26/10/2018

Customer Records

# Introduction

This document reports on mini-project requested by Intercom as part of recruitment process for Mina Youssef for position of Sr. Software Engineer

## Development Stack

Python 3 w/ PyCharm

## Run Instructions

~/Intercom>python main.py --input\_file ./customers.txt --latitude 53.339428 --longitude -6.257664 --search\_radius 100

Contents

[Introduction 1](#_Toc528364122)

[Development Stack 1](#_Toc528364123)

[Run Instructions 1](#_Toc528364124)

[Design 2](#_Toc528364125)

[App Component 2](#_Toc528364126)

[Services Component 3](#_Toc528364127)

[Distance calculation kernel 3](#_Toc528364128)

[Testing 4](#_Toc528364129)

[Unit testing 4](#_Toc528364130)

[System Testing 4](#_Toc528364131)

[Special Case – Antipode 5](#_Toc528364132)

[Appendex I – Logging 6](#_Toc528364133)

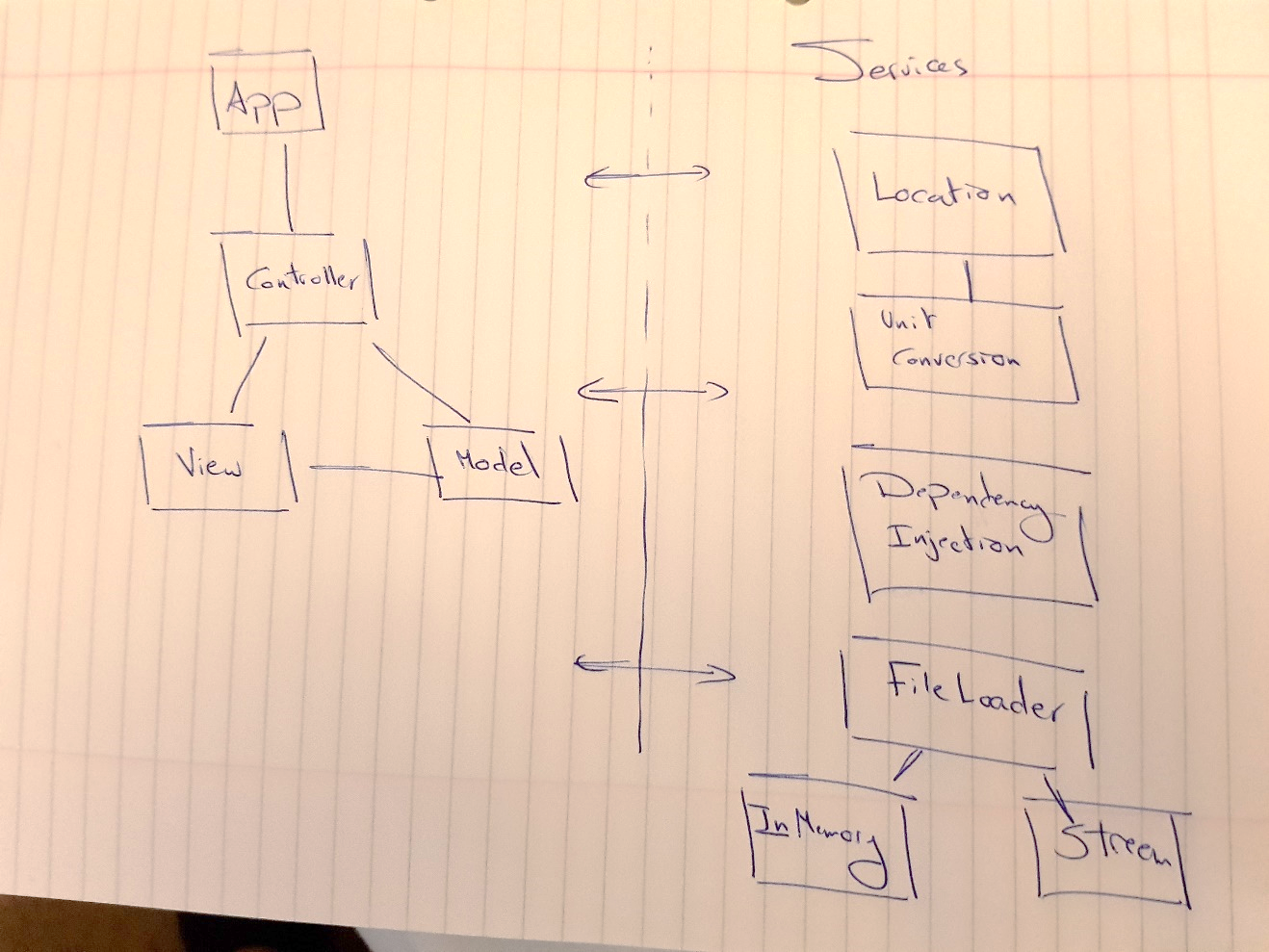
[Appendex II – Config 6](#_Toc528364134)

[Appendex III – Command Line ARGS 6](#_Toc528364135)

[Appendex – IV 8](#_Toc528364136)

# Design

The following diagram shows the basic building blocks



From high level design there are two main components

**App** – Implements MVC pattern and use façade class LocationApp that is called from main module

**Services** – Independent entities that provides uncorrelated functionality as services

## App Component

### Model

Defining the structures that is used like Location Class, DataRecord Class…etc

### Controller

Defines how execution is performed, I’ve two main approaches, single or multi-threaded.

And both shared the same base class so the façade class would use Strategy pattern in invoking the desired controller in runtime

NOTE: Due to time constraint, I've decided to submit without the Multi-threaded part

### View

I’ve implemented the view using Publisher/Subscriber pattern where controller is publisher and the print wrapper for the view class act as subscriber ~/common/PublisherSubscriber.py

## Services Component

There are 4 different services used by the application component as following:

LocationService (See [Distance Calculation Kernel Section](#_Distance_calculation_kernel))

### FileLoaderService

Provide a file loading service to application component, I’ve implemented two different policies for I/O

1. InMemory FileLoader
2. Stream File Loader

In case we have at least 70% of the free memory that can fit the target customer file then we will use the InMemory. Otherwise we use Stream file loader for big file size that won’t fit in the memory.

### DependencyInjectionService

Using DI for all instantiation

### UnitsConversionServices

Provide distance units conversion (currently between metre to kilometer but included for generality if empirical system would be required)

## Distance calculation kernel

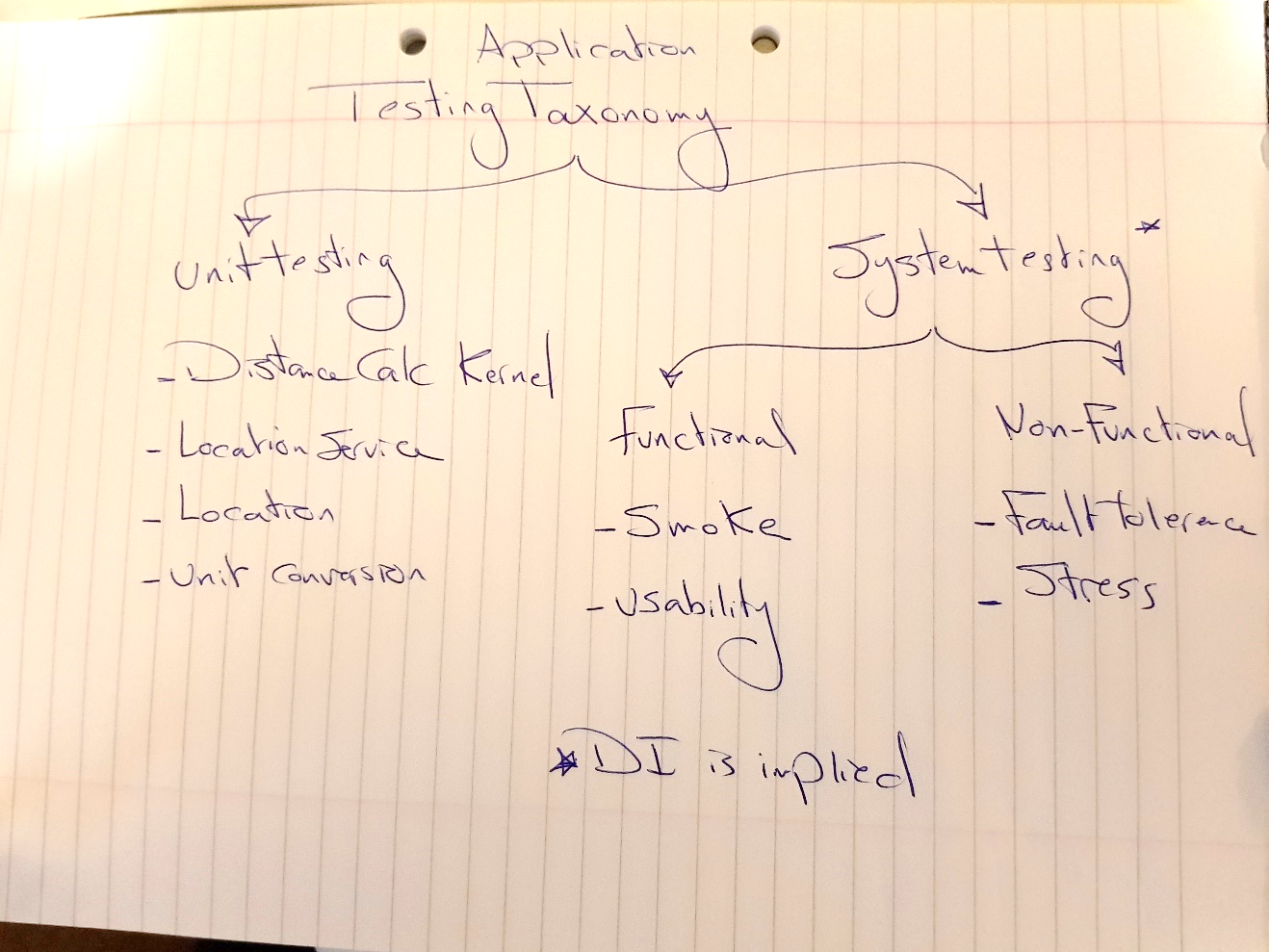
The core implementation for calculating a distance between two points is located in

DistanceCalculatorKernel (~\Intercom\Services\Location\DistanceCalculatorKernel.py)

From this kernel function we built LocationService class (~C:\data\dev\Intercom\Services\Location\LocationService.py) that would take a reference location (in our case Intercom office) and report the distance for all customers)

# Testing

The following shows a testing taxonomy for the application



We developed two categories of testing:

* Unit testing
* System testing

## Unit testing

In unit testing we focused on the services components which include

1. Distance Calculator Kernel Tester
2. Location Services Tester
3. Units Conversion
4. DI is tested as part of the system testing

## System Testing

For System testing we have two main parts

1. Functional System Testing
2. Non-Functional System Testing

### Functional System Testing

Include **Smoke testing** for quick validation

Include **Usability testing** for making sure application level parameters are respected by user

### Non-Functional System Testing

Include **Stress testing** for making sure application won’t misbehave with large files, in this test we take the content of the standard customer files and replicated for large file 50MB and allow the application to press all the data without any kind of failure.

Include **Fault Tolerance** testing make sure the passed file content is in correct format without missing invalid field(s)

And each of the above categories implement set of test cases methods for different combination. All under QA directory.

## Special Case – Antipode

From Wikipedia (<https://en.wikipedia.org/wiki/Great-circle_distance>) we used [Vincenty formula](https://en.wikipedia.org/wiki/Vincenty%27s_formulae)  that handles antipode corner case and a test case has been developed to test it at **DistanceCalculatorKernelTester.test\_valid\_antipode\_distance**

# Appendex I – Logging

As part of production tasks is to provide logging, I’ve added logging level

Every run will produce new log file stamped by data and time up to millisecond of the run.

Sample of log file for standard customer file



# Appendex II – Config

For application level configuration a class Config @ (~\Intercom\common\config.py) contains precision and default unit used

SEARCH\_PRECISION\_DEFAULT = 1  
"""  
Search precision defines searching rounding level, where 1 => 0.#, 2 => 0.## ...etc  
"""  
  
SEARCH\_DISTANCE\_UNITS = DistanceUnit.KILOMETRE  
"""  
Search distance units defines application level unit to be used in distances calculation.   
"""

# Appendex III – Command Line ARGS

Running application without any input would show all required command line arguments as following

C:\ProgramData\Anaconda3\python.exe C:/data/dev/Intercom/main.py

usage: main.py [-h]

**--input\_file** INPUT\_FILE

Path of Command line argument

**--latitude** LATITUDE

Reference latitude e.g. InterCom office Latitude

**--longitude** LONGITUDE

Reference Longitude e.g. InterCom office Longitude

**--search\_radius** SEARCH\_RADIUS

Searching radius that would accept the customer for invitiation (In KM) e.g. 100KM

[**--stop\_on\_failure** STOP\_ON\_FAILURE]

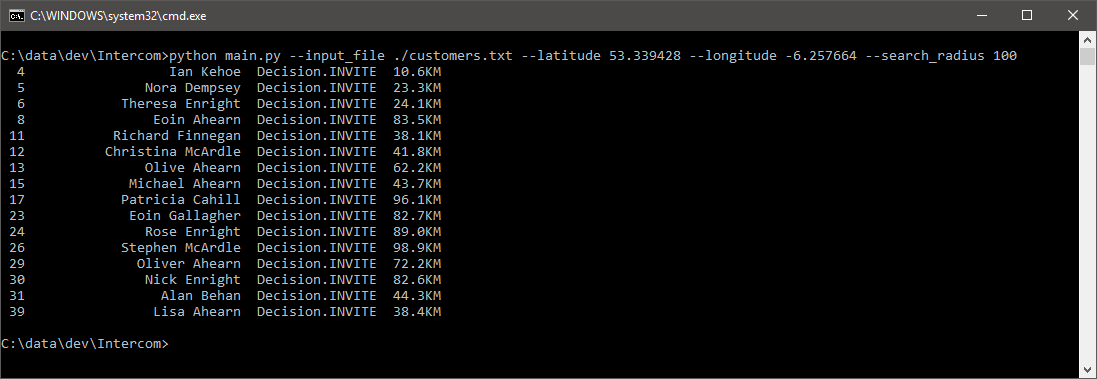
Flat to stop the execution of the application if one-line error, otherwise it will show that error and continue (default False)

[**--multithread-enable** MULTITHREAD\_ENABLE]

Enable multi-threading execution

main.py: the following arguments are required: --input\_file, --latitude, --longitude, --search\_radius

Sample Success Run:



# Appendex – IV

Testing Results

