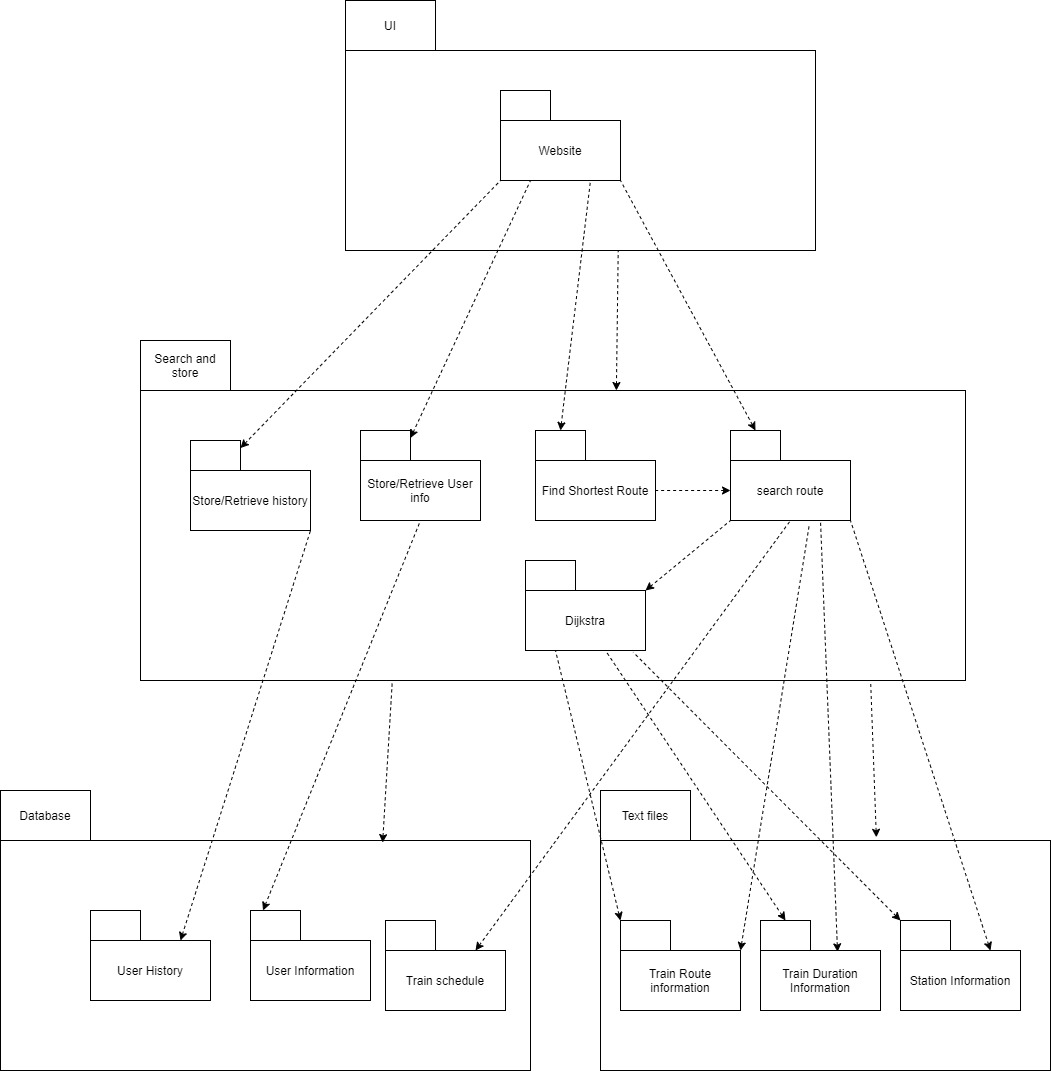
Testing:

Our program has gone under severe testing to ensure that it is bug and error free. To ensure that the program is bug free, we have implemented multiple testing listed in the table below

|  |  |
| --- | --- |
| Process | Technique |
| 1. Class Testing: test methods and state behaviour of classes | Random, Partition and White-Box Tests |
| 1. Integration Testing: test the interaction of sets of classes | Random Tests |
| 1. Validation Testing: test whether customer requirements are satisfied | Spelling, Use-case based black box and Acceptance tests |
| 1. System Testing: test the behaviour of the system as part of a larger environment | Recovery, security, and performance tests |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type of test | Why | How | Result | Normal cases | Extreme cases |
| White Box Testing | To ensure that the input and output are correct | We have used multiple if else and print () statement in the program. We will have an expected output in our head and compare that output with the actual output. | If the result passes, then we continue, otherwise we will fix the code before continuing. | passed | N/A |
| Random Testing | To ensure that the functions will work with random inputs and outputs | test all functions and compare it with its output. Use random values as our input, formulate the output ourselves and compare the result using junits. | If we fail something in junits, then we try to figure out why. If we pass it, then we continue with the remaining work | Passed | Passed |
| Partition Testing | To ensure that the functions will work with any kind of inputs and outputs that represent different kind of inputs. | We use values like “Kraaifontein to Brackenfell on Monday”, if this works, then all trains in that partition (Area North with only 1 train) should work. You can see out inputs in testing.py | If one testing fails, we assume that all partition related to that test fails. So we fix the problem related to that partition | Passed | Passed |
| Acceptance test | To see whether the user will be able to use the program to their satisfaction | We have manually checked whether the output created is correct and able to use. We see if the shortest time is the shortest if the closest time to depart is the closest time. | When we found something that we do not like about the output, we tried to change our code so that the output is acceptable | Passed | Passed |
| Use-case based black box: | To ensure the customers satisfactory | We have run the website on different computers to ensure that it works with them. We have asked multiple people to use the application to hear from their feedback. | The website functions just fine with different computers. From feedbacks, We improve our website for better user experience | Passed | N/A |
| Recovery test | To ensure that data is not lost due to break down or accidental close of the application | We do many different testings, once bug is found, we fix them immediately. We tried to store then close website immediately, check if the database was able to record the data. | The program is working better every time we fix a bug. Everything is recorded in the database, so nothing is lost when website is closed suddenly after the clicking of the button | Passed | Passed |
| Security test | To ensure the privacy of the users | we have created a user and password protection to ensure that all users have their own account if they want it. We tried to log in with different account using different passwords. Checking if the user can log in. | When the user enters an username with wrong password, there will be an error preventing them from accessing the other account’s data | Passed | N/A |
| Performance test | To improve efficiency | We import time.  User time.start() and time.time() to measure the time taken to execute different inputs | Simon’s Town – Strand on Monday took 0.006  Mowbray – Stellenbosch on Saturday took 0.005  Cape Town to Avondale on Monday took 0.004.  The time normally won’t go over 0.01 seconds | Passed | Passed |
| Spelling Testing | To ensure that all spellings of the train stations are correct | We have written a python file TextOrganizer.py. This file will go through all the Train Route and compare them with the duration, If anything is missing or misspelt, it will return an error | We find the location that caused the error. Edit text files then rerun the program to check for if we have any more errors | Passed | Passed |



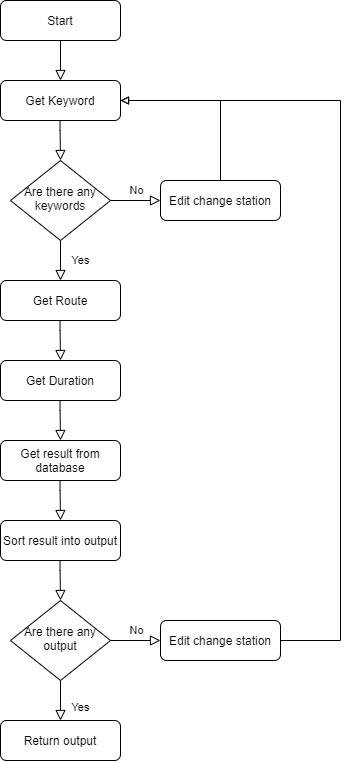
Architectural diagram

There are 4 main sections in our program:

* UI: This is the front end of the program that the user will see and interact with
* Search and store: This is the heart of the backend that executes commands depending on the user inputs. They take information from database and text files and generate an output for displaying on the website
* Database: This is where all the information about the trains, user, and their histories are stored.
* Text files: This is a package that provides addition information for search and store to compute and generate output.

How classes interact to find train route:

The core file is called searchRoute, the website will be calling the function to find all the routes that goes from start to end. This returns a 2d array in this format: [list of routes] [route information]. Dijkstra is used for finding the shortest path between two locations. The information from Dijkstra will be passed to searchRoute. searchRoute uses this information and get a list of the routes. shortestRoute is used for finding the shortest duration and closest time to the user wanted departure time.



This is a flowchart of how search () function in searchRoute works

List of important sections in the search ()

Get Keyword: This is the most important function in the program, this will return a list of unique keys that represents all the routes that a user can take from start to end destination. All other functions build up from this function

Get Route + Get Duration: This is the build up from getKeyword. It will first find all the stations that the train will go pass, use that information combined with their duration information will return the time it takes for the train to travel from start to end.

Get result from database and sort result into output: This is retrieving the data from the database and sort it before other programs uses it.

Special programming techniques: We used SQL queries to help us find the train routes from the database. SQL queries are good at searching and sorting information throughout the whole database with simple lines. We used PyPDF and its build in functions to help us easily read the pdf. I also used time library to help me improve the efficiency of the program. We can limit the runtime below 0.01 seconds which is really fast considering all the information we need to search through.