke test is currently not in place due the relatively small amount of implemented code in ID-1. The build manager is currently working to incorporate Protractor, an automated UI testing tool. Research is being conducted on protractor installation and the Travis-CI script.

Build Status:

The builds for iOS and android are running and simulating correctly. The Linux build is currently building, then stalling and waiting for input. We intend to have the Linux build simulating before the end of ID-1. Server builds are currently not in effect, because the team will begin back-end requirements in ID-2. All builds are currently slow, with build time reduction a priority for ID-2.

SDK’s, Packages, and Tools:

All SDKs, packages, and tools employed in our build, as well as their version number, are subject to change. These frameworks are still in question due to lack of experience. These decisions will be made final once the build manger has a firm understanding of all automated testing, deploying, server builds, and system builds.

Current list of SDK’s, Packages, and Tools:

* Cordova CLI: 6.5.0
* Ionic Framework Version: 2.0.0-rc.5
* Ionic CLI Version: 2.2.1
* Ionic App Lib Version: 2.2.0
* Ionic App Scripts Version: 1.0.0
* npm: 3.10.10
* jdk: 1.8.0\_121
* nvm: 0.32.0
* node: 6.9.4
* plus ~400 other Ionic dependency packages
* Android:
  + SDK Platform Android 7.1.1, API 25, revision 3
  + Android SDK Tools, revision 25.2.5
  + Android SDK Build-tools, revision 25.0.1
  + Android SDK Platform-tools, revision 25.0.3
  + Google Repository, revision 42
  + Android Support Repository, revision 42
* iOS:
  + OS: OS X El Capitan
  + Xcode version: Xcode 7.3.1 Build version 7D1014

Corodova 6.5.0 requires both jdk 1.8 (or higher), and npm v2.2.1 and node v4.0.0. We will be using the most recent version of node and npm to reduce version conflicts. All developers and testers have been set up with the latest versions of all tools. We are using Xcode 7.3.1 for the time being. The Ionic dependencies are extensive, and can be viewed in further detail at the following link:

[https://ionicframework.com/docs/](https://ionicframework.com/docs/%20)

# 7.0 Defect Report

**Testing:** Invalid phone-number registration

**Data:** Phone#: “notANumber”, or any characters

**Expected result:** Disallow characters to be entered

**Actual result:** Characters are accepted

**Priority:** Low

**Testing:** Symbols entered in filter price range

**Data:** Symbols entered (+-+-)

**Expected result:** Disallow user to enter symbols

**Actual result:** Symbols are accepted

**Priority:** Low

# 8.0 Upcoming Requirements

## 8.1 ID2 User Requirements:

* Generate Dummy Data Structures:
  + UserVotes:
    - Integer 🡪 UserID
    - Integer 🡪 ListingID
    - Boolean 🡪 isLiked
  + Filter:
    - Location 🡪 Location
    - Float 🡪 minPrice
    - Float 🡪 maxPrice
    - Integer 🡪 minBedroom
    - Integer 🡪 maxBedroom
    - Integer 🡪 minFeet
    - Integer 🡪 maxFeet
    - Boolean 🡪 isFavourites
  + BrowseList:
    - Integer 🡪Cursor
    - Integer[100] 🡪 ListingIDs
    - Listings[5] 🡪 curLoaded
  + ListingImages:
    - Integer 🡪 ListingID
    - Image 🡪 Image
    - Integer 🡪 OrderDisplayed
* Generate Mock Classes:
  + ListingProvider:
    - addListing(ListingToJSON: String)
    - editListing(ListingToJSON: String)
    - removeListing(ListingID: Integer)
    - dislike(ListingID: Integer)
    - addToFavourites(ListingID: Integer)
    - removeFromFavourites(ListingID: Integer)
    - search(Filter: Filter) 🡪 Filter can be Null
  + LoginProvider (to be determined)

General User Requirements:

* Log on
* Sign up
* Edit settings and User Info
* View all property Listings
* Listing swipe feature (cursor)

Buyer Specific Requirements:

* Filter Search Based on:
  + City/Town
  + Address
  + Category (house, condo, building, etc.)
  + Square Feet
  + Price range
  + Number of bedrooms
  + Number of bathrooms
* View all Listing Info (including pictures, descriptions, and seller contact info)
* Save Listing to Favourites
* Browse Favourites
* Remove Listing from Favourites
* Receive notifications regarding Favourites
  + Price changes
  + Listing removed/edited

Seller Specific Requirements:

* View personal Listings
* Edit personal Listings (text fields, description, and images)
* Add new Listings (contact info can be automatically added via sign-up info)
  + Upload images
  + Input:
    - City/Town
    - Address
    - Category (house, condo, building, etc.)
    - Square Feet
    - Price
    - Number of bedrooms
    - Number of bathrooms
* Remove Listings

## 8.2 Back-end System Requirements

**System Design:**

The back-end system implementation is separated into two main modules – User accounts and Listings information. The account module handles user Sign-in, Sign-out, Sign-up, email verification, forgotten passwords, and resetting passwords. The Listings module defines a set of data related to a listed property, such as its location, price, description, images etc. To start the back-end, the http server is initialized, which calls all system modules before serving user requests. Thus, it has complete control over all parts of the system, and can decide to close any aspect should an issue arise.

**System Requirements:**

Fundamental aspects of the back-end behaviour can be defined by the following set of requirements:

**Functional Requirements:**

1. The back-end must gather data sent from devices and store it in the database for future reference.
2. User requests must be handled appropriately, and relevant information stored in the database must be sent to the device interface for display.
3. The system should be capable of recovering from failures and crashes whilst maintaining the integrity of any stored data.

**Non-functional Requirements:**

1. The back-end system should be responsive to user requests, so that delays in displaying data are minimized.
2. Data integrity and error correction mechanisms should be implemented so that no erroneous data is stored in the database.
3. The system should send informative error messages to the client about the source of error.
4. The system should provide an appropriate debugging environment, in which new code can be easily integrated, tested, and checked for errors.

**Software:**

The back-end system is implemented in Python and uses several external sources for specific implementations:

1. Google App Engine
2. NoSQL
3. Google Datastore NDB Client Library
4. Webapp2: a lightweight Python web framework

## 8.3 Future Requirements (Nice to Haves)

* Book a viewing feature
* Sign in using Facebook feature
* Set price watch on a given Listing
* Users can sign up to receive “hot list” notifications
  + Feature Listings (paid for by Sellers)
  + Newly added Listings
  + Price changes on Favourites
  + Based on previous search history
* Push notifications if something changes regarding a Listing saved in Favourites (change in database triggers notification)
* Sellers receive notifications regarding personal Listings:
  + When a Listing is saved to Favourites
  + When someone requests a viewing
  + When someone sets a price watch
* Integration with Google Maps
* Super admin User:
  + Log in as Super Admin
  + Add new Listing under any user
  + Edit any Listing
  + Remove any Listing

# 9.0 Triage Meeting:

<https://github.com/CMPT371Team1/Documentation/blob/master/Meetings/TriageTeamMeetings.txt>