|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BlueT_nobackgroundcmyk_sml | | | | | logo | | | |
|  | | | | |  | | | |
|  | | | | | | |
| HPSE Assessment - Design Document | | | | | | |
|  | | |  |  | | | |
| Status: | Draft | | | |
| Version: | 0.1 | | | |
| Date: | **16 Nov 2018** | | | |
| Project ID |  | | | |

# Document Purpose

This document provides the Detail Design for HPSE Assessment Usecase to identify the shortest path to each city from origin city and return to origin city. The purpose of this document is to capture the implementation and data flow required to satisfy the requirements.

* It enables user to find the shortest path for various input scenarios.
* Able to see the historical requests submitted to find the shortest path.
* It gives the Graph view for better understanding of what user is trying to achieve

# Scope & Out of Scope

# Scope

* The scope is to implement the shortest path algorithm to identify the shortest path between each city and return to origin city
* Capability of HOSE Assessment to Submit request with inputs and identify the shortest path.

**Road Map for HPSE Assessment**

* Minimum Viable Product developed initially will comprise of the below listed components
  + - User will be able to submit the request with the input to identify the shortest path

• Plan is to extend this to deploy in three tier architecture

• Use Vagrant and create a three tier architecture VM with Proxy in tier one, App VM in server 2 and DB VM in Server 3

• Plan is to write a script which will create a VM and deploy the docker image and establish the connection with DB server and make entry in HAProxy config and restart the service.

# Requirement Reference



# Assumptions and Dependencies



# Assumptions

* + - A simple application is required to find the shortest path algorithm to visit each city and return to origin city
    - The docker version used to create a image should be available with end user to pull the image from docker hub and execute in his/her own machine.

# Dependencies

* + - Due to policy restrictions in installing Vagrant, the Use case 2 is pending to start with.

# Key Architecture Decisions

The following is the key architecture decisions that would need to be addressed as part of the solution definition for HPSE Assessment

|  |  |  |  |
| --- | --- | --- | --- |
| Sl. No | Dependency Description | Impacted Systems | Priority |
| AD-01 | Identifying the best suitable algorithm to find the shortest path | NA | 1 |

Table #1

| **Subject Area** | **Role management** | | |
| --- | --- | --- | --- |
| **Architectural Decision** | Which algorithm suits best to identify the shortest path for the given input with less processing time and avoid duplicate execution of same path to ensure less processing time? | | |
| **OPTION 1** | **Brute Force Algorithm:** The algorithm is simple and will always find a solution if it exists, its cost is directly proportional to the number of vertices which in turn many problems tends to grow very quickly as the size of problem increases. This is typically used when there is a reduced set of vertices defined | | |
| **OPTIION 2** | **Bellman-Held-Karp algorithm:** This algorithm is a dynamic programming algorithm. It computes the solution for all sub problems starting with the smallest. This algorithm is mostly used in all kind of Travelling Salesman Problem (extension on Hamilton circuit problem) | | |
| **OPTION 3** | **Nearest Neighbor: This algorithm** is one of the first [algorithms](https://en.wikipedia.org/wiki/Algorithm) used to determine a solution to the [Travelling Salesman Problem](https://en.wikipedia.org/wiki/Travelling_salesman_problem). In it, the salesman starts at a random city and ***repeatedly visits the nearest city*** until all have been visited. It quickly yields a short tour, but usually not the optimal one. | | |
|  |  |  |  |
| **Decision** | Option 2 is identified as the best practices since the Bellman-Hel-Karp uses dynamic programming and the Time complexity is O(n2 2n) which is less compared to Brute force and Nearest Neighbor algorithms | | |