**Software Implementation and Testing Document**

**For**

**Group <18>**

Version 1.0

**Authors**:

Alex Bundy

Kester Mbah

Johnathan Gutierrez-Diaz

Christopher Laughlin

Rodjna Pierre Louis

# Programming Languages (5 points)

HTML: Used for creating the user interface within the project. This includes the screen to signup, login, search apartments, add apartments, and just to exist on the home page. Those files that use it are AppApartment.html, Apartment Search.html, header.html, index.html, login.html, and signup.html. Html is very common to use for webpages so we went ahead with it.

CSS: Used to style our webpages in our styles.css file. Allows us to alter the style easier down the line with this file rather than styling each html file specifically. Essentially makes the code more able to change later as the requirements change.

JavaScript: Used to implement some of our interactive features like using the google maps to show where an apartment is and the distance matrix api to determine how far it is. This is used in both index.html and apartmentsearch.html to provide functionality, such as using the Google Maps API. JavaScript was used due to it being the common language for client-side processing in web development..

Python: In app.py and setup.py python is used to help manage both the databases and how the program flows. It specifies the routing for the html files, it initializes flask for the system, it connects the project to the database, and manages the backend for the html files like rendering them when necessary. Essentially python runs the backend of our program, with it managing the interactions between screens as well as the database for both accounts and apartments. Python was used due to its ability to interact with all these systems in a relatively simple manner.

# Platforms, APIs, Databases, and other technologies used (5 points)

Google Maps API: Used in index.html and apartmentsearch.html to display the locations of nearby apartments. We used this due to a map on these type of websites being common, and Google Maps specifically provides a nice looking easy to use map that we could add markers to indicate apartments.

Flask: Flask is used in app.py to handle server-side logic including routing between html pages, processing user input within these pages, and managing actual submissions like login, signup, and add apartment. Flask is good for this type of project due to its simplicity and it works well with python, supporting our language of choice n this project.

SQL Database: SQL database is setup in setup.py and app.py, with setup.py initializing it by establishing a connection to the database, creating the database, selecting the database, and creating the tables necessary within the database. In app.py the database actually becomes used, as of now it only interacts with signing up and logging in

Google Distance Matrix: This API is used in tandem with Google Maps API to calculate the distance between Florida State University and the apartment of choice within the application. This is used in both index.html and apartmentsearch.html. This provides extra calculations that could come into play when choosing an apartment close to campus.

Web Scraper: We created our own web scraper that goes to apartments.com and gets all the apartments near a specific place. For example, if the user enters in UCLA it will go to apartments.com and get the apartments that pop up near UCLA and we take that information and put it on our website.

Selenium: This tool is used to automate web browsers and was implemented to make the web scraper work with the application. It automates the process of retrieving the info for the chosen college and displaying it on the application.

# **Execution-based Functional Testing (10 points)**

We did test for the functional requirements listed in the Software Requirements and Design Document and found most of them to be fulfilled. The user registration and user login were tested by creating an account and logging into it once created, this was successful, so these two requirements are met. Once in I began to test both view apartment listings and apartment search, the requirement to view apartment listings was fulfilled with the main page listing various apartments for rent and apartment search was also fulfilled given that I was able to look one up in particular on the same screen. This fulfills both requirements. Traveling between pages was tested by going from the main page to the signup/signin page fulfilling this requirement. Map functionality was tested, and the apartments were shown in the right place making this requirement fulfilled. The requirement that a user is able to logout was tested by logging out after being logged in, this fulfills this requirement. Admin control is fulfilled given that when an admin logs in they have access to an admin dashboard which has been tested and is able to provide various tools for admin’s, providing control and fulfilling the requirement. Add apartment is available for admin’s in the background but as of now no apartment owner can list their apartment, not fulfilling this requirement. As of now save apartment listings does not function, not fulfilling this requirement. As of now there is no way of recovering a lost password not fulfilling this requirement. As of now users cannot manage their profile not fulfilling this requirement. As of now the distance calculation has not been implemented, not fulfilling this requirement. The user feedback portion was not implemented therefore this requirement is not met. The search feature has been extensively tested and works perfectly with the web scraper to properly display the correct apartments based on the college of choice. The edit apartment feature now functions where an admin is now able to edit the information on an input apartment as well as remove it entirely, fulfilling this requirement.

4. Execution-based Non-Functional Testing (10 points)

We did pursue some non-functional requirements listed in the Software Requirements and Design Document. The first one that was reviewed was the need for data integrity and so far, the initial data we input towards the start of this increment has not changed so I believe this requirement is met. The second was that has been relatively reviewed is scalability, as of now there has been a change of performance since including the web scraper but due to this web scraper allowing for no new manual data entry I believe this will be the only performance drop ensuring scalability. I believe this also meets the requirement of performance in that the performance can only be improved form this point forward with techniques like caching and as of now it is still a realistic load time even without these techniques. Reliability has also been essentially confirmed since the database’s creation and since the initial figuring out of the ip’s there has not been an issue with data integrity. I believe with the current structure this also fits the requirement of maintainability, with the addition of the web scraper this site is essentially able to maintain itself with it fetching its own information. Security I feel is lacking with the lack of an ability to change account information once a password is leaked the account is basically destroyed.

5. Non-Execution-based Testing (10 points)  
Describe how/if you performed non-execution-based testing (such as code  
reviews/inspections/walkthroughs).

There was some non-execution-based testing throughout the project such as code reviewing and inspecting. Most essential documents have been reviewed by someone other than the author to look for issues that may have been overlooked such as misnamed variables, improper syntax, and useless code. The general review for code has usually just been left for one other pair of eyes to review rather than the whole team to save time. This person is usually determined simply by who is available at the time for review. I believe this is the only non-execution-based testing we would have done up to this point.