KASPER - ID ONE

Software Development Team:

Project Manager: Tushita Patel

Developing Lead: Dylan Prefontaine

Testing 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 2](#_Toc473832447)

[1.1 Requirements ID-1: 2](#_Toc473832448)

[1.2 Platform Description (iOS, Android, and Web Browser) 3](#_Toc473832449)

[1.3 GUI Mock Ups: 4](#_Toc473832450)

[1.4 Use Cases: 6](#_Toc473832451)

[1.5 Sequence Flow Diagrams: 10](#_Toc473832452)

[2.0 Design 12](#_Toc473832453)

[3.0 Smoke Test Status 12](#_Toc473832454)

[4.0 Mini Milestones for ID1 12](#_Toc473832455)

[5.0 Risk Assessment 13](#_Toc473832456)

[Intro 13](#_Toc473832457)

[5.1 Technical Risks: 13](#_Toc473832458)

[5.2 Non-Technical Risks: 16](#_Toc473832459)

[5.3 Future Risks: 19](#_Toc473832460)

[6.0 Inspections Report 19](#_Toc473832461)

[7.0 Client Meetings 19](#_Toc473832462)

[8.0 Testing Results and Defect Reports 19](#_Toc473832463)

[9.0 Upcoming ID-2 Requirements 20](#_Toc473832464)

[User Requirements: 20](#_Toc473832465)

[System Requirements 21](#_Toc473832466)

[10.0 Future Requirements (Nice to Haves) 21](#_Toc473832467)

1.0 Requirements Document

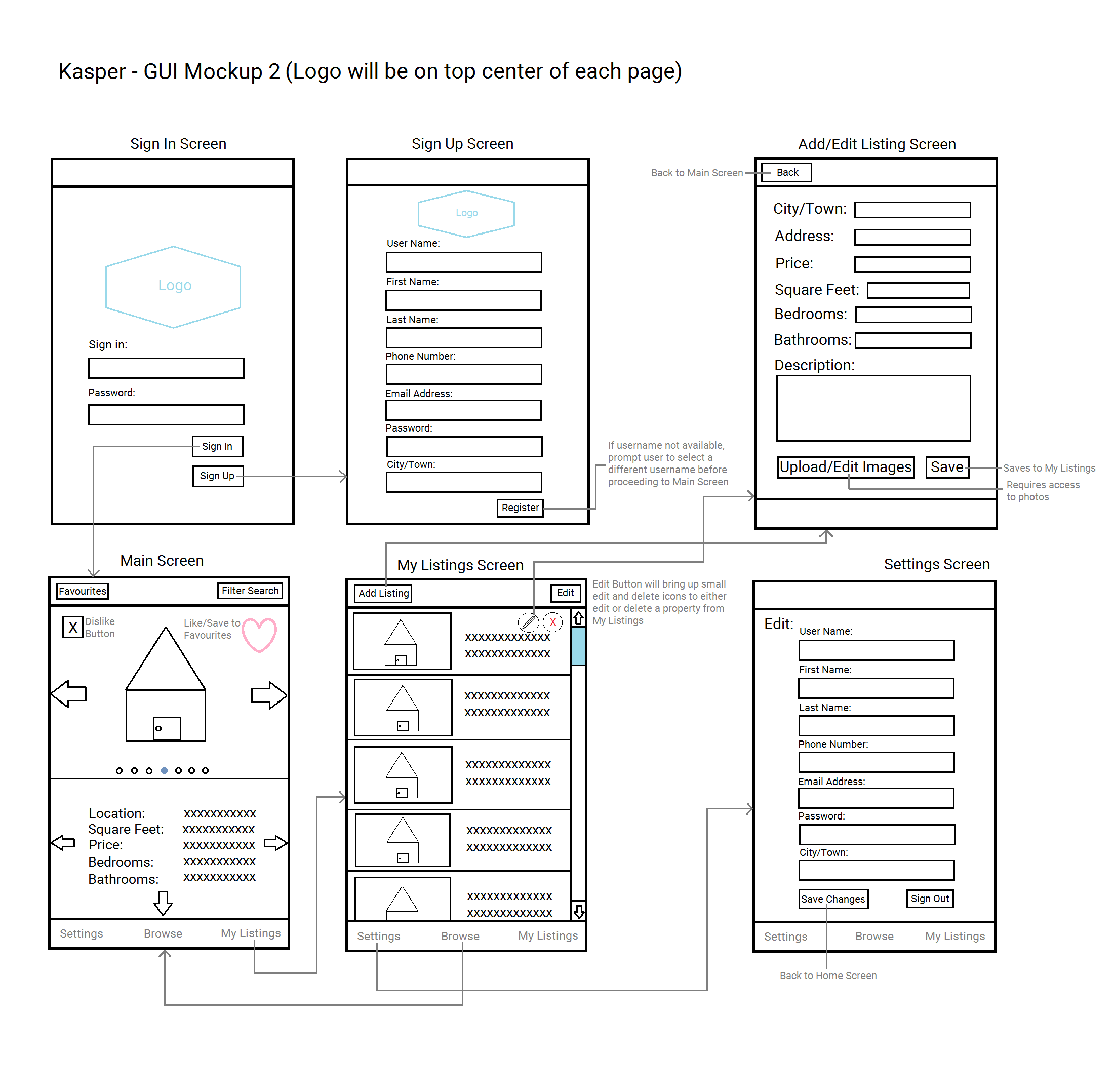
## 1.1 Requirements ID-1:

* 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

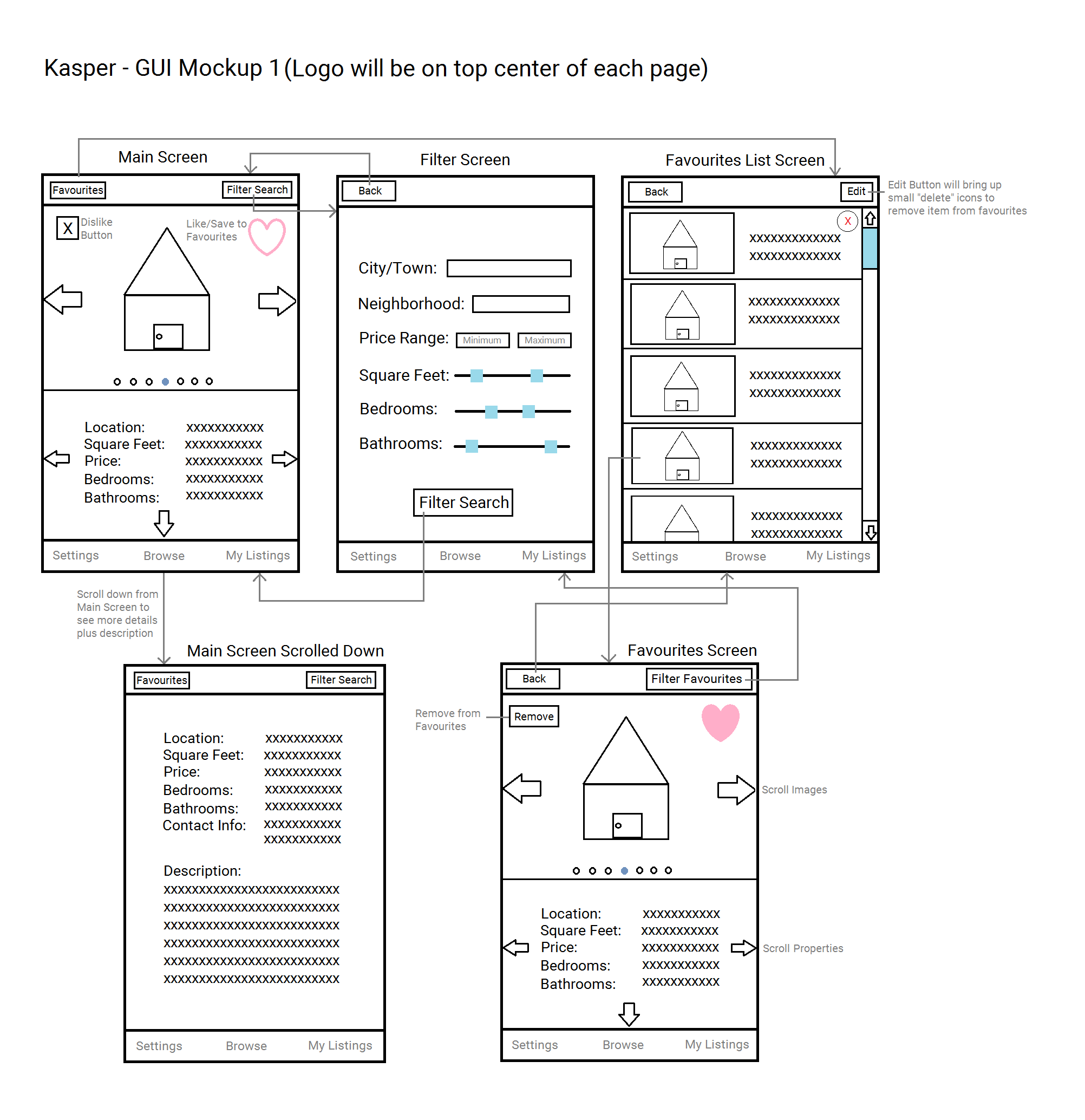
## 1.2 Platform Description (iOS, Android, and Web Browser)

* Insert Platform Description by Tushita

## 1.3 GUI Mock Ups:

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

The above diagram shows the flow of events when a user chooses to either Sign in or Sign up. When a user signs in or signs up successfully, they will be taken to the Main Screen, where they can browse through an unfiltered set of property Listings. Users who wish to post their own Listings may do so by choosing the My Listings option from the bottom tab bar. Clicking on My Listings will take the user to the My Listings screen, where they can Edit, Remove, and Add New Listings. Choosing the Add Listing button from the top tab bar will navigate to the Add/Edit Listing screen, where the user can set up their new Listing, add pictures, and provide a description. Lastly, users may edit their personal profile in the Settings screen.

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

The two large sideways arrows on the Main Screen indicate the ability to scroll through the provided images, while the two smaller sideways arrows below indicate the ability to swipe on to a new Listing. The small arrow on the bottom shows the ability to swipe down for more information as shown. Users will have the option to filter a search and be shown only those Listings which match their filter criteria. Users can tap the heart button to save a Listing to Favourites, and all Listings saved to Favourites can be removed by choosing the Edit button on the Favourites List Screen. To view the Listing of an item saved to Favourites, the user must click on the desired Listing in the Favourites List screen, which will bring up their selected Listing in the Favourites Screen. Users may also remove a Listing from Favourites by hitting remove on the Favourites screen.

## 1.4 Use Cases:

Sign-in:

* 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:

* 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

Filter:

* Actor(s):
  + User
* 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 entries are updated with search results
* Alternate Flow:

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

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

Swipe through Listings:

* 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

Add Listing to Favourites:

* Actor(s):
  + User, Database
* Precondition(s):
  + User is on the Main Screen
* Basic Flow:
  1. User selects Favourite Button on a particular Listing
  2. The Listing is added to user Favourites
* Alternate Flow: none
* Post Condition(s):
  + Listing is added to user Favourites

Dislike a Listing:

* Actor(s):
  + User, Database
* Precondition(s):
  + User is on the Main Screen
* Basic Flow:
  1. User selects the Dislike Button on a particular Listing
  2. The property is blocked from being re-added to the Listings buffer
* Alternate Flow:

3\*. The user hits the Dislike Button again and the Listing is unblocked from re-entering the buffer

* Post Condition(s):
  + The Listing is blocked from re-entering the Listings buffer for that user

Settings:

* 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

Add Listing:

* 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:

* 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: none
* Post Condition(s):
  + Listing modified in My Listings and the Database

View Favourites List:

* Actor(s):
  + User
* Precondition(s):
  + User is on the Main Screen
* Basic Flow:
  + User selects Favourites Button
* Alternate Flow: none
* Post Condition(s):
  + The user is on the Favourites List Screen

View a Favourite Listing:

* Actor(s):
  + User
* Precondition(s):
  + User is on the Main Screen
* Basic Flow:
  + User selects a Listing from the Favourites List
* Alternate Flow: none
* Post Condition(s):
  + The user is on the Favourites Screen of the particular Listing

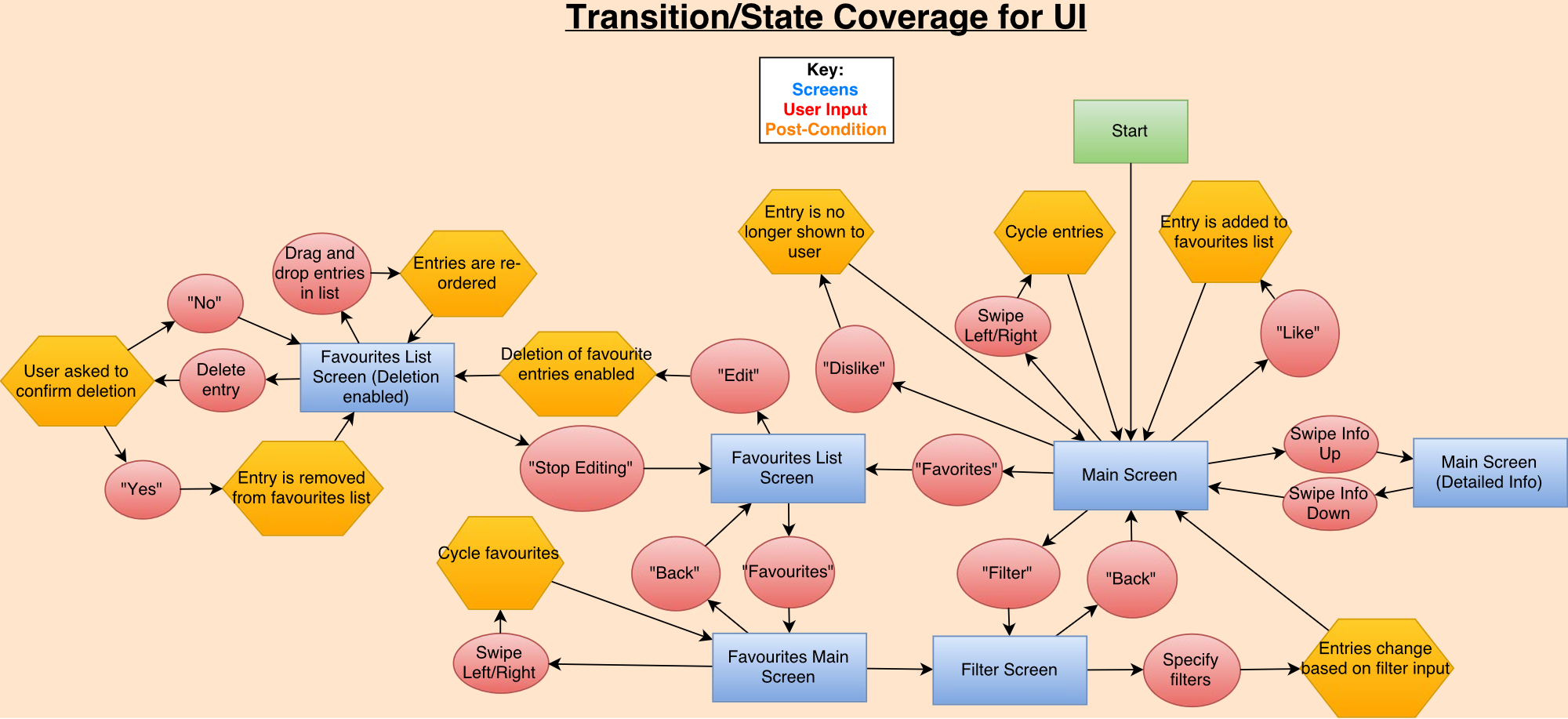
Sign-up:

* Actor(s):
* Precondition(s):
* Basic Flow:
* Alternate Flow:
* Post Condition(s):

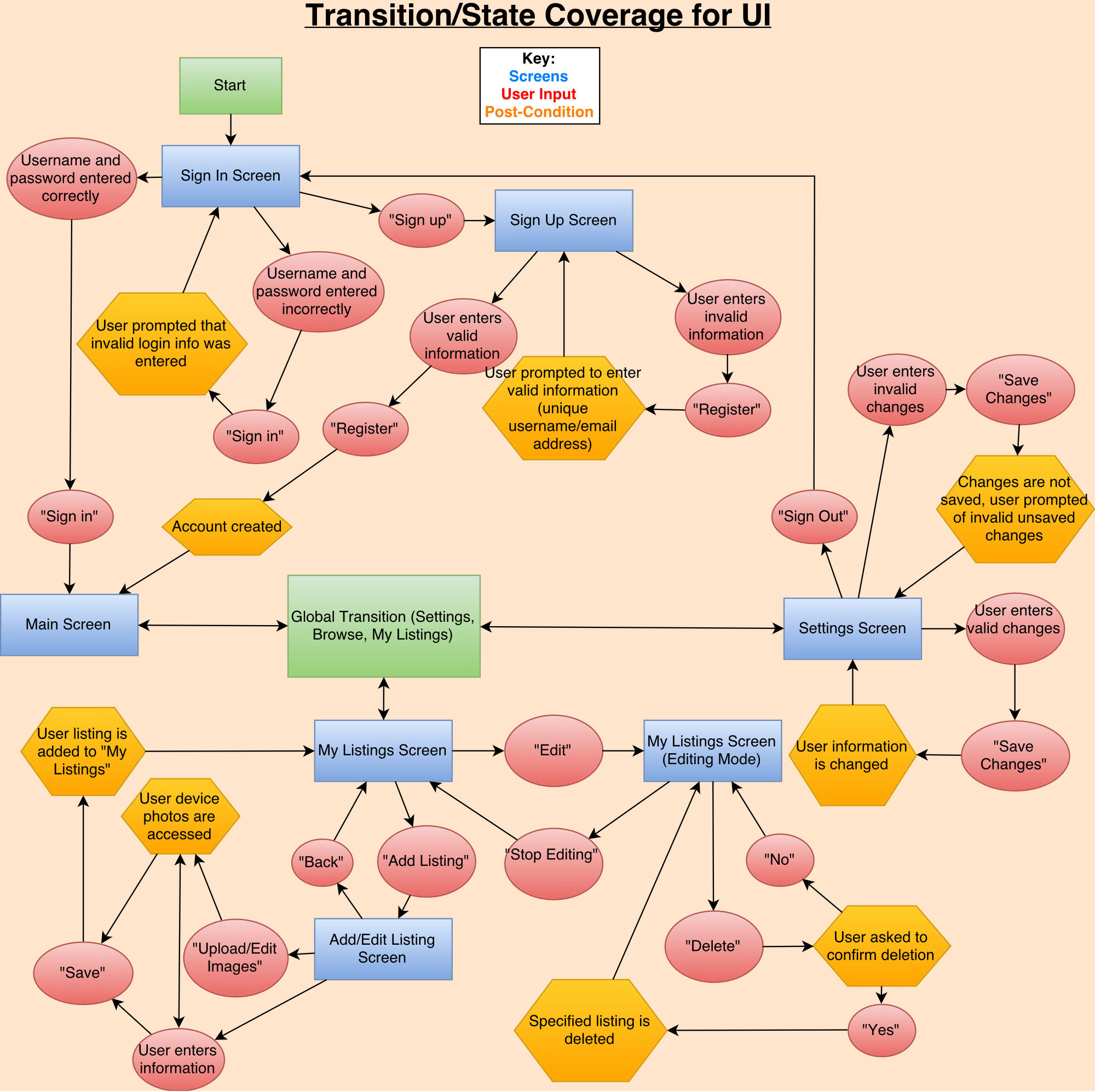
Remove Listing from Favourites:

* Actor(s):
  + User, Database
* Precondition(s):
  + User is on Favourites List Screen
* Basic Flow:
  + User selects the Edit Button
  + User selects the Remove Button
  + User confirms removal of Listing
* Alternate Flow:
* Post Condition(s):
  + Listing is removed from Favourites List

## 1.5 Sequence Flow Diagrams:

****

* Jeremy will replace these and test team will provide brief description

****

# 2.0 Design

* UML Diagrams -> DYLAN
* Architecture Diagram -> GAURAV
* Architecture Description / Defense -> KRISTOF

SELECT \* FROM (table) WHERE (current range\*) ORDER BY (chosen field), id LIMIT 100

\*this range will be stored on the device to resume the search later.

* when you get to 100, call the same sort (sorted by filter) again on the database and skip to the stored cursor position

Architecture is 3-Tiered such that tier 1 is View/Model components, tier 2 is Controller, tier 3 is the server. MVC is used by ionic.

# 3.0 Smoke Test Status

* Chris MR

# 4.0 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 ✔
* Finalize Design for the System
* Implement GUI Designs

**Test Team:**

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

**Build:**

* Set up Travis with Ionic
* Verify code style (low priority)
* Set up Django, goDaddy and server

**Documents:**

* Testing Plan Document
* Risk Report
* Requirements Document
* Design Document
* All documents edited and compiled into one fluid document

**Other:**

* Each Developer and Tester has completed a pair programming session
* Get server to host Django
* Learn how to host Website on goDaddy.com
* 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 ✔

# 5.0 Risk Assessment

## Intro

The purpose of this Risk Assessment is to identify and plan for issues that may come up from the beginning of the project until the due date for ID-1. The risks are divided into three categories – technical, non-technical, and future. Technical risks are risks arising from the construction and design of our system. Non-technical risks focus on risks related to group coordination, management, client relations, and various other components associated with managing our team. For each of the risks identified, a description of the risk will be given including the severity of the risk, the likeliness of the risk arising, potential scenarios that would cause the risk to materialize, a plan to mitigate the risk, and a contingency plan to deal with the issue if it occurs. The risk and loss sections are on a scale from zero to one, with one being most severe. These values are informal approximations and should not be considered measured probabilities.

## 5.1 Technical Risks:

Problems with Integration

**Probability:** 0.8

**Loss:** 0.6

**Scenario:** For this project, most of the team will be using new and unfamiliar technologies. This may cause issues with different pieces functioning together. This might include problems with build tools, testing frameworks, development tools, or one of the many other tools being used by the team.

**Mitigation:** Team members will do extensive research in deciding which tools to use. This will (hopefully) reduce the number of integration issues that arise.

**Contingency Plan:** If an integration issue arises, further development will be halted until the issue is fixed. The exact manner of resolving the issue will depend on the problem. If the problem is not fixable, the integration of different tools may come under consideration.

Dev Team Does Not Provide Test Team with Code by the Agreed Upon Deadline

**Probability**: 0.4 Medium

**Loss**: 0.2

**Scenario**: Dev team attempts to finish all requirements without regard for the agreed upon deadline and/or

underestimates the time needed to finish something. This could result in the deliverable being improperly tested and underlying bugs may persist.

**Mitigation**: Set firm deadlines for Dev team to complete implementation.

**Contingency plan:** If the Dev team does not adhere to the deadlines, the outcome is that the code will not be thoroughly tested.

Code is Not Designed for Testing

**Probability**: 0.6

**Loss**: 0.3

**Scenario**: A segment of code has too large a scope to test individual functionality and therefore cannot be thoroughly tested.

**Mitigation:** Dev team must maintain awareness that their code will be tested and must make use of assertions; this will allow the Test team to catch defects earlier.

**Contingency Plan:** If the issue is caught early enough, the Dev team may be able to restructure the code to make testing easier. Otherwise the test team will have extra work.

Code is Not Well Tested by the Deliverable

**Probability**: 0.6

**Loss**: 0.4

**Scenario**: The Test team becomes busy and does not spend enough time on testing. Unknown bugs will exist in the project.

**Mitigation**: Set up schedules for each deliverable and communicate effectively to ensure enough time and effort is spent on testing.

**Contingency plan:** Hope that no major bugs appear during presentation. Inform client that more testing will be done in future. If this issue arises, it may have to be accepted.

Team Members do not Document Work Well Enough

**Probability**: 0.7

**Loss**: 0.5

**Scenario**: A team member is more concerned about finishing their work than documenting what they did. Could also arise from the assumption that everyone understands what needs done, but there is a lack of communication regarding who is doing what.

**Mitigation**: Schedule and individual work logs are set up on GitHub. Team members will encourage each other to keep it updated. Grades are associated with work logs, which will hopefully encourage group members to keep track of their work.

**Contingency Plan:** If there is a lack of documentation, past work sessions will be documented as accurately as can be remembered.

We Do Not Use the Best Tools for the Job

**Probability**: 0.8

**Loss**: 0.2

**Scenario**: Assumptions that certain tools will work together might be false, or Travis could undergo an update and no longer support a required package.

**Mitigation**: Tool options will be thoroughly researched and risk scans will be performed to ensure the most accurate assessment possible.

**Contingency plan**: If the wrong tool is chosen, work that is not useable will be thrown away to make room for the correct tool.

Misunderstanding Requirements

**Probability**: 0.7

**Loss:** 0.6

**Scenario**: This risk is expected to occur during client meetings and would result from miscommunication. This could be due to the use of “jargon” that the client might misinterpret. The client might also not understand each component involved in producing the final product. Another similar risk is having conflicting priorities with the client. In order to display our progress, new work and increasing functionality will be provided for each ID deadline. The team must meet with the client often enough to set realistic goals for each ID.

**Mitigation:** Many of these issues can be limited if there is frequent and qualitative communication between our project manager and the client. It is also important to note that communication with the client should be limited for the other team members. This will keep the project better organized and improve efficiency by allowing team members to focus on their individual tasks. User functionality desired by the client will require evaluation of the underlying components involved to achieve their goals.

**Contingency plan:** If there is a misunderstanding of requirements during the project, we may have to call a team meeting to discuss any misconceptions and ensure that everyone is in agreement on the requirements. We will also reduce the requirements planned for the present ID in order to fix the issues that exist and get the project back on track.

Travis Servers are Busy

**Probability**: 0.8

**Loss**: 0.4

**Scenario**: Travis servers are busy and our build is more time consuming than anticipated. This will cause delay in debugging.

**Mitigation**: Build manager will set up a local Travis container so that testing can occur without the Travis server.

**Contingency Plan**: If Travis is busy right before and ID deadline, contact Travis support. Otherwise, be sure to finish deployment well in advance of the deadline.

## 5.2 Non-Technical Risks:

Busy Schedules

**Probability:** 0.9

**Loss:** 0.3

**Scenario:** During the course of the term, group members may have other commitments that will reduce the amount of time they can dedicate to the project. The majority of our team is participating in the CSPIP program and are expected to have interviews over the last two weeks of January (ID-1). It is also anticipated that many team members will be less available during midterm season (ID-3), as well as towards the end of term (ID-5). This risk is expected to occur with all team members at some point throughout the term.

**Mitigation:** In order to reduce the effects of unavailable team members, knowledge of the entire project must be shared amongst the group. This will allow the project to move forward regardless of varying availabilities. Clear communication is required for scheduling meetings and deadlines that work for everyone. Communication will be done at stand-up meetings and via Slack. Additionally, daytime meetings and peer reviews will not be scheduled from January 23rd – 30th, when group members have scheduled interviews. All meetings, peer reviews, and stand-ups are scheduled for a minimum amount of time, leaving the majority of communication to occur via Slack.

**Contingency Plan:** When certain group members are unavailable, others must step in to make up for their absence. Vice leads for each team will be assigned and will take over when the leads are absent.

Team Member Unexpected Absence

**Probability:** 0.8

**Loss:** 0.3

**Scenario:** An unexpected absence for a team member may include sickness or other emergencies that cannot be planned for in advance.

**Mitigation:** Knowledge of the entire project must be shared amongst team members. This will allow the team to continue production even if someone is unavailable. To gain a thorough team understanding, each member will perform pair programming sessions at least once per ID. Regular code reviews with as many members as possible will also be a strong preventative measure.

**Contingency Plan:** Communication between team members when someone is unavailable and fair distribution of the missing member’s assigned work.

Team Member Informed Absence

**Probability:** 1

**Loss:** 0.2

**Scenario:** A team member might be absent and unable to contribute for a known amount of time. This might also occur over shorter periods for events such as job interviews.

**Mitigation:** Knowledge of the entire project must be shared amongst team members. This will allow the team to continue production even if someone is unavailable. Group members are also frequently advised to write well-commented code to aid in readability.

**Contingency Plan:** Time will be set aside in group meetings to communicate informed absences and to redistribute the work of the absent team member.

Team Member Drops Class

**Probability:** 0.2

**Loss:** 0.7

**Mitigation:** Keeping team morale high could help prevent students from dropping the class. This involves good communication, patience, positivity, and inclusion. All group members have been asked to inform the team as early as possible if they intend to drop the class. This could provide the team with extra time to adjust.

**Contingency Plan:** If somebody drops the class, that person’s work must be fairly distributed amongst the team.

Group Dynamic Problems

**Probability:** 0.8

**Loss:** 0.4

**Scenario:** If somebody feels that another group member is not contributing in a meaningful way, they may choose to deal with the issue in an inappropriate way. This may also arise from a disagreement over design details, what tools to use, functionality, etc. Our team expects to encounter disagreements throughout the project.

**Mitigation:** Keeping team morale high will help to ensure that team members approach conflict from a constructive place. Team members will be reminded that there are effective ways to confront an issue, and that all group members have valuable ideas, and therefore all opinions should be heard.

**Contingency Plan:** If a group member is upset, time should be taken to calm down before dealing with the issue. The group member should then try to handle the issue individually with the person they are in conflict with. Should this prove ineffective, they can take their issue to the project manager to help mediate the situation.

Inaccurate Time Estimations

**Probability:** 0.8

**Loss:** 0.4

**Scenario:** Time estimations may be inaccurate due to lack of experience.

**Mitigation:** Thorough research can be performed toimprove our knowledge and the accuracy of our estimations.

**Contingency Plan:** Where reasonable, reduction of requirements for the current ID can create extra time to implement the top priority features. This does not necessarily apply to situations where team members fail to allocate sufficient time for a prioritized task.

Client Is Not Satisfied with Product

**Probability**: 0.2 low (client is flexible and open minded)

**Loss**: 0.2

**Scenarios**: The work required to achieve the final product is underestimated and the team is unable to deliver what was promised.

**Mitigation**: Frequent communicate with the client will be required in order to keep everyone on the same page. Risks will be monitored in order to improve accuracy of time predictions.

**Contingency plan:** Let client know as soon as we know we cannot deliver what we promise. He may have to just accept it how it is.

Client Becomes Unavailable

**Probability**: 0.8

**Loss**: 0.2

**Scenario**: The CTO goes on vacation and does not reply to our emails.

**Mitigation**: Open and frequent communication with the client will help the team gain awareness of when the client might become unavailable.

**Contingency Plan**: The team will move forward with the project based on the agreed upon highest priority.

Build Master Becomes Busy

**Probability:** 0.8

**Loss:** 0.3

**Scenario:** Chris becomes busy with other classes and is temporarily unable to contribute.

**Mitigation**: Designate a member to fill in for Chris if he becomes absent. The designated person should shadow/pair program with Chris so that both of them understand how the build works.

**Contingency plan:** Working on designating a temporary Build Master.

## 5.3 Future Risks:

No Access to Database by ID-2

**Probability:** 0.2

**Loss:** 0.4

**Scenario:** In order to develop certain aspects of the app, the team will require access to the client’s database. This is currently unavailable.

**Mitigation:** Communication with the client must be clear so that we can plan our ID requirements based on what is available.

**Contingency Plan:** If access to the database is not provided in time, we will be forced to set up our own database in order to properly develop and test our app. Departmental help may be required for set-up.

Server issues

* To be determined in later deliverables

# 6.0 Inspections Report

* Tushita

# 7.0 Client Meetings

* Tushita

# 8.0 Testing Results and Defect Reports

* Jeremy and Test Team

# 9.0 Upcoming ID-2 Requirements

## 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

## System Requirements

* SSL certificate
* Server for data storage
* Domain access

# 10.0 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
* Future System Requirements
  + Tian