SYSTEM DOCUMENTATION: UG NAVIGATE

1. Executive Summary

This document provides a comprehensive overview of the system architecture and software components for the UG Navigate application. UG Navigate is a standalone desktop application developed using the JavaFX framework, designed to provide an efficient and user-friendly route-finding solution for the University of Ghana campus.

The application allows users to find the most effective routes between two points based on criteria such as shortest distance and optimal travel time. It also supports advanced queries, such as finding routes that pass through specified landmarks. This guide will detail the function of each file within the project, explaining its specific role in the overall operation of the application. The intended audience for this document includes developers, project managers, and any stakeholders interested in the technical design of the application.

2. System Architecture

The UG Navigate application is designed using a well-structured architecture that separates different aspects of the program. This makes the system easier to manage, update, and debug. The design is similar to the classic **Model-View-Controller (MVC)** pattern, which can be understood as follows:

- ***The Model (The Brains and the Library):*** This is the core of the application. It manages all the data (the campus map) and contains all the business logic (how to find routes, calculate distances, and sort results). It is the "thinking" part of the application and has no knowledge of how the user interface looks.
- ***The View (The User's Dashboard):*** This is everything the user sees and interacts with. It includes all the buttons, dropdown menus, and the map display. Its job is to present information to the user and capture their input. The View is defined by the FXML and CSS files.
- ***The Controller (The Conductor):*** This acts as the bridge between the Model and the View. When a user clicks a button in the View, the Controller receives that action and tells the Model what to do. Once the Model has a result, the Controller takes that information and updates the View to show it to the user.

This separation ensures that the application is robust, scalable, and well-organized.

3. Component Descriptions: The Code Files (dcit204.map)

This section describes the function of each Java file, which collectively form the application's engine.

Core Logic Components (The Model)

- DataLoader.java: **The Cartographer.** This component is responsible for loading the digital map of the campus. It reads the raw data from data.json, which contains all locations and their connections, and builds the in-memory graph structure that the rest of the application uses to navigate.
- RouteFinder.java: **The Master Navigator.** This is a highly specialized component that contains the primary route-finding intelligence. Using powerful pathfinding algorithms (like Dijkstra's), it searches the campus map graph to discover valid paths from a user's starting point to their destination.
- DistanceCalculator.java: **The Surveyor.** A focused and essential utility. Its sole purpose is to calculate the physical distance between any two points on the map, as requested by other components like the RouteFinder.
- SearchAndLandmarks.java: **The Tour Planner.** This component handles more complex user requests. When a user wishes to travel via specific landmarks (e.g., "pass

by the Hall"), this component works with the RouteFinder to construct and validate routes that meet these specific criteria.

- SortingAlgorithms.java: **The Prioritizer.** After multiple routes have been found, this component steps in to organize them. It uses efficient sorting algorithms (Quick Sort, Merge Sort) to arrange the routes in a meaningful order—either by shortest distance or by optimal time—so the user is presented with the best options first.
- TrafficSimulator.java: **The Real-World Analyst.** This component adds a layer of intelligence by simulating real-world conditions. It can adjust the estimated travel time for routes based on factors like the time of day, making the "optimal time" calculation more realistic.
- RouteOption.java: **The Route Data Card.** This is not a worker but a blueprint. It defines the structure for storing information about a single route. Think of it as an index card that holds all the details for one path: its sequence of locations, its total distance, its calculated time, and any landmarks along the way.

User Interface Components (The View & Controller)

- HelloApplication.java: **The Application Launcher.** This is the official entry point of the application. Its primary responsibility is to initialize the JavaFX framework, load the main user interface from the hello-view.fxml file, apply the necessary styling, and display the main window to the user.
- HelloController.java: **The Central Command.** As the primary Controller, this component is the link between the user interface and the application's core logic. It captures all user interactions (button clicks, dropdown selections) and delegates tasks to the appropriate model components (RouteFinder, SortingAlgorithms, etc.). It is also responsible for receiving the results and updating the user interface to display them.

Resource & Configuration Files

This section describes the non-code files that are essential for the application's appearance, data, and configuration.

- hello-view.fxml: **The UI Blueprint.** An XML-based file that defines the complete structure and layout of the user interface. It specifies which controls are present (buttons, labels, etc.) and where they are positioned, acting as the architectural plan for the application's visual front-end.
- styles.css: **The Visual Style Guide.** This Cascading Style Sheet (CSS) file contains all the styling rules that define the application's visual appearance. It controls colors, fonts, spacing, and other decorative elements to ensure a clean, professional, and consistent look and feel.
- navigation-bar.fxml: **Legacy UI Blueprint.** Similar to the NavigationController, this FXML file defined the layout for the now-removed sidebar menu. It is no longer loaded by the application.
- data.json: **The Campus Map Database.** This is the single source of truth for the application's geographical data. It stores all campus locations, landmarks, and the direct connections (edges) between them, including distance information. This file is read by the DataLoader at startup.
- module-info.java: **The System Dependency Manifest.** This is a configuration file for the Java Platform Module System. It explicitly declares the external JavaFX modules that UG Navigate depends on, such as javafx.controls, javafx.fxml, and javafx.web. This ensures the application is robust and has access to all the necessary libraries to function correctly.