VISVESVARAYA TECHNOLOGICAL UNIVERSITY,
BELAGAVI 590018

Project Report on

"Campus Route Finder"

By

PRAJWAL S (1BM24CS209)
PRAMOD SUBHASH CHOUGULE (1BM24CS211)
PRAMODH RAO H (1BM24CS212)
PRANAV R REDDY (1BM24CS216)

Under the Guidance of

Prof. Monisha HM
Assistant Professor, Department of CSE
BMS College of Engineering

Work Carried Out At

Department of Computer Science and Engineering
BMS College of Engineering
(Autonomous college under VTU)
P.O. Box No.: 1908, Bull Temple Road, Bangalore-560 019
2025-2026

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the OOPS with JAVA project titled "**Campus Route Finder**" has been carried out by PRAJWAL S (1BM24CS209), PRAMOD SUBHASH CHOUGULE (1BM24CS211), PRAMODH RAO H (1BM24CS212), PRANAV R REDDY (1BM24CD216) during the academic year 2025-2026.

Signature of the guide
Prof. Monisha HM
Assistant Professor,
Department of Computer Science and Engineering
BMS College of Engineering, Bangalore

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, PRAJWAL S (1BM24CS209), PRAMOD SUBHASH CHOUGULE (1BM24CS211), PRAMODH RAO H (1BM24CS212), PRANAV R REDDY (1BM24CS216), students of 3rd Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this project work entitled "**CAMPUS ROUTE FINDER**" has been carried out by us under the guidance of Prof. Monisha HM**,** Assistant Professor, Department of CSE, BMS College of Engineering, Bangalore during the academic semester Sep-Dec 2025. We also declare that to the best of our knowledge and belief, the project reported here is not from part of any other report by any other students.

Signature of the Candidates

PRAJWAL S (1BM24CS209)

PRAMOD SUBHASH CHOUGULE (1BM24CS211)

PRAMODH RAO H (1BM24CS212)

PRANAV R REDDY (1BM24CS216)

PROBLEM STATEMENT

Navigating a large college campus can be confusing and time-consuming, especially for new students, visitors, and guests who are unfamiliar with the layout. Campuses often consist of multiple buildings, hostels, administrative blocks, libraries, cafeterias, playgrounds, and junctions connected by a network of roads and pathways. Without clear guidance, individuals may struggle to locate destinations efficiently, leading to delays, frustration, and reduced productivity.

Currently, most institutions rely on static maps, printed guides, or verbal directions from peers and staff. These methods are often inadequate because they do not provide real-time, step-by-step navigation or account for multiple possible routes. In addition, curved roads, junctions, and complex pathways make it difficult to visualize the shortest or most accessible path between two points.

The absence of a digital, interactive solution creates a gap in campus usability. Students waste valuable time searching for classrooms or offices, visitors face difficulties during events or admissions, and staff members are frequently interrupted to provide directions. This problem becomes more significant during the start of academic sessions when freshers and parents are new to the environment.

To address this challenge, the **Campus Route Finder project** aims to design and implement a JavaFX-based application that converts a campus sketch into a functional digital map. The system will allow users to select a source and destination, calculate the shortest or most efficient route, and display clear directions along with distance and estimated travel time. By modeling the campus as a graph of nodes (locations, junctions) and edges (paths with distances), the application will provide accurate navigation while remaining simple and user-friendly.

This project not only enhances campus accessibility but also demonstrates practical application of data structures, algorithms, and user interface design. It bridges the gap between theoretical learning and real-world usability, offering a solution that can be scaled and adapted for any institution.

INTRODUCTION

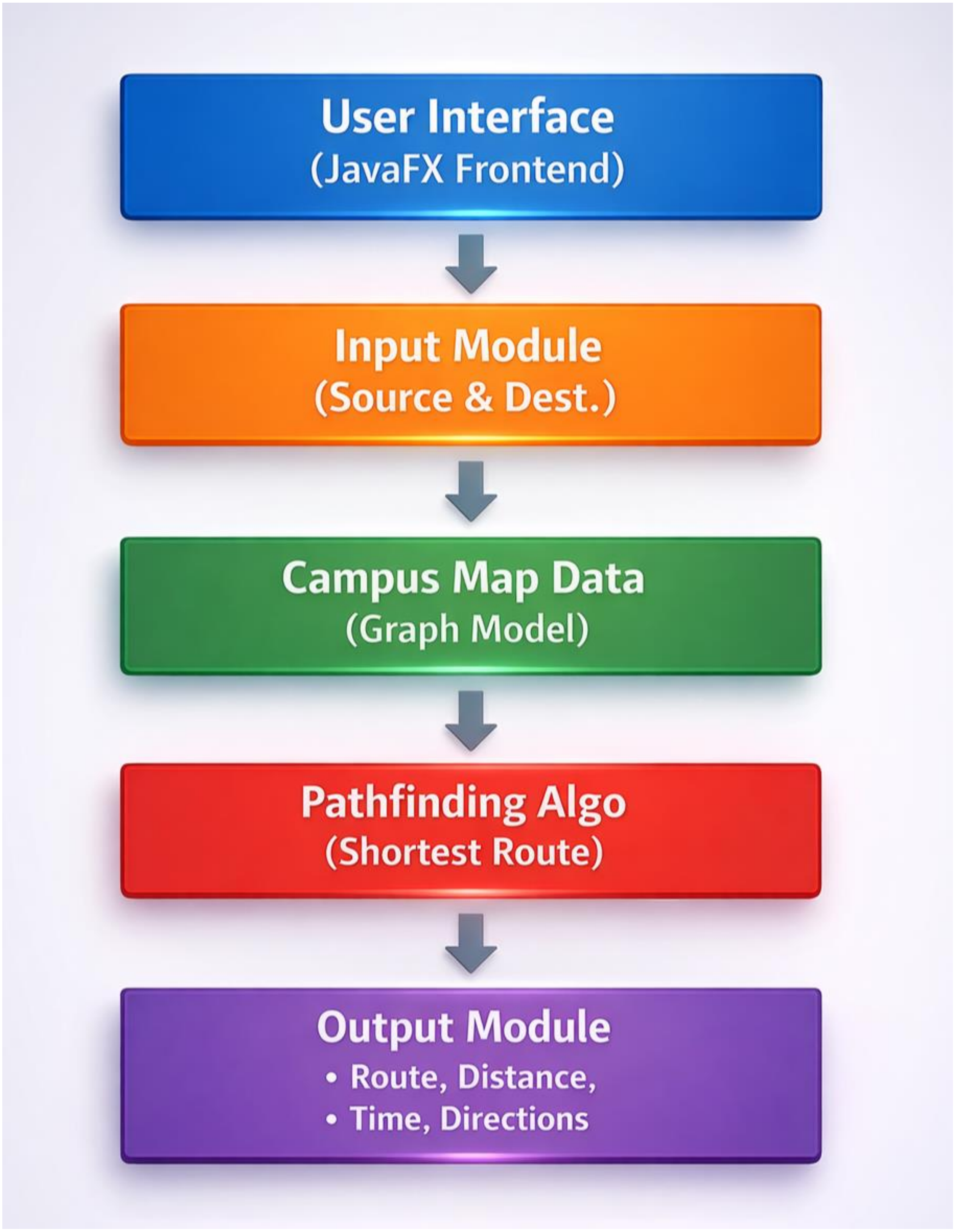
Large educational institutions often span vast areas with multiple buildings, hostels, administrative blocks, libraries, cafeterias, playgrounds, and laboratories interconnected by a network of roads and pathways. For new students, visitors, and even staff members, navigating such a campus can be confusing and time-consuming. Traditional solutions such as printed maps, notice boards, or verbal directions are static, limited, and fail to provide personalized guidance. As a result, individuals may struggle to locate destinations efficiently, leading to delays, frustration, and reduced productivity.

With the increasing reliance on digital technologies, there is a growing need for interactive campus navigation systems that can provide clear, step-by-step directions. A digital route finder not only improves accessibility but also enhances the overall campus experience by offering accurate information about distances, estimated travel times, and the shortest or most convenient paths. Such systems are particularly valuable during admission seasons, orientation programs, and large events when many newcomers are unfamiliar with the campus layout.

The **Campus Route Finder project** has been developed to address this challenge. Using JavaFX as the primary framework, the project models the campus as a graph consisting of nodes (representing buildings, hostels, and junctions) and edges (representing roads or pathways with associated distances and travel times). By applying shortest-path algorithms, the system can compute optimal routes between any two locations and present them in a user-friendly interface. The application also supports visual representation of the campus map, allowing users to better understand the layout and follow directions intuitively.

This project demonstrates the practical application of data structures, algorithms, and user interface design in solving real-world problems. It bridges the gap between theoretical learning and practical usability, offering a scalable solution that can be adapted for different institutions. Beyond its academic value, the system contributes to improving campus accessibility, reducing confusion, and enhancing the overall efficiency of navigation for students, visitors, and staff alike.

OVERVIEW OF THE PROJECT (BLOCK DIAGRAM)



The block diagram of the Campus Route Finder project illustrates the flow of information through five interconnected modules:

At the top, the **User Interface** built with JavaFX provides the front-end interaction where users can select their source and destination using dropdown menus, trigger route calculations through buttons, and view the campus map and paths on a canvas or pane.

These inputs are captured and processed by the **Input Module**, which validates the user's selections and passes them to the core system for computation.

The validated data is then mapped onto the **Campus Map Data**, which is modeled as a graph structure. In this graph, nodes represent physical entities such as hostels, libraries, administrative blocks, and road junctions, while edges represent the connecting paths annotated with distance, travel time, and accessibility information. This structured representation enables the system to analyze the campus layout in a computationally efficient manner.

The **Pathfinding Algorithm** forms the computational core of the project. Algorithms such as Dijkstra's or Breadth-First Search (BFS) are applied to the graph to determine the shortest or most optimal route between the chosen source and destination. The algorithm considers both distance and time, ensuring that the route is practical and efficient.

Finally, the **Route Output Module** presents the results back to the user. It displays the ordered sequence of locations to be traversed, the total distance, and the estimated travel time. Additionally, it provides clear step-by-step directions such as left, right, or straight, making navigation intuitive and user-friendly. Together, these modules transform a static campus sketch into a dynamic digital navigation system that enhances accessibility and usability for students, visitors, and staff.

TOOLS USED

The development of the Campus Route Finder project required a combination of programming frameworks, libraries, and supporting tools to ensure both functionality and usability. The following tools were primarily used:

1. Java Programming Language:

- Core language for implementing the project.
- Provides object-oriented features for modeling campus locations, paths, and algorithms.
- Ensures portability and scalability across platforms.

2. JavaFX Framework:

- Used for building the graphical user interface (GUI).
- Provides components such as dropdown menus, buttons, and canvas/pane for map visualization.
- Enables interactive features like selecting source and destination, displaying routes, and rendering directions.

3. Integrated Development Environment (IDE):

- Tools such as Visual Studio Code was used for coding, debugging, and project management.
- Offers features like syntax highlighting, error detection, and version control integration.

4. Graph Data Structures and Algorithms:

- Implemented using Java collections and custom classes.
- Dijkstra’s Algorithm and Breadth-First Search (BFS) were applied to compute shortest paths.
- Ensures efficient route calculation based on distance and time.

5. Google Maps (for Reference):

- Utilized to measure approximate distances between campus locations.
- Assisted in assigning realistic values for edges in the graph model.

OOPS CONCEPTS USED & ITS EXPLANATION

The Campus Route Finder project is developed using Object-Oriented Programming (OOP) concepts. OOP helps in organizing the program in a structured way and makes the code easy to understand and maintain. The following OOP concepts are used in this project.

Classes_and_Objects:

The project models real-world entities such as hostels, libraries, administrative blocks, and road junctions as objects. Each of these is represented by a class with attributes (like name, coordinates, and type) and behaviors (such as calculating distance or retrieving neighbors). This object-oriented representation makes the system intuitive and closely aligned with the actual campus layout.

Encapsulation:

Encapsulation ensures that the internal details of each class are hidden from the user. For example, the data about distances or coordinates is stored privately within objects, and only specific methods are provided to access or modify them. This prevents accidental misuse of data and maintains consistency across the system.

Abstraction:

Abstraction is applied by exposing only the essential features of the system to the user while hiding the complex implementation details. For instance, when a user requests the shortest route, they only interact with a simple function call, while the underlying algorithm (such as Dijkstra's or BFS) operates invisibly in the background. This makes the system easier to use and understand.

Inheritance:

Inheritance allows the project to reuse and extend existing functionality. For example, different types of locations (such as buildings, hostels, or junctions) can inherit from a common base class "Location." This avoids duplication of code and ensures that all location types share common properties while still allowing specialized behavior.

Polymorphism:

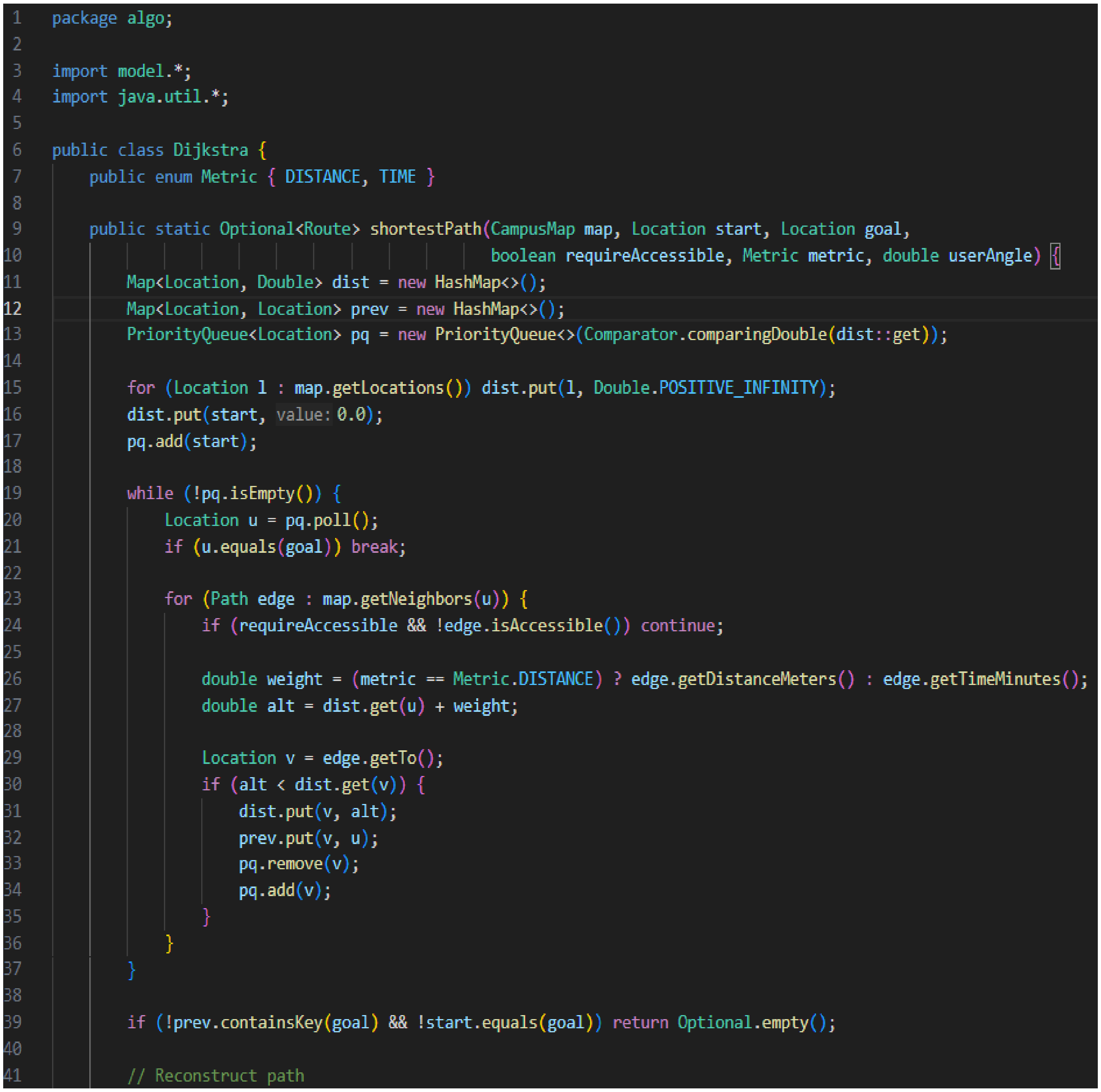
Polymorphism enables the system to handle multiple forms of behavior through a common interface. In this project, different pathfinding strategies (shortest distance, fastest time, or accessible route) can be implemented separately but used interchangeably. The algorithm module can call the same method regardless of which strategy is chosen, making the system flexible and extensible.

Modularity_and_Reusability:

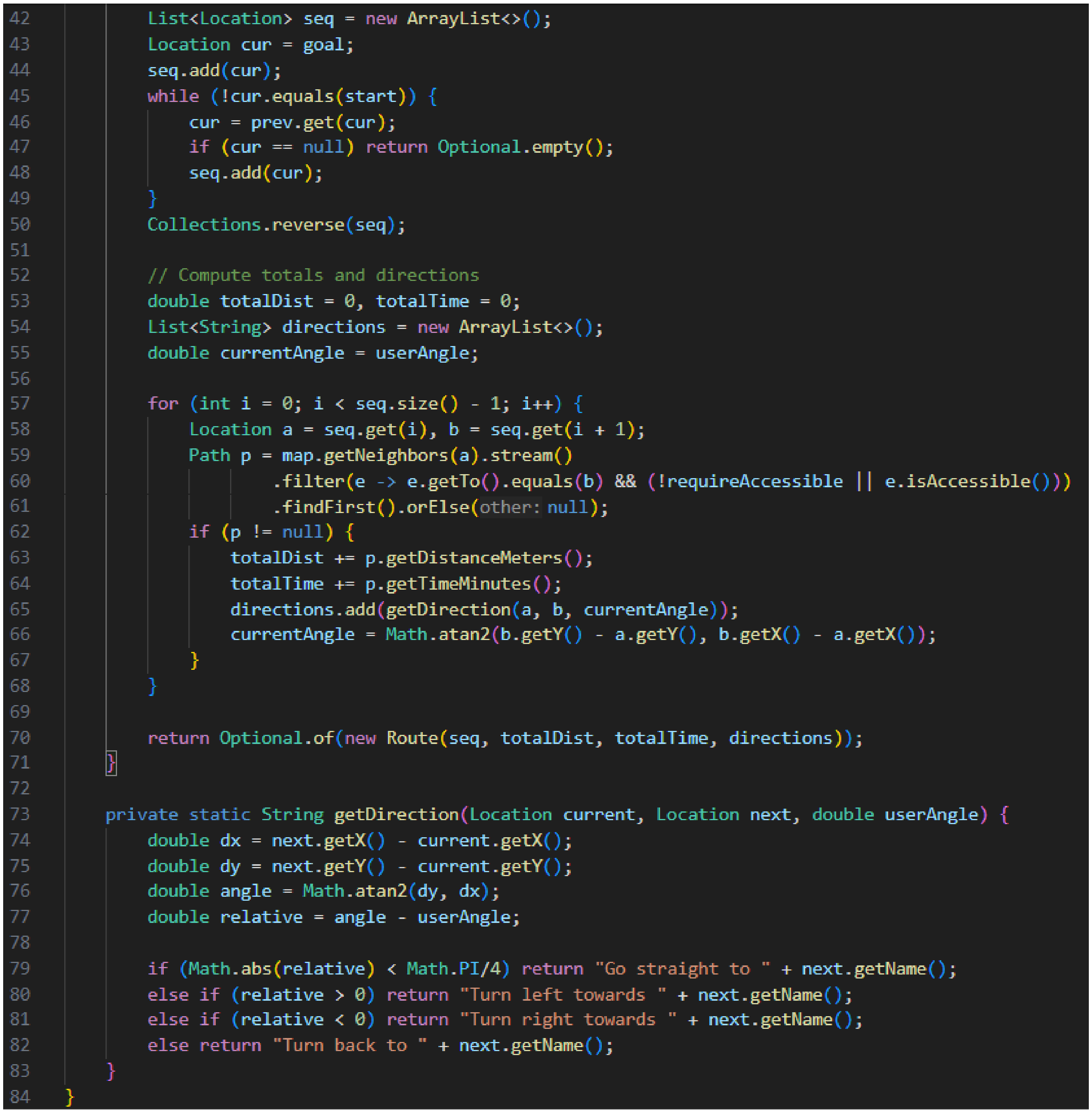
By organizing the project into distinct classes and modules, each with a single responsibility, the system becomes easier to maintain and extend. For example, the user interface, graph data, and pathfinding algorithm are separate modules that can be modified independently without affecting the others.

IMPLEMENTATION / CODE

CAMPUSNAVIGATOR/src/algo/Dijkstra.java

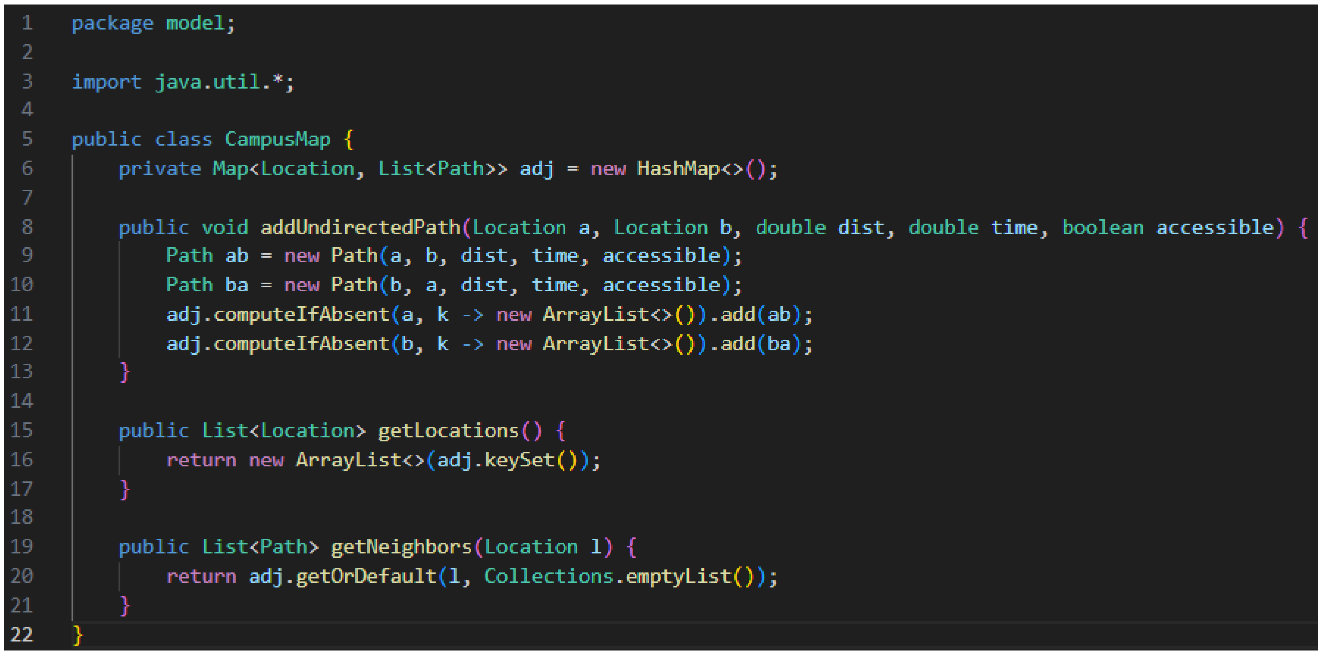


```java
package algo;

import model.*;
import java.util.*;

public class Dijkstra {
    public enum Metric { DISTANCE, TIME }

    public static Optional<Route> shortestPath(CampusMap map, Location start, Location goal,
                                               boolean requireAccessible, Metric metric, double userAngle) {
        Map<Location, Double> dist = new HashMap<>();
        Map<Location, Location> prev = new HashMap<>();
        PriorityQueue<Location> pq = new PriorityQueue<>(Comparator.comparingDouble(dist::get));

        for (Location l : map.getLocations()) dist.put(l, Double.POSITIVE_INFINITY);
        dist.put(start, value: 0.0);
        pq.add(start);

        while (!pq.isEmpty()) {
            Location u = pq.poll();
            if (u.equals(goal)) break;

            for (Path edge : map.getNeighbors(u)) {
                if (requireAccessible && !edge.isAccessible()) continue;

                double weight = (metric == Metric.DISTANCE) ? edge.getDistanceMeters() : edge.getTimeMinutes();
                double alt = dist.get(u) + weight;

                Location v = edge.getTo();
                if (alt < dist.get(v)) {
                    dist.put(v, alt);
                    prev.put(v, u);
                    pq.remove(v);
                    pq.add(v);
                }
            }
        }

        if (!prev.containsKey(goal) && !start.equals(goal)) return Optional.empty();

        // Reconstruct path
```



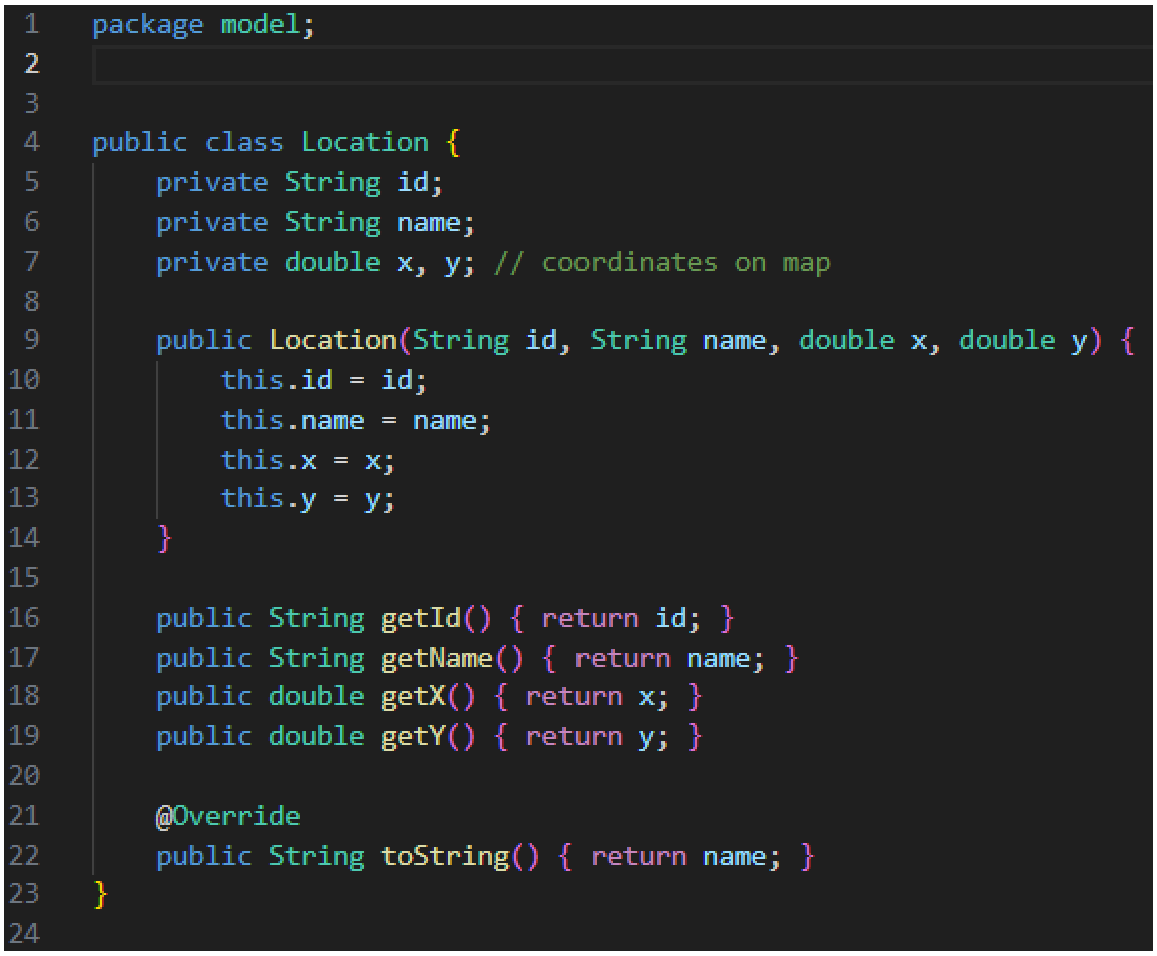
```java
        List<Location> seq = new ArrayList<>();
        Location cur = goal;
        seq.add(cur);
        while (!cur.equals(start)) {
            cur = prev.get(cur);
            if (cur == null) return Optional.empty();
            seq.add(cur);
        }
        Collections.reverse(seq);

        // Compute totals and directions
        double totalDist = 0, totalTime = 0;
        List<String> directions = new ArrayList<>();
        double currentAngle = userAngle;

        for (int i = 0; i < seq.size() - 1; i++) {
            Location a = seq.get(i), b = seq.get(i + 1);
            Path p = map.getNeighbors(a).stream()
                    .filter(e -> e.getTo().equals(b) && (!requireAccessible || e.isAccessible()))
                    .findFirst().orElse(other: null);
            if (p != null) {
                totalDist += p.getDistanceMeters();
                totalTime += p.getTimeMinutes();
                directions.add(getDirection(a, b, currentAngle));
                currentAngle = Math.atan2(b.getY() - a.getY(), b.getX() - a.getX());
            }
        }

        return Optional.of(new Route(seq, totalDist, totalTime, directions));
    }

    private static String getDirection(Location current, Location next, double userAngle) {
        double dx = next.getX() - current.getX();
        double dy = next.getY() - current.getY();
        double angle = Math.atan2(dy, dx);
        double relative = angle - userAngle;

        if (Math.abs(relative) < Math.PI/4) return "Go straight to " + next.getName();
        else if (relative > 0) return "Turn left towards " + next.getName();
        else if (relative < 0) return "Turn right towards " + next.getName();
        else return "Turn back to " + next.getName();
    }
}
```

CAMPUSNAVIGATOR/src/model/CampusMap.java



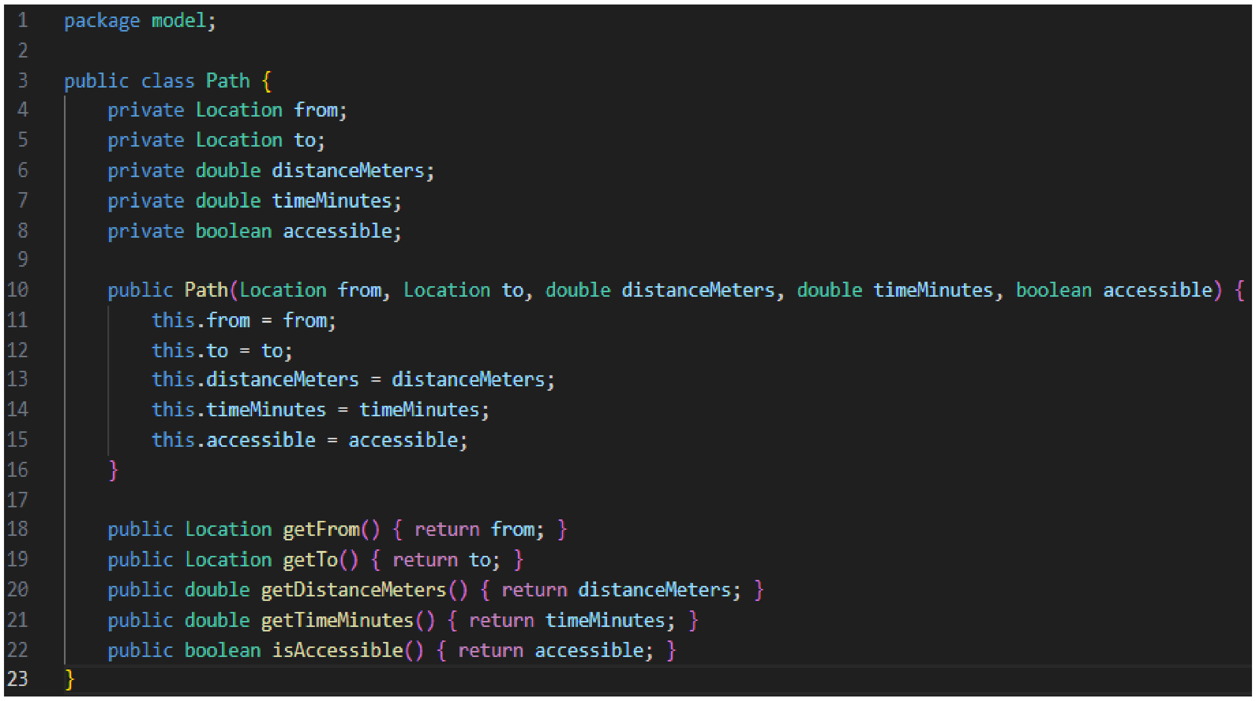
```java
package model;

import java.util.*;

public class CampusMap {
    private Map<Location, List<Path>> adj = new HashMap<>();

    public void addUndirectedPath(Location a, Location b, double dist, double time, boolean accessible) {
        Path ab = new Path(a, b, dist, time, accessible);
        Path ba = new Path(b, a, dist, time, accessible);
        adj.computeIfAbsent(a, k -> new ArrayList<>()).add(ab);
        adj.computeIfAbsent(b, k -> new ArrayList<>()).add(ba);
    }

    public List<Location> getLocations() {
        return new ArrayList<>(adj.keySet());
    }

    public List<Path> getNeighbors(Location l) {
        return adj.getOrDefault(l, Collections.emptyList());
    }
}
```

CAMPUSNAVIGATOR/src/model/Location.java



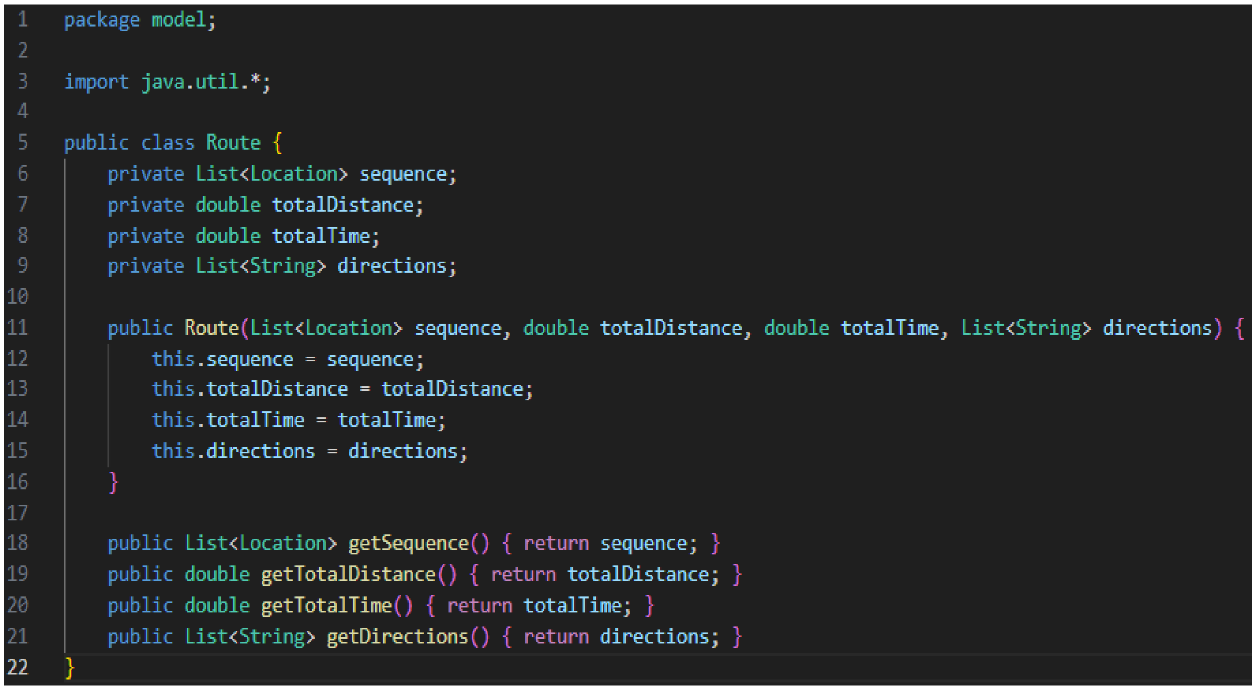
```java
package model;


public class Location {
    private String id;
    private String name;
    private double x, y; // coordinates on map

    public Location(String id, String name, double x, double y) {
        this.id = id;
        this.name = name;
        this.x = x;
        this.y = y;
    }

    public String getId() { return id; }
    public String getName() { return name; }
    public double getX() { return x; }
    public double getY() { return y; }

    @Override
    public String toString() { return name; }
}

```

CAMPUSNAVIGATOR/src/model/Path.java



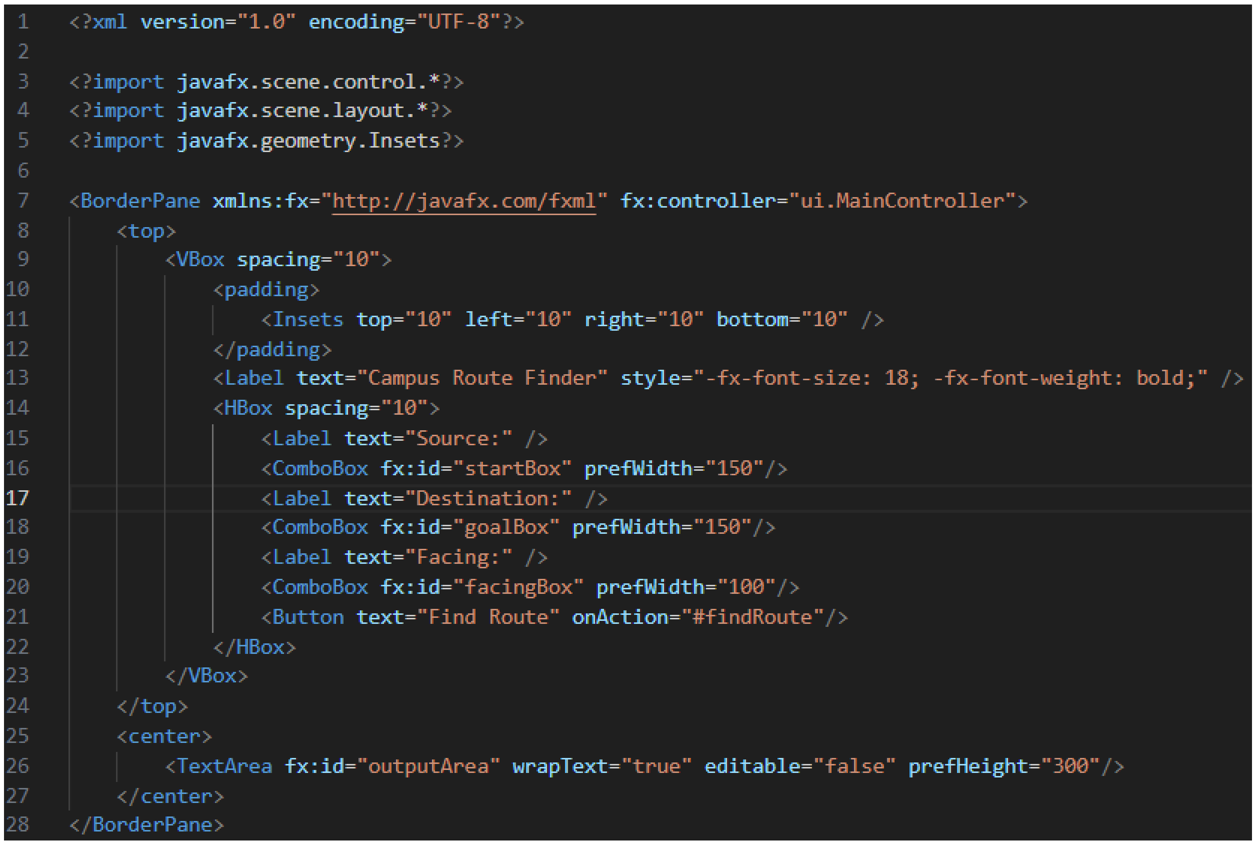
```java
package model;

public class Path {
    private Location from;
    private Location to;
    private double distanceMeters;
    private double timeMinutes;
    private boolean accessible;

    public Path(Location from, Location to, double distanceMeters, double timeMinutes, boolean accessible) {
        this.from = from;
        this.to = to;
        this.distanceMeters = distanceMeters;
        this.timeMinutes = timeMinutes;
        this.accessible = accessible;
    }

    public Location getFrom() { return from; }
    public Location getTo() { return to; }
    public double getDistanceMeters() { return distanceMeters; }
    public double getTimeMinutes() { return timeMinutes; }
    public boolean isAccessible() { return accessible; }
}
```

CAMPUSNAVIGATOR/src/model/Route.java



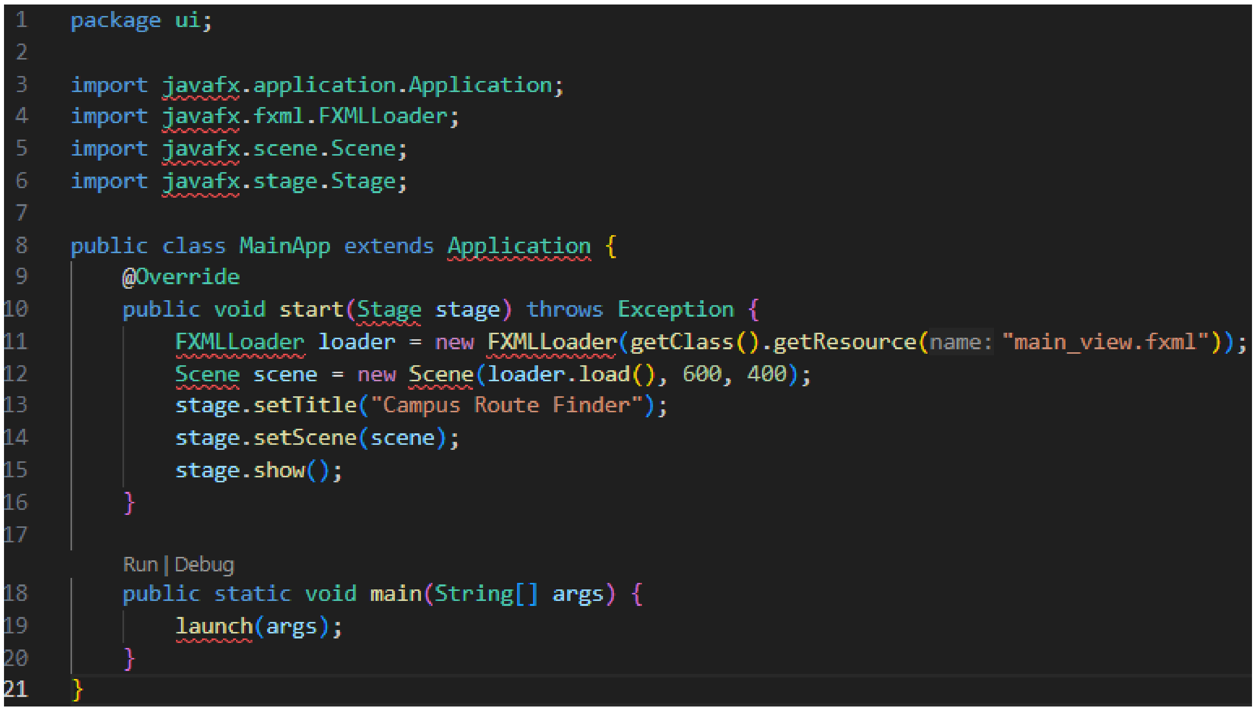
```java
package model;

import java.util.*;

public class Route {
    private List<Location> sequence;
    private double totalDistance;
    private double totalTime;
    private List<String> directions;

    public Route(List<Location> sequence, double totalDistance, double totalTime, List<String> directions) {
        this.sequence = sequence;
        this.totalDistance = totalDistance;
        this.totalTime = totalTime;
        this.directions = directions;
    }

    public List<Location> getSequence() { return sequence; }
    public double getTotalDistance() { return totalDistance; }
    public double getTotalTime() { return totalTime; }
    public List<String> getDirections() { return directions; }
}
```

CAMPUSNAVIGATOR/src/ui/main_view.fxml



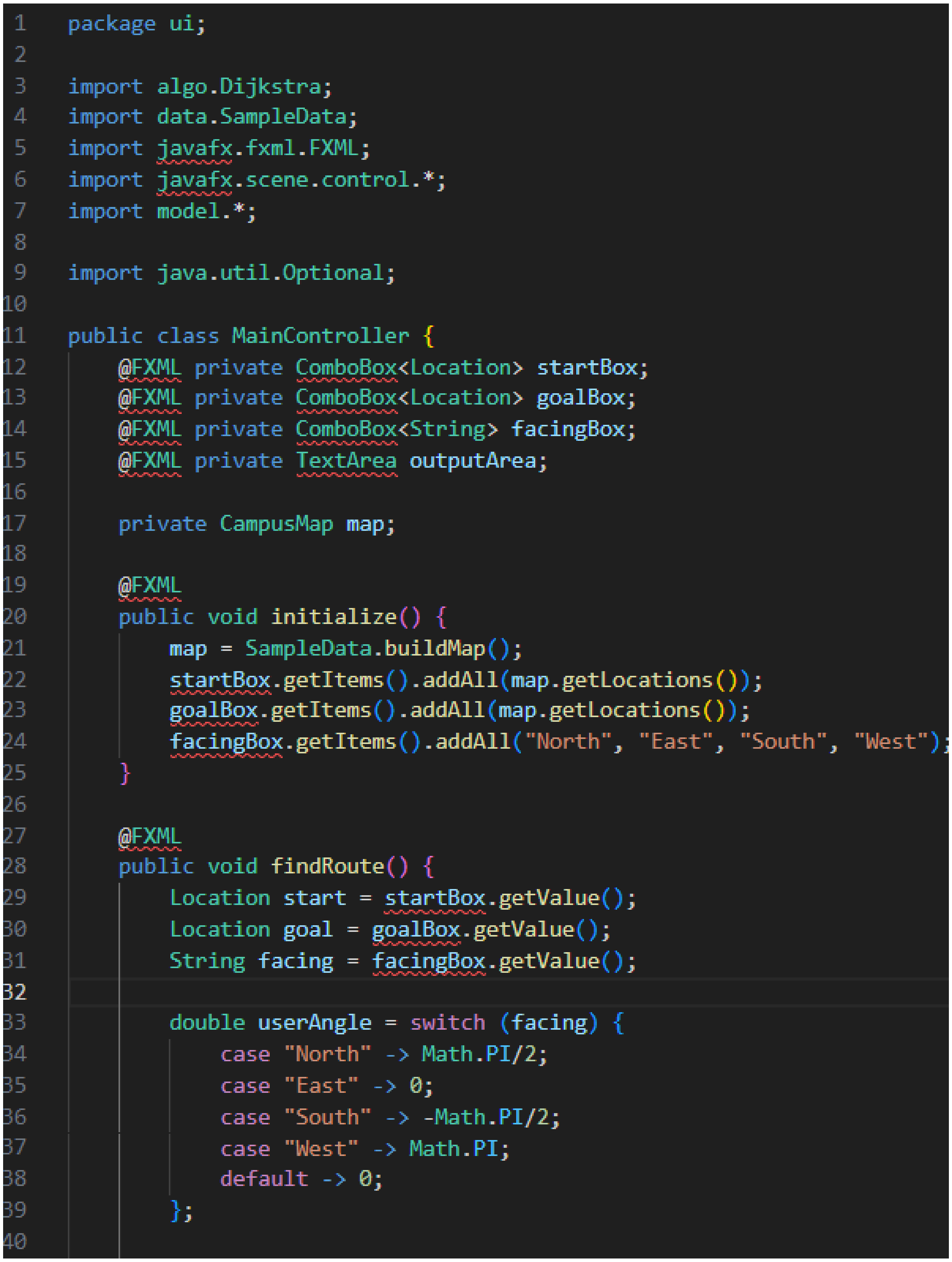
```xml
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.geometry.Insets?>

<BorderPane xmlns:fx="http://javafx.com/fxml" fx:controller="ui.MainController">
    <top>
        <VBox spacing="10">
            <padding>
                <Insets top="10" left="10" right="10" bottom="10" />
            </padding>
            <Label text="Campus Route Finder" style="-fx-font-size: 18; -fx-font-weight: bold;" />
            <HBox spacing="10">
                <Label text="Source:" />
                <ComboBox fx:id="startBox" prefWidth="150"/>
                <Label text="Destination:" />
                <ComboBox fx:id="goalBox" prefWidth="150"/>
                <Label text="Facing:" />
                <ComboBox fx:id="facingBox" prefWidth="100"/>
                <Button text="Find Route" onAction="#findRoute"/>
            </HBox>
        </VBox>
    </top>
    <center>
        <TextArea fx:id="outputArea" wrapText="true" editable="false" prefHeight="300"/>
    </center>
</BorderPane>
```

CAMPUSNAVIGATOR/src/ui/MainApp.java

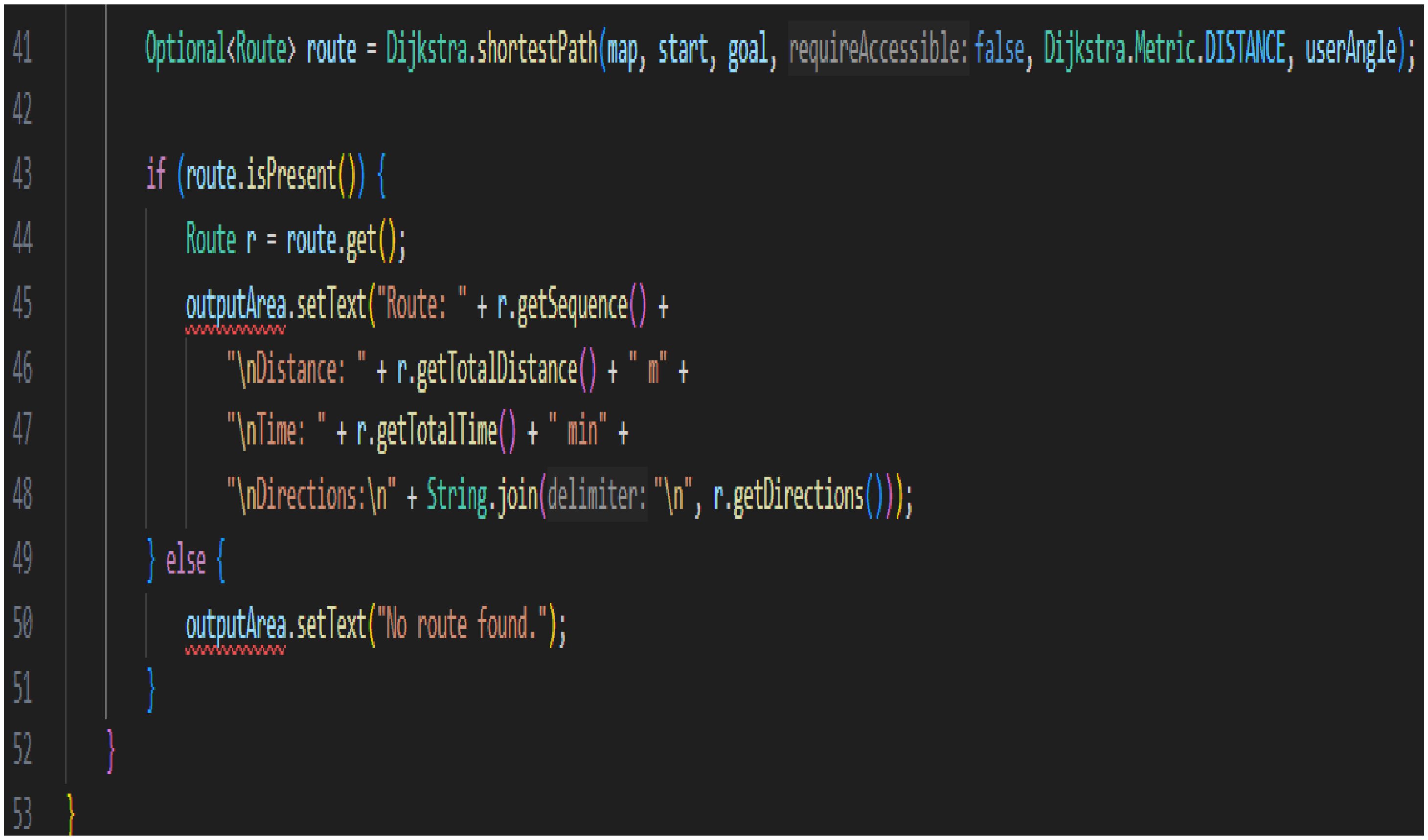


```java
package ui;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class MainApp extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        FXMLLoader loader = new FXMLLoader(getClass().getResource(name: "main_view.fxml"));
        Scene scene = new Scene(loader.load(), 600, 400);
        stage.setTitle("Campus Route Finder");
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

CAMPUSNAVIGATOR/src/ui/MainController.java

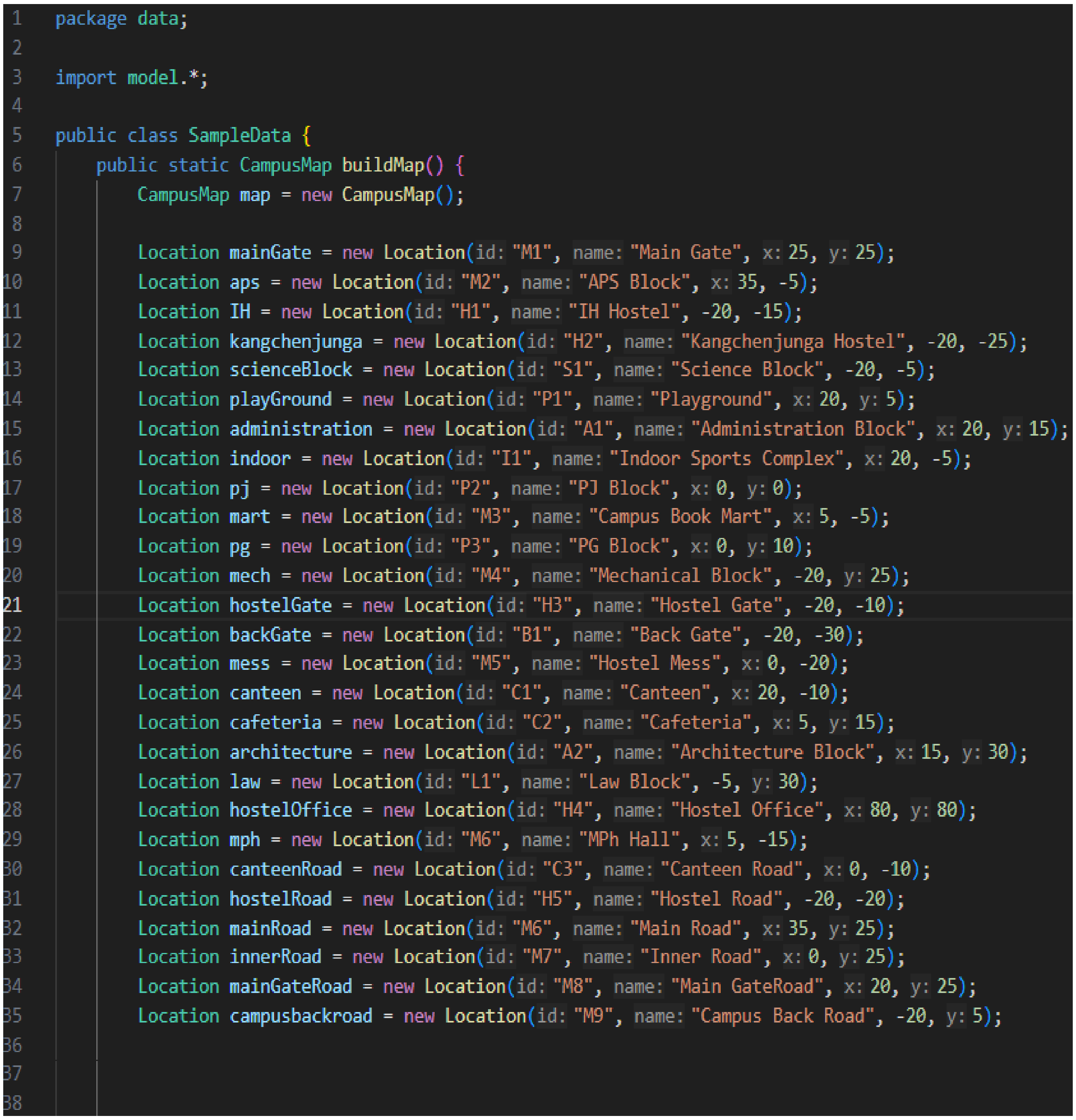


```java
package ui;

import algo.Dijkstra;
import data.SampleData;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import model.*;

import java.util.Optional;

public class MainController {
    @FXML private ComboBox<Location> startBox;
    @FXML private ComboBox<Location> goalBox;
    @FXML private ComboBox<String> facingBox;
    @FXML private TextArea outputArea;

    private CampusMap map;

    @FXML
    public void initialize() {
        map = SampleData.buildMap();
        startBox.getItems().addAll(map.getLocations());
        goalBox.getItems().addAll(map.getLocations());
        facingBox.getItems().addAll("North", "East", "South", "West");
    }

    @FXML
    public void findRoute() {
        Location start = startBox.getValue();
        Location goal = goalBox.getValue();
        String facing = facingBox.getValue();

        double userAngle = switch (facing) {
            case "North" -> Math.PI/2;
            case "East" -> 0;
            case "South" -> -Math.PI/2;
            case "West" -> Math.PI;
            default -> 0;
        };

```

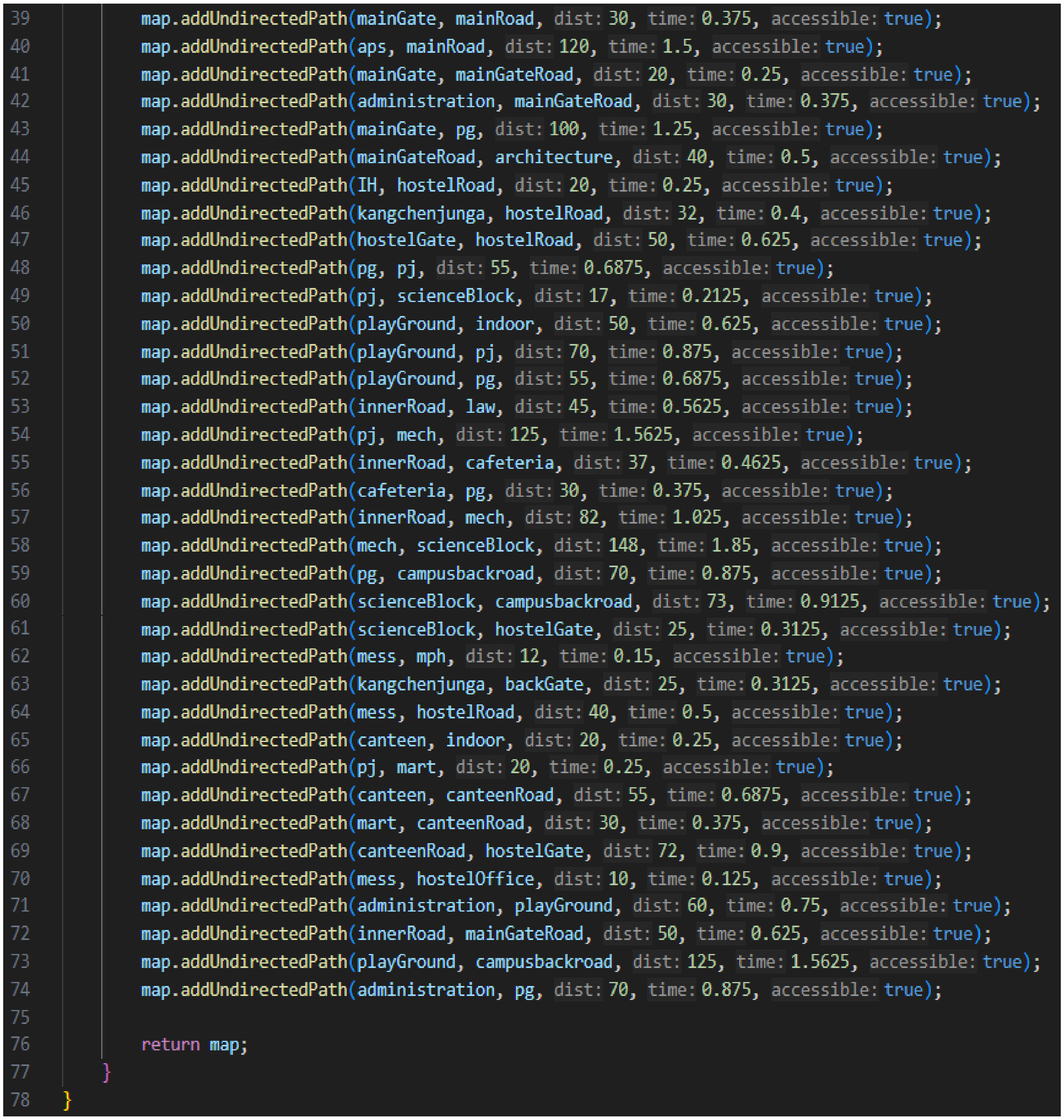


```
        Optional<Route> route = Dijkstra.shortestPath(map, start, goal, requireAccessible: false, Dijkstra.Metric.DISTANCE, userAngle);

        if (route.isPresent()) {
            Route r = route.get();
            outputArea.setText("Route: " + r.getSequence() +
                "\nDistance: " + r.getTotalDistance() + " m" +
                "\nTime: " + r.getTotalTime() + " min" +
                "\nDirections:\n" + String.join(delimiter: "\n", r.getDirections()));
        } else {
            outputArea.setText("No route found.");
        }
    }
}
```

CAMPUSNAVIGATOR/src/data/SampleData.java

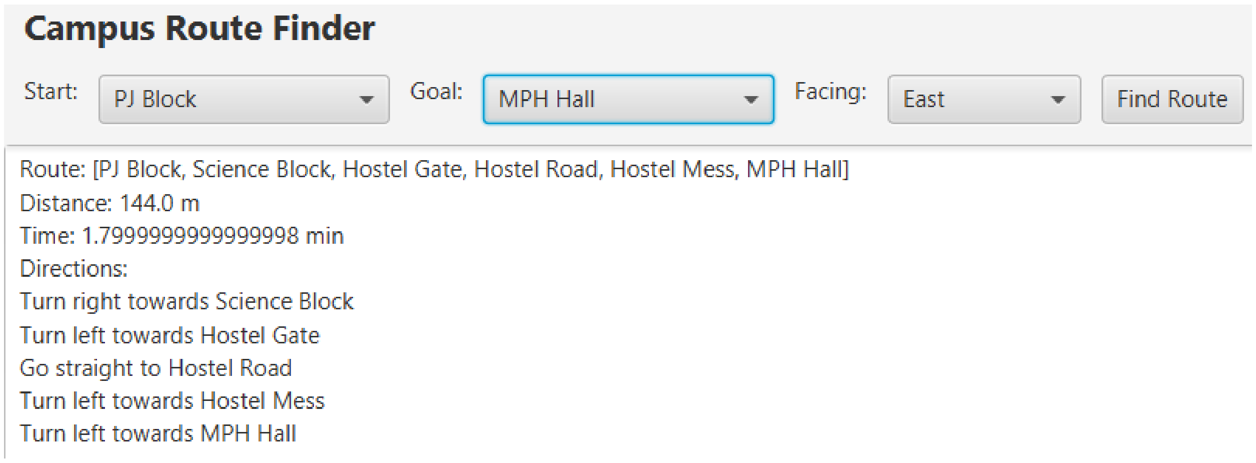


```java
package data;

import model.*;

public class SampleData {
    public static CampusMap buildMap() {
        CampusMap map = new CampusMap();

        Location mainGate = new Location(id: "M1", name: "Main Gate", x: 25, y: 25);
        Location aps = new Location(id: "M2", name: "APS Block", x: 35, -5);
        Location IH = new Location(id: "H1", name: "IH Hostel", -20, -15);
        Location kangchenjunga = new Location(id: "H2", name: "Kangchenjunga Hostel", -20, -25);
        Location scienceBlock = new Location(id: "S1", name: "Science Block", -20, -5);
        Location playGround = new Location(id: "P1", name: "Playground", x: 20, y: 5);
        Location administration = new Location(id: "A1", name: "Administration Block", x: 20, y: 15);
        Location indoor = new Location(id: "I1", name: "Indoor Sports Complex", x: 20, -5);
        Location pj = new Location(id: "P2", name: "PJ Block", x: 0, y: 0);
        Location mart = new Location(id: "M3", name: "Campus Book Mart", x: 5, -5);
        Location pg = new Location(id: "P3", name: "PG Block", x: 0, y: 10);
        Location mech = new Location(id: "M4", name: "Mechanical Block", -20, y: 25);
        Location hostelGate = new Location(id: "H3", name: "Hostel Gate", -20, -10);
        Location backGate = new Location(id: "B1", name: "Back Gate", -20, -30);
        Location mess = new Location(id: "M5", name: "Hostel Mess", x: 0, -20);
        Location canteen = new Location(id: "C1", name: "Canteen", x: 20, -10);
        Location cafeteria = new Location(id: "C2", name: "Cafeteria", x: 5, y: 15);
        Location architecture = new Location(id: "A2", name: "Architecture Block", x: 15, y: 30);
        Location law = new Location(id: "L1", name: "Law Block", -5, y: 30);
        Location hostelOffice = new Location(id: "H4", name: "Hostel Office", x: 80, y: 80);
        Location mph = new Location(id: "M6", name: "MPh Hall", x: 5, -15);
        Location canteenRoad = new Location(id: "C3", name: "Canteen Road", x: 0, -10);
        Location hostelRoad = new Location(id: "H5", name: "Hostel Road", -20, -20);
        Location mainRoad = new Location(id: "M6", name: "Main Road", x: 35, y: 25);
        Location innerRoad = new Location(id: "M7", name: "Inner Road", x: 0, y: 25);
        Location mainGateRoad = new Location(id: "M8", name: "Main GateRoad", x: 20, y: 25);
        Location campusbackroad = new Location(id: "M9", name: "Campus Back Road", -20, y: 5);


```

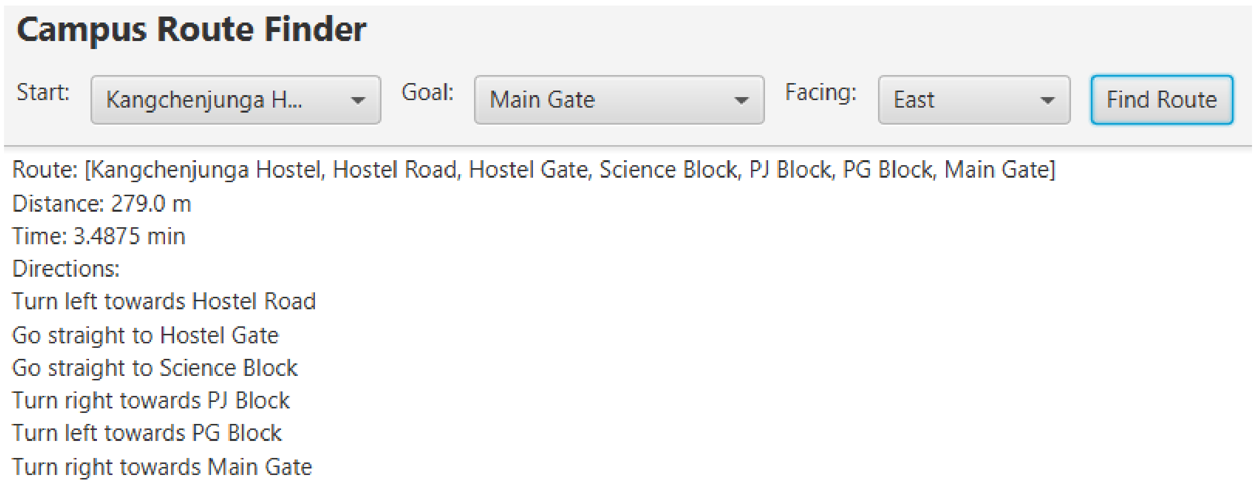


```java
        map.addUndirectedPath(mainGate, mainRoad, dist: 30, time: 0.375, accessible: true);
        map.addUndirectedPath(aps, mainRoad, dist: 120, time: 1.5, accessible: true);
        map.addUndirectedPath(mainGate, mainGateRoad, dist: 20, time: 0.25, accessible: true);
        map.addUndirectedPath(administration, mainGateRoad, dist: 30, time: 0.375, accessible: true);
        map.addUndirectedPath(mainGate, pg, dist: 100, time: 1.25, accessible: true);
        map.addUndirectedPath(mainGateRoad, architecture, dist: 40, time: 0.5, accessible: true);
        map.addUndirectedPath(IH, hostelRoad, dist: 20, time: 0.25, accessible: true);
        map.addUndirectedPath(kangchenjunga, hostelRoad, dist: 32, time: 0.4, accessible: true);
        map.addUndirectedPath(hostelGate, hostelRoad, dist: 50, time: 0.625, accessible: true);
        map.addUndirectedPath(pg, pj, dist: 55, time: 0.6875, accessible: true);
        map.addUndirectedPath(pj, scienceBlock, dist: 17, time: 0.2125, accessible: true);
        map.addUndirectedPath(playGround, indoor, dist: 50, time: 0.625, accessible: true);
        map.addUndirectedPath(playGround, pj, dist: 70, time: 0.875, accessible: true);
        map.addUndirectedPath(playGround, pg, dist: 55, time: 0.6875, accessible: true);
        map.addUndirectedPath(innerRoad, law, dist: 45, time: 0.5625, accessible: true);
        map.addUndirectedPath(pj, mech, dist: 125, time: 1.5625, accessible: true);
        map.addUndirectedPath(innerRoad, cafeteria, dist: 37, time: 0.4625, accessible: true);
        map.addUndirectedPath(cafeteria, pg, dist: 30, time: 0.375, accessible: true);
        map.addUndirectedPath(innerRoad, mech, dist: 82, time: 1.025, accessible: true);
        map.addUndirectedPath(mech, scienceBlock, dist: 148, time: 1.85, accessible: true);
        map.addUndirectedPath(pg, campusbackroad, dist: 70, time: 0.875, accessible: true);
        map.addUndirectedPath(scienceBlock, campusbackroad, dist: 73, time: 0.9125, accessible: true);
        map.addUndirectedPath(scienceBlock, hostelGate, dist: 25, time: 0.3125, accessible: true);
        map.addUndirectedPath(mess, mph, dist: 12, time: 0.15, accessible: true);
        map.addUndirectedPath(kangchenjunga, backGate, dist: 25, time: 0.3125, accessible: true);
        map.addUndirectedPath(mess, hostelRoad, dist: 40, time: 0.5, accessible: true);
        map.addUndirectedPath(canteen, indoor, dist: 20, time: 0.25, accessible: true);
        map.addUndirectedPath(pj, mart, dist: 20, time: 0.25, accessible: true);
        map.addUndirectedPath(canteen, canteenRoad, dist: 55, time: 0.6875, accessible: true);
        map.addUndirectedPath(mart, canteenRoad, dist: 30, time: 0.375, accessible: true);
        map.addUndirectedPath(canteenRoad, hostelGate, dist: 72, time: 0.9, accessible: true);
        map.addUndirectedPath(mess, hostelOffice, dist: 10, time: 0.125, accessible: true);
        map.addUndirectedPath(administration, playGround, dist: 60, time: 0.75, accessible: true);
        map.addUndirectedPath(innerRoad, mainGateRoad, dist: 50, time: 0.625, accessible: true);
        map.addUndirectedPath(playGround, campusbackroad, dist: 125, time: 1.5625, accessible: true);
        map.addUndirectedPath(administration, pg, dist: 70, time: 0.875, accessible: true);

        return map;
    }
}
```

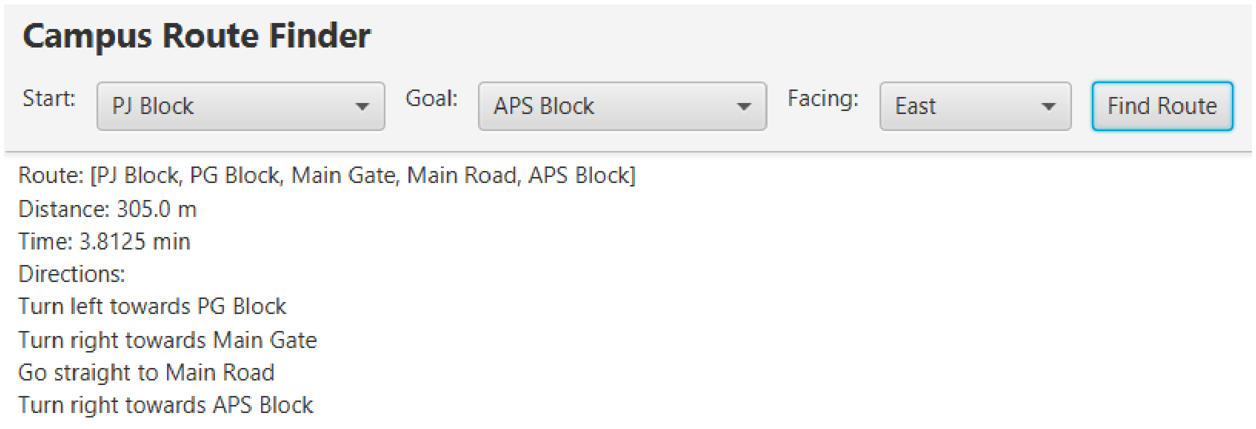
RESULT / SNAPSHOTS



```
Route: [PJ Block, Science Block, Hostel Gate, Hostel Road, Hostel Mess, MPH Hall]
Distance: 144.0 m
Time: 1.7999999999999998 min
Directions:
Turn right towards Science Block
Turn left towards Hostel Gate
Go straight to Hostel Road
Turn left towards Hostel Mess
Turn left towards MPH Hall
```



```
Route: [Kangchenjunga Hostel, Hostel Road, Hostel Gate, Science Block, PJ Block, PG Block, Main Gate]
Distance: 279.0 m
Time: 3.4875 min
Directions:
Turn left towards Hostel Road
Go straight to Hostel Gate
Go straight to Science Block
Turn right towards PJ Block
Turn left towards PG Block
Turn right towards Main Gate
```



```
Route: [PJ Block, PG Block, Main Gate, Main Road, APS Block]
Distance: 305.0 m
Time: 3.8125 min
Directions:
Turn left towards PG Block
Turn right towards Main Gate
Go straight to Main Road
Turn right towards APS Block
```

Future Developments:

The Campus Route Finder project provides a functional solution for basic navigation across the campus. However, several enhancements can be introduced to make the system more advanced, user-friendly, and scalable:

1. Real-Time Location Tracking:

- Integrating GPS or Wi-Fi positioning to track a user's live location on the campus map.
- Enables dynamic route guidance that updates as the user moves.

2. Mobile Application Integration:

- Developing Android/iOS versions of the route finder for easy access on smartphones.
- Provides portability and convenience for students and visitors.

3. Voice-Based Assistance:

- Adding voice input and output so users can ask for directions verbally and receive spoken guidance.
- Improves accessibility for visually impaired users.

4. Multi-Criteria Routing:

- Allowing users to choose routes based on shortest distance, fastest time, or accessibility (wheelchair-friendly paths).
- Adds personalization and inclusivity.

5. Event-Based Navigation:

- Highlighting temporary routes during campus events, festivals, or construction work.
- Keeps navigation up-to-date with real-world changes.

6. Integration with Campus Services:

- Linking the route finder with facilities such as libraries, cafeterias, hostels, and labs.
- Users can search for services (e.g., “nearest cafeteria”) instead of just locations.

7. Augmented Reality (AR) Guidance:

- Using AR overlays on mobile devices to show arrows and directions directly on the camera view.
- Provides an immersive navigation experience.

8. Scalability to Other Institutions:

- Adapting the system for different campuses, universities, or even large public spaces like hospitals or malls.
- Expands the project’s usability beyond a single institution.

9. Cloud-Based Data Storage:

- Hosting campus maps and route data on the cloud for easier updates and collaboration.
- Ensures that changes in campus layout are reflected instantly.

10. Analytics and Feedback System:

- Collecting user feedback and analyzing frequently used routes.
- Helps administrators improve campus infrastructure and optimize pathways.

REFERENCES

1. YouTube Tutorials on Java Programming and JavaFX.

2. Class Notes and Reference Materials provided by the Instructor.

3. Google Maps to measure campus distances.

4. GeeksforGeeks. Dijkstra's Shortest Path Algorithm. Retrieved from https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-priority_queue-stl/

5. Personal Practice and Self-Learning Materials.

6. BMS College of Engineering. Campus Guide and Map. 2025.