KASPER – ID3 PROJECT DOCUMENTATION

SOFTWARE DEVELOPMENT TEAM:

Project Manager: Tushita Patel

Dev Lead: Kristof Mercier/Dylan Prefontaine

Test Lead: Jeremy Liau

Build Manager: Christopher Mykota-Reid

Developers: Gaurav Arora, Haotian Ma, Melody (Tian) Zhao

Test Team: Christopher May, Ryan Tetland

Documentation: Arianne Butler

Contents

[1.0 Requirements Document ID3 3](#_Toc476317591)

[1.1 Front-end Requirements 3](#_Toc476317592)

[1.2 Back-end Requirements 3](#_Toc476317593)

[1.2 Mini Milestones for ID3 5](#_Toc476317594)

[2.0 Time Estimations 5](#_Toc476317595)

[3.0 Design 6](#_Toc476317596)

[3.1 API Document 7](#_Toc476317597)

[3.2 Updated Data Structures for ID3 8](#_Toc476317598)

[3.3 Development Unit Tests 9](#_Toc476317599)

[4.0 Testing Document 9](#_Toc476317600)

[4.1 Test Plan 9](#_Toc476317601)

[4.2 Test Report 11](#_Toc476317602)

[4.3 Defect Report 11](#_Toc476317603)

[4.4 State Transition Diagrams ID3 (to be updated) 11](#_Toc476317604)

[4.5 Critical Use Cases 12](#_Toc476317605)

[5.0 Coding Style Guide 13](#_Toc476317606)

[6.0 Build Report 13](#_Toc476317607)

[7.0 Upcoming Requirements 15](#_Toc476317608)

[7.1 ID4 Requirements 15](#_Toc476317609)

[7.2 Future Requirements 16](#_Toc476317610)

[8.0 Triage Meeting ID3 17](#_Toc476317611)

# Requirements Document ID3

## 1.1 Front-end Requirements

Priority (Not Finished in ID2)

* Receive notifications regarding Favourites
  + Price changes
  + Listing removed/edited
* Seller Upload images

Requirements Changes

This section highlights the changes to our UI since ID2. These changes were discussed and agreed upon during Dylan’s formal code review, and have been implemented during ID3. For each screen that will undergo change, a screenshot of it’s original state, along with a point form description of the changes are shown. Finalized changes to the UI for ID3 can be found at the link following this section.

**ADD KRISTOFS PART**

The above UI changes have been implemented and can be found at the following link:

Insert new link

## 1.2 Back-end Requirements

Priority (Not Finished in ID2)

* Email verification (not done)
* Forgot password (not done)
* Change password
* Get all Listings for Browse page (removed)
* Get filtered Listings
* Edit Listings
* Like/Dislike
* Edit Account
* Get Favourites Listings

Back-end 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. It includes creating new Listings, getting filtered Listings, and edit existing Listings. 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

Back-end Requirements Changes from ID2

* Remove cursor and lastListing from the getlistings API call
* Allow an optional filter data structure to be passed to the getlistings API call
  + If the filter is not given, a default will be applied. It also optionally takes and "included fields" as well as "limit" which is the number of fields that are returned. By default only returns id's. Also have a "featureImage" option in the include fields.
* some sort of listing datasrtructure or some sort of array of blobs to contain image.
* EditListing call must update modified date, return modified date in response (JSON)
* Login with token or otherwise needs to update the "lastSeen".
* Must update the server if own listing is modified on the device, must update the device if favourites has been changed on the server. Check for validity must be done on front and back end. If the backend gets an invalid listing, force update the listing on the device. This shouldn't happen in normal app usage so it's fine.
* (maybe) user specefic rate limit so that can't brute force the api.
* Have verified in user table, do a verify email thing.
* change the monstrosity of a name "FavouriteListingsUser" to "GetFavourites"
* "Sign In with token" should return everything that "SignIn" returns.
* "requestContactInformation" changed to "contactSeller". We need a new table for each user that contains incoming messages.
  + Table: senderID, lisingID, message, phone, email, received, createDate.
  + Need a push notificatin to seller that will tell them they have new messages. Then their app will need to call an api call that updates their app with the new message.
* new post request for "GetMessages" : takes in userId, returns
* editMessage: takes userId, messageID(this is just senderID apend listingID), token | either deletes the message or sets its status to read

## 1.2 Mini Milestones for ID3

**Development**:

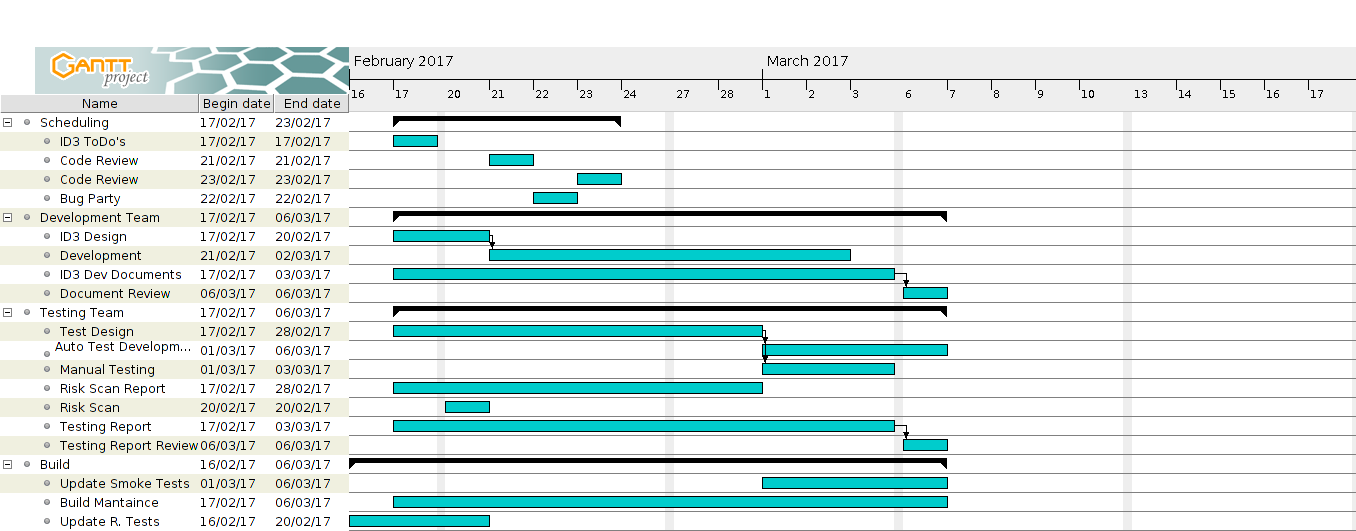
* Finish all new requirements for front and back-end on Trello boards
* Have the server up on a usask machine
* Have the front-end communicating with the server (achieved with 2 API calls, Sign in and Sign up are working)
* Make two test accounts, and dummy Listings

**Testing**:

**Documentation**:

# 2.0 Time Estimations

Gantt Diagram Time Estimation ID3:



To see time estimations on all individual tasks, please follow the link in Process Documentation, section 2.0 Activity Log, and click on the bottom tab labelled “Individual Activity Log”.

For ID3, development time estimations have been added for each assigned task on Trello. Trello cards now contain a coloured label indicating the approximate estimation of each task. These estimates were approximated by the dev team.

The following image shows the colour coded time estimation scale with which each trello card will be rated.



# 3.0 Design

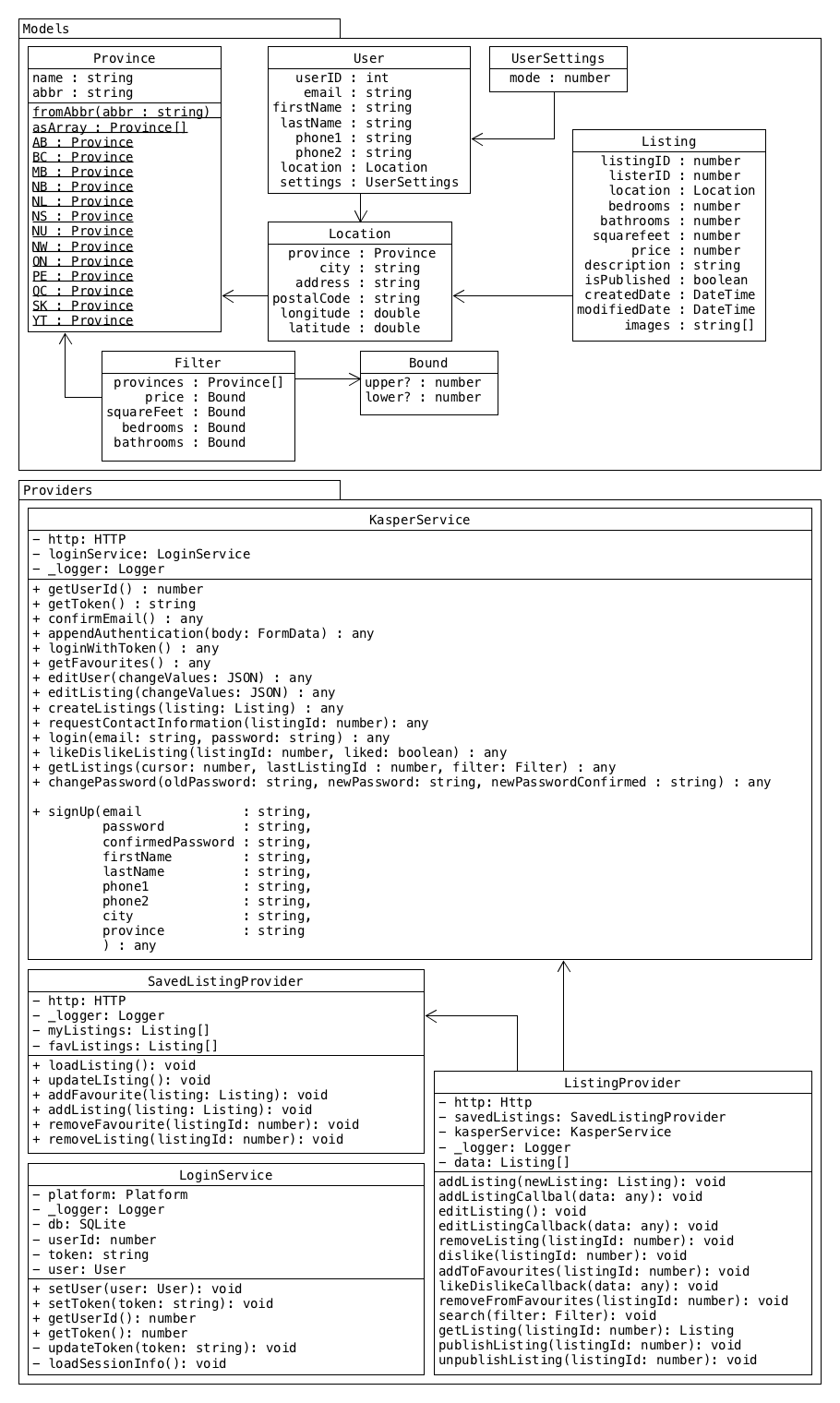
## 3.1 API Document

The API document outlines the client server communication of our system. Our project will contain roughly twelve API calls (or calls to the server), three of which are currently implemented. API calls are robust and complex calls to the server. These calls send a JSON body with relevant data. The server receives, parses, and processes the JSON using the database. When finished, it will reply to the sender with a token containing the reply status and a JSON body containing the requested information. The following link contains our API document and details on the various calls to our database.

Note: We are currently experiencing difficulty with this link. Please copy paste it into your web browser.

[https://docs.google.com/document/d/1N4jt1\_PgxPhXwdc1TcT7TBjFNOZqYO5L10ha3bpO5M8/edit#heading=h.1sskatsa28we](https://docs.google.com/document/d/1N4jt1_PgxPhXwdc1TcT7TBjFNOZqYO5L10ha3bpO5M8/edit%23heading=h.1sskatsa28we)

## 3.2 Updated Data Structures for ID3



## 3.3 Development Unit Tests

The procedure to be followed by all developers for writing both front and back-end unit tests respectively, can be found at the following links:

Front-end Guide for Unit Testing:

<https://github.com/CMPT371Team1/Project/wiki/Coding-Style-Example-(JavaScript)>

Back-end Guide for Unit Testing:

<https://github.com/CMPT371Team1/Project/wiki/Coding-Style-Guide-(Python)>

# 4.0 Testing Document

## 4.1 Test Plan

4.1.1 Test Team Responsibilities

4.1.2 Test Strategy

**Critical Items:**

**Other Items:**

**Specifics (in the form of use-cases):**

**Check for invalid inputs in the forms:**

**End-to-end testing:**

**Automatic unit testing:**

4.1.3 Training Requirements

All testers must gain familiarity with the testing tools, specifically Protractor and Istanbul. Protractor is used for end-to-end test automation, and Istanbul is used to check test coverage. Testers will write JavaScript tests for the test configuration file to use. A meeting for the test team on February 9th will go over Protractor specifically. Istanbul is a simple command line test coverage tool that outputs test coverage and requires no prior knowledge. Karma is no longer being used as our unit test runner.

4.1.4 Documentation

**Test Matrix:**

The X-axis represents what is being tested, and the Y-axis represents the requirements

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Sign-In** | **Sign-Up** | **Browse Listings Filter** | **Add New Listing** | **My Profile** | **Browse** | **My Favourites** |
| **Add a listing in ‘Seller Mode’** |  |  |  |  |  |  |  |
| **Registering a new user** |  |  |  |  |  |  |  |
| **Browse in ‘Buyer Mode’** |  |  |  |  |  |  |  |
| **Edit account information** |  |  |  |  |  |  |  |

**Defect Report:**

|  |  |
| --- | --- |
| **Bug:** |  |
| **Reproducibility:** |  |
| **Steps taken:** |  |
| **Variations:** |  |
| **Screenshot:** |  |

Testers must follow the guidelines below when producing a bug report:

* + List the bug
  + Is the bug reproducible? To what certainty?
  + List the steps to re-create the bug (input data, prior usage of application, etc.)
  + Note any variations that may occur while attempting to recreate. A screenshot can be taken.

4.1.5 Schedule

**ID1:**

Most testing will be done manually on the GUI. The test team will begin creation of automated tests with Protractor. All data should be recorded in the test matrix and all defects should have an accompanying defect report.

**ID2:**

The Test Team will integrate automated end-to-end tests with TravisCI, update the defect report, update flow charts to better reflect requirements changes/current state of the system, and re-create, simplify, and update the test matrix.

**ID3 and Future ID’s:**

Hold a bug party

Continue updating the test matrix, defect report, and flow charts

Compatibility testing for iOS and Android

Performance testing

**ID4 and Future ID’s:**

## 4.2 Test Report

The following summarizes various observations and measures taken by the Test Team in ID3:

## 4.3 Defect Report

## 4.4 State Transition Diagrams ID3 (to be updated)

C:\Users\Arianne\Desktop\CMPT371_TestTeamDocs_ID2\KasperUIFlowDiag1_ID2.png

C:\Users\Arianne\Desktop\CMPT371_TestTeamDocs_ID2\KasperUIFlowDiag2_ID2.png

## 4.5 Critical Use Cases

The following critical use cases outline the low level details of the most important features of our application.

# 5.0 Coding Style Guide

The development team has put together a set of guidelines to serve our purposes. These guidelines highlight the salient features of the coding style to be followed by developers. Useful examples are provided for quick referencing.

For front-end development using Ionic, the coding guide and sample can be found at the following wiki page:

<https://github.com/CMPT371Team1/Project/wiki/(Rough)-Coding-Style-Example-(JavaScript)>

For back-end development using Python, the coding guide and sample can be found at the following wiki page:

<https://github.com/CMPT371Team1/Project/wiki/Coding-Style-Guide-(Python)>

# 6.0 Build Report

Smoke Test Status:

Front-end tests are now implemented using protractor and are currently failing due to a change in requirements. The back-end smoke tests are implemented and failing due to more development and bug fixing needing to go on.

Build Status:

The builds for iOS and Android are running and simulating correctly. The Linux build is currently undergoing a smoke test. The server is hosted on a remote server and is being smoke tested as well. It is now integrated with the front-end. Currently we are in the process of getting it and Travis connecting then smoke tests should pass on the front-end. End-to-end smoke tests are being run using Protractor on the Firefox browser, and for the back end, being ran by a set of python scripts. Until now, all testing has been done on the browser version of our app; smoke tests are currently not in place for the other platforms.

SDKs, 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 manager has a firm understanding of automated testing, deployment, 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
* packages listed in package.json
* 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 7D1014Server:
* Server:
  + python 2.7
  + Google Cloud sdk v143.0.1
  + Python Extension for google cloud v1.9.50
  + Python Extension (Extra Libs) v1.9.49
* End to End Tests
  + Protractor v5.0.0
  + Firefox v47.0.1
  + Selenium v3.1

Corodova 6.5.0 requires both jdk 1.8 (or higher), as well as 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. Google Cloud is required for the Google App engine, the platform our servers are built upon. The server uses Python 2.7, the most recent version of Google Cloud SDK, and Python extensions for Google Cloud SDK. Our end-to-end tests are driven by Protractor, which sends to Ionic through the Selenium server. We are currently testing our system on Firefox v47.0.1, because of compatibility issues with newer versions. All developers and testers have been set up with the latest versions of the required 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)

Releases for our build can be found at the link below:

<https://github.com/CMPT371Team1/Project/tree/develop/releases>

# 7.0 Upcoming Requirements

## 7.1 ID4 Requirements

Front-end Requirements

* Form control validation for all remaining pages
* Replace descriptive text with icons
* Fix description box in Add Listings
* Add Google Maps section to Detail page
* Display currently added images
* Get a legitimate Kasper icon
* Finalize method of ordering images
* Make Favourites page the same style as Browse page
* Get front-end fully integrated with back-end

Priority Back-end Requirements

* Email verification (not done for ID3)
* Forgot password (not done for ID3)
* Confirm email
* Logout
* Contact Seller
* Sign in / Sign up using Facebook
* Delete Listing
* Get the front-end fully integrated with the back-end for all API calls

Other Back-end Requirements

* Thorough review of test cases for all back-end code
* Finalize and complete UI flow
* Push notifications if something changes regarding a Listing saved in Favourites (change in database triggers notification)
* Integration with Google Maps

## 7.2 Future Requirements

This section outlines requirements identified for upcoming ID’s, some of which will take priority over others, and some of which may not be implemented this term.

* Send a message to the seller including contact information
* Users can sign up to receive “hot list” notifications
  + Feature Listing (paid for by Sellers)
  + Newly added Listings
  + Price changes on Favourites
  + Based on previous search history
* Sellers receive notifications regarding personal Listings:
  + When a Listing is saved to Favourites
  + When someone requests a viewing
  + When someone sets a price watch
* Super Admin User
  + Log in as Super Admin
  + Add new Listing under any user
  + Edit any Listing
  + Remove any Listing

# 8.0 Triage Meeting ID3

**Date**:

**Start Time**:

**End Time**:

**Location**:

**Members Present**:

**Summary**:

**1) Front-end**

**2) Back-end**

**3) Testing**

**4) Build**