CAMPUS NAVIGATION SYSTEM

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

DATA STRUCTURES

PROJECT REPORT

PARINIKA KATH – 22103094

B4 Batch

PROBLEM STATEMENT

There exist many advanced navigation systems but most of them are unable to provide routes precisely as well as information of buildings within a region such as campus, shopping mall, hospital, etc. Nowadays, as people are getting more and more connected to technology, they have lost their human touch. Also, people feel more convenient to search for the problem themselves rather than asking someone for help.

An informative, reliable and precise guidance system is very important in this technological era. It should be able to navigate the user no matter if the user is under the indoor or outdoor environment. The guidance system must be user-friendly and able to process data efficiently. Since the size of any college/university campus can vary from 30 acres to anywhere around 200 acres, students spend the majority of their time traveling between different buildings. New students feel it is inconvenient to search their way inside the campus. Therefore, a navigation system is required to find the optimal path within the campus and for the aforementioned problem.

BASIC CONCEPTS AND TOOLS USED:

- Linked List (Circular LL)
- Graphs
- Trees (Shortest Path Tree)

MAIN FUNCTIONING OF THE PROGRAM

Dijkstra’s algorithm is for minimum spanning tree. In Dijkstra’s we generate a SPT **(shortest path tree)** with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, and other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source. The Dijkstra’s algorithm maintains two sets of vertices which are: L: Labeled vertices (shortest path is known) C: Candidate vertices (shortest path not yet known) One vertex is moved from C to L in each iteration

The steps to be followed for completing the proposed project are as follows: 1 Identification of key locations in campus infrastructure- Key Structural points in the campus are to be identified, which work as the node points for the Dijkstra’s Algorithm 2 Distance Estimation -Estimated distance is calculated which works as the weight between **two adjustment nodes**, hence a **weighted directed graph** is generated. 3 Proposing the shortest Path to the User- Using the Dijkstra’s Algorithm the program will **suggest the shortest optimal path.**

FLOWCHART:

START THE PROGRAM

|

|

ESTIMATE DISTANCE BETWEEN VARIOUS CAMPUS INFRASTRUCTURE

|

|

MAP DESIGNING

|

|

ALGORITHM IMPLEMENTATION AND CODING

|

|

MAP DESIGN AND ALGORITHM INTEGRATION

|

|

FINALISING..

|

|

PROGRAM ENDS!

SHORTEST PATH FOUND AND DISPLAYED!

PROJECT CODE

```
#include <iostream>

#include <vector>

#include <limits>

#include <map>

#include <stack>
```

```
using namespace std;

const int INF = numeric_limits<int>::max();

class Graph

{

//class creation

public:

    int vertices;

    vector<string> buildingNames;

    vector<vector<pair<int, int>>> adjList;

//Initialize variables

    Graph(int V) : vertices(V), adjList(V)

    {

        //GRAPH

    }

    void addEdge(int src, int dest, int weight)
```

```
  {

      adjList[src].emplace_back(dest, weight);

      adjList[dest].emplace_back(src, weight);

  }

  vector<int> dijkstra(int src, map<int, int>& predecessors)

  {
//performing dijkstra
//to find the shortest path
//between any 2 buildings
      vector<int> distance(vertices, INF);

      vector<bool> visited(vertices, false);



      distance[src] = 0;



      for (int i = 0; i < vertices - 1; ++i)

        {
```

```
    int u = minDistance(distance, visited);

    visited[u] = true;



    for (const auto &neighbor : adjList[u])

      {

      int v = neighbor.first;

      int weight = neighbor.second;



      if (!visited[v] && distance[u] != INF && distance[u] + weight < distance[v])

        {
        distance[v] = distance[u] + weight;

        predecessors[v] = u;
      }

    }

  }
  return distance;
```

```
}
int minDistance(const vector<int> &distance, const vector<bool> &visited)

{

    int minDist = INF;

    int minIndex = -1;



    for (int v = 0; v < vertices; ++v)

    {

        if (!visited[v] && distance[v] < minDist)

            {

            minDist = distance[v];

            minIndex = v;

        }

    }
    return minIndex;
```

```
   }



   void printSolution(const vector<int> &distance, const map<int, int>& predecessors, int
dest)

   {
//display solution function

       cout << "Shortest distance from " << buildingNames[0] << " to " <<
buildingNames[dest] << ": " << distance[dest] << " units\n";



       cout << "Shortest path: ";

       stack<int> path;

       int current = dest;

       while (current != -1)

           {

           path.push(current);

           current = predecessors.at(current);

       }
```

```
    while (!path.empty())

        {

        cout << buildingNames[path.top()] << " -> ";

        path.pop();

    }

    cout << endl;

}



map<pair<string, string>, int> calculateDistances()

{

    map<pair<string, string>, int> distanceMap;



    for (int i = 0; i < vertices; ++i)

        {

        for (const auto &neighbor : adjList[i])
```

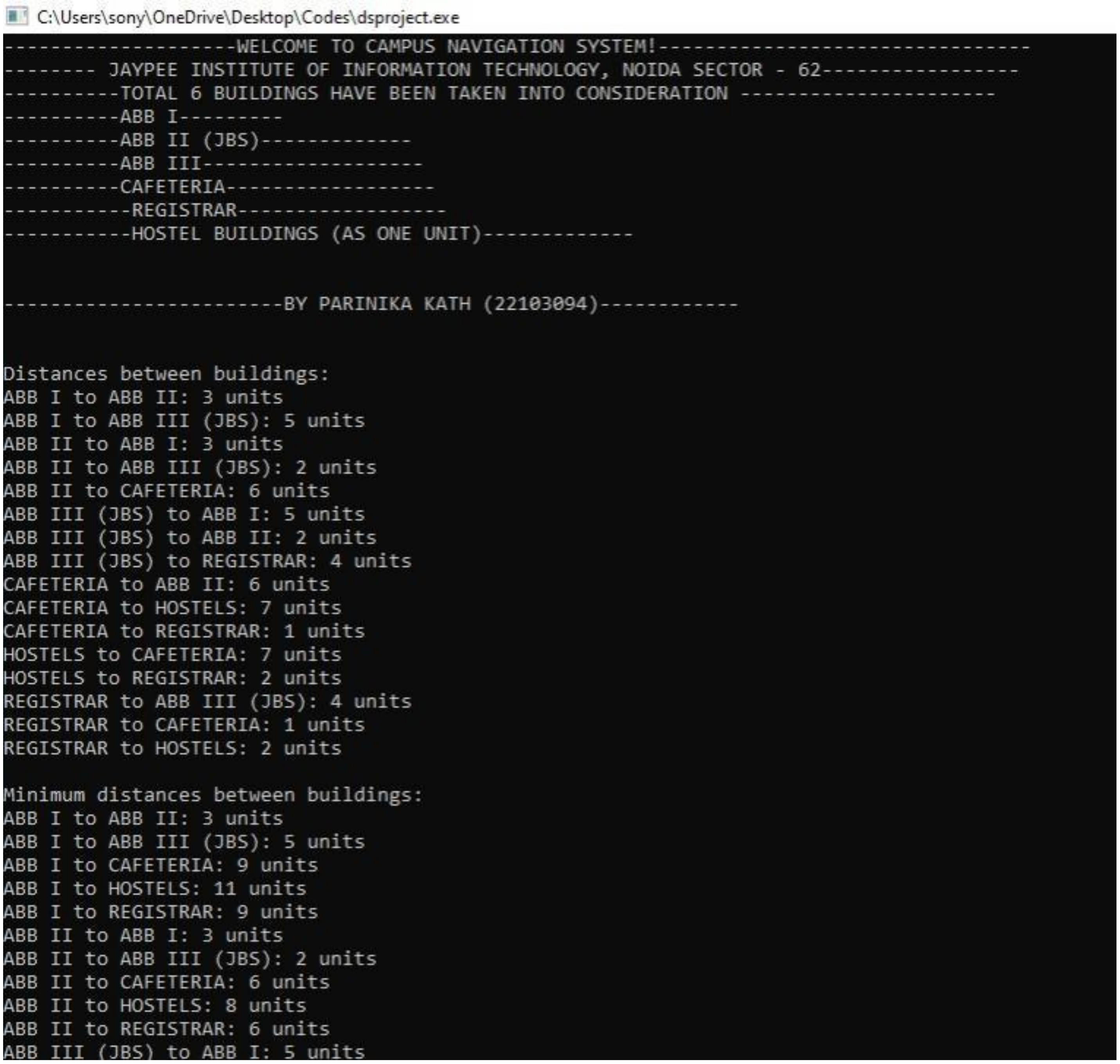
```
         {

            int v = neighbor.first;

            int weight = neighbor.second;


            distanceMap[{buildingNames[i], buildingNames[v]}] = weight;

         }

      }
      return distanceMap;
//min distance returned
//to main
   }
   map<pair<string, string>, int> calculateMinDistances()
   {

      map<pair<string, string>, int> minDistanceMap;
      for (int i = 0; i < vertices; ++i)

         {
         map<int, int> predecessors;

         vector<int> distances = dijkstra(i, predecessors);
```

```
          for (int j = 0; j < vertices; ++j)

             {

             if (i != j)

             {
                minDistanceMap[{buildingNames[i], buildingNames[j]}] = distances[j];

             }
          }

       }

       return minDistanceMap;
    }
};
int main()
{
//main starts
    cout<<"--------------------WELCOME TO CAMPUS NAVIGATION SYSTEM!-------------------------------
-"<<endl;

    cout<<"-------- JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA SECTOR - 62-----
------------"<<endl;
    cout<<"----------TOTAL 6 BUILDINGS HAVE BEEN TAKEN INTO CONSIDERATION ----------------
------"<<endl;
```

```
    cout<<"----------ABB I---------\n";

    cout<<"----------ABB II (JBS)-------------\n";

    cout<<"----------ABB III-------------------\n";

    cout<<"----------CAFETERIA------------------\n";
    cout<<"-----------REGISTRAR------------------\n";
    cout<<"-----------HOSTEL BUILDINGS (AS ONE UNIT)-------------\n";
    cout<<endl;

    cout<<endl;
    cout<<"------------------------BY PARINIKA KATH (22103094)------------\n";

    cout<<endl;

    cout<<endl;

    Graph campus(6);

    campus.buildingNames = {"ABB I", "ABB II", "ABB III (JBS)", "CAFETERIA", "REGISTRAR",
"HOSTELS"};

    campus.addEdge(0, 1, 3);
//abb 1

    campus.addEdge(0, 2, 5);
```

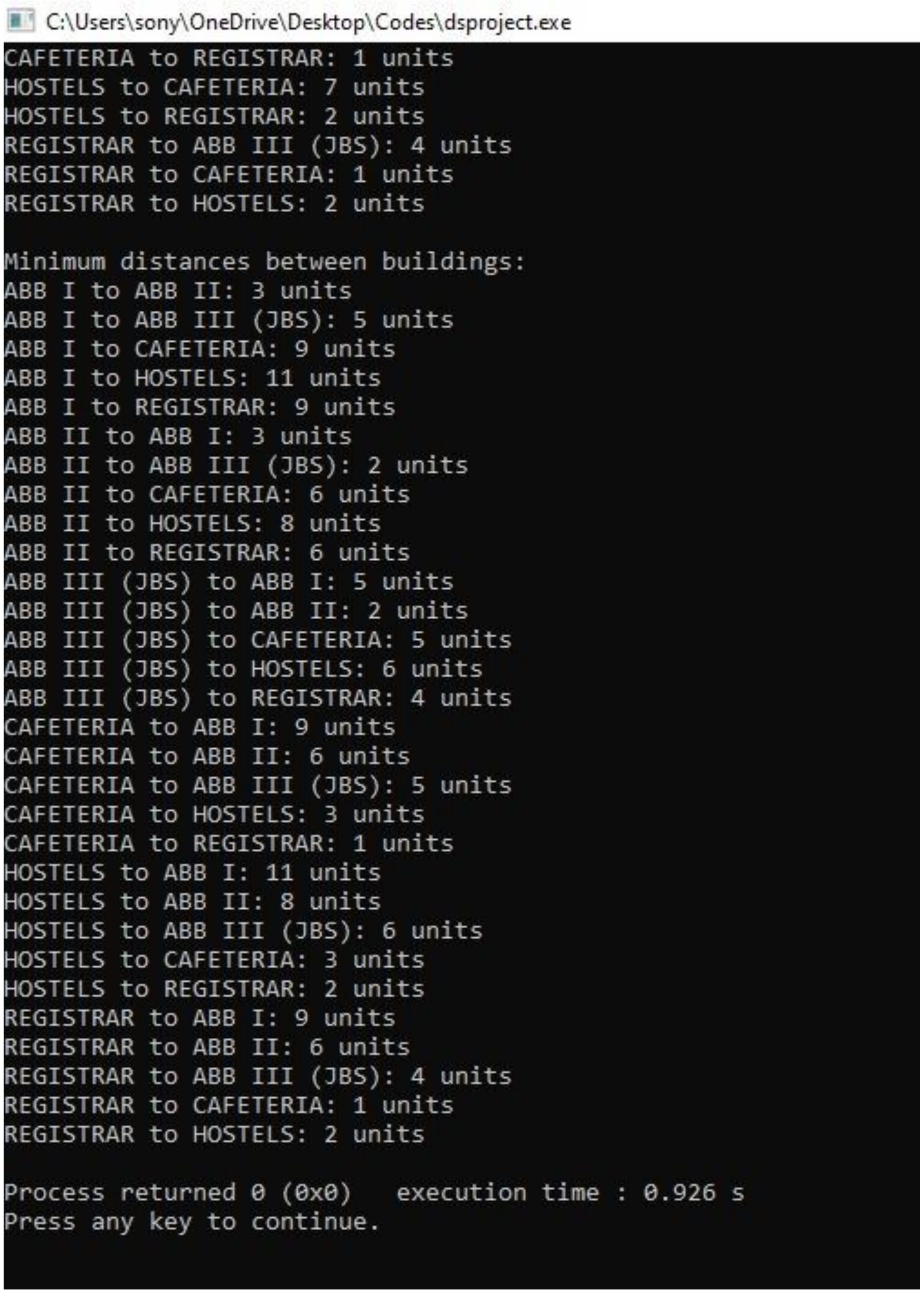
```
//abb 2
//jbs

    campus.addEdge(1, 2, 2);
//abb 3

    campus.addEdge(1, 3, 6);
//cafeteria

    campus.addEdge(2, 4, 4);
//registrar

    campus.addEdge(3, 4, 1);
//hostel

    campus.addEdge(3, 5, 7);
//hostel
    campus.addEdge(4, 5, 2);

//done adding edges

    map<pair<string, string>, int> distanceMap = campus.calculateDistances();
//map creation
    // Calculate and display distances between all pairs of buildings

    cout << "Distances between buildings:\n";
    for (const auto &entry : distanceMap)

    {

        cout << entry.first.first << " to " << entry.first.second << ": " << entry.second << "
units\n";

    }
    // Calculate and display minimum distances between all pairs of buildings
    map<pair<string, string>, int> minDistanceMap = campus.calculateMinDistances();
    cout << "\nMinimum distances between buildings:\n";
    for (const auto &entry : minDistanceMap)
        {
        cout << entry.first.first << " to " << entry.first.second << ": " << entry.second << "
units\n";
    }
    return 0;
//main ends
}
```

OUTPUT SCREENSHOT:



C:\Users\sony\OneDrive\Desktop\Codes\dsproject.exe

```
--------------------WELCOME TO CAMPUS NAVIGATION SYSTEM!--------------------------------
-------- JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA SECTOR - 62-----------------
----------TOTAL 6 BUILDINGS HAVE BEEN TAKEN INTO CONSIDERATION ----------------------
----------ABB I---------
----------ABB II (JBS)-------------
----------ABB III------------------
----------CAFETERIA-----------------
-----------REGISTRAR-----------------
-----------HOSTEL BUILDINGS (AS ONE UNIT)-------------


------------------------BY PARINIKA KATH (22103094)------------


Distances between buildings:
ABB I to ABB II: 3 units
ABB I to ABB III (JBS): 5 units
ABB II to ABB I: 3 units
ABB II to ABB III (JBS): 2 units
ABB II to CAFETERIA: 6 units
ABB III (JBS) to ABB I: 5 units
ABB III (JBS) to ABB II: 2 units
ABB III (JBS) to REGISTRAR: 4 units
CAFETERIA to ABB II: 6 units
CAFETERIA to HOSTELS: 7 units
CAFETERIA to REGISTRAR: 1 units
HOSTELS to CAFETERIA: 7 units
HOSTELS to REGISTRAR: 2 units
REGISTRAR to ABB III (JBS): 4 units
REGISTRAR to CAFETERIA: 1 units
REGISTRAR to HOSTELS: 2 units

Minimum distances between buildings:
ABB I to ABB II: 3 units
ABB I to ABB III (JBS): 5 units
ABB I to CAFETERIA: 9 units
ABB I to HOSTELS: 11 units
ABB I to REGISTRAR: 9 units
ABB II to ABB I: 3 units
ABB II to ABB III (JBS): 2 units
ABB II to CAFETERIA: 6 units
ABB II to HOSTELS: 8 units
ABB II to REGISTRAR: 6 units
ABB III (JBS) to ABB I: 5 units
```



C:\Users\sony\OneDrive\Desktop\Codes\dsproject.exe

```
CAFETERIA to REGISTRAR: 1 units
HOSTELS to CAFETERIA: 7 units
HOSTELS to REGISTRAR: 2 units
REGISTRAR to ABB III (JBS): 4 units
REGISTRAR to CAFETERIA: 1 units
REGISTRAR to HOSTELS: 2 units

Minimum distances between buildings:
ABB I to ABB II: 3 units
ABB I to ABB III (JBS): 5 units
ABB I to CAFETERIA: 9 units
ABB I to HOSTELS: 11 units
ABB I to REGISTRAR: 9 units
ABB II to ABB I: 3 units
ABB II to ABB III (JBS): 2 units
ABB II to CAFETERIA: 6 units
ABB II to HOSTELS: 8 units
ABB II to REGISTRAR: 6 units
ABB III (JBS) to ABB I: 5 units
ABB III (JBS) to ABB II: 2 units
ABB III (JBS) to CAFETERIA: 5 units
ABB III (JBS) to HOSTELS: 6 units
ABB III (JBS) to REGISTRAR: 4 units
CAFETERIA to ABB I: 9 units
CAFETERIA to ABB II: 6 units
CAFETERIA to ABB III (JBS): 5 units
CAFETERIA to HOSTELS: 3 units
CAFETERIA to REGISTRAR: 1 units
HOSTELS to ABB I: 11 units
HOSTELS to ABB II: 8 units
HOSTELS to ABB III (JBS): 6 units
HOSTELS to CAFETERIA: 3 units
HOSTELS to REGISTRAR: 2 units
REGISTRAR to ABB I: 9 units
REGISTRAR to ABB II: 6 units
REGISTRAR to ABB III (JBS): 4 units
REGISTRAR to CAFETERIA: 1 units
REGISTRAR to HOSTELS: 2 units

Process returned 0 (0x0)   execution time : 0.926 s
Press any key to continue.
```

REQUIREMENT SPECIFICATION:-

C++ code complier

CONCLUSION :

The provided C++ program implements a campus navigation system using Dijkstra's algorithm to find the shortest paths between key locations (buildings) within the campus. The program includes features such as:

1. Graph Representation: Buildings are represented as nodes in a graph, and weighted edges represent the distances between buildings.

2. Dijkstra's Algorithm: The algorithm is employed to calculate the shortest paths from a selected source building to all other buildings in the campus.

3. Building Names: Each node (building) is associated with a name (e.g., Building 1, Building 2), enhancing user-friendliness.

4. Distance Calculation: The program calculates and displays the distances between all pairs of buildings and the minimum distances between them.

5. Path Reconstruction: For each pair of buildings, the program reconstructs and displays the shortest path, providing users with optimal navigation guidance.

Overall, the project aims to address the navigation challenges within a campus, offering an informative and user-friendly solution that leverages Dijkstra's algorithm for efficient pathfinding.

FUTURE SCOPE:

1. **Enhanced User Interface:** Develop a user-friendly mobile or web application with a visually intuitive interface.

2. **Real-Time Updates:** Integrate real-time data for dynamic route adjustments based on current conditions.

3. **Indoor Navigation:** Extend navigation support to indoor spaces using technologies like Bluetooth or Wi-Fi positioning.

4. **Voice Guidance:** Implement turn-by-turn voice guidance for a hands-free navigation experience.

5. **User Preferences:** Allow customization of routes based on user preferences, such as avoiding stairs or choosing well-lit paths.

6. **Crowdsourced Feedback:** Enable users to provide feedback and report changes, incorporating crowdsourced data for continuous improvement.

7. **Integration with Campus Services:** Include information about campus services and facilities in the navigation system.

8. **Augmented Reality (AR):** Explore AR technology for overlaying navigation instructions onto the user's real-world view.

9. **Machine Learning Optimization:** Use machine learning to optimize route suggestions based on historical data and user behavior.

10. **Multi-Campus Support:** Extend the system to cover larger or multiple campuses, adapting to diverse infrastructures.

11. **Security Features:** Integrate safety features, such as emergency routes, to enhance user security.

12. **Cross-Platform Compatibility:** Ensure compatibility with various devices and platforms for widespread accessibility.

REFERENCES:-

1. https://www.kashipara.com/project/idea/java/offline-campus-navigation-system_3043.html
2. https://www.daniweb.com/programming/software-development/threads/472164/real-time-traffic-softwares-and-maps
3. https://www.behance.net/search/projects/?search=Campus+navigation