# Smart Route Assistance

*A Web-Based Optimal Path Finding System*

**Project Report**

**Author:**

Muhammad Gulraiz Khan

**Date:**

December 15, 2025

**Department of Cyber Security**
**Air University**

# Contents

# Chapter 1

# Introduction

## 1.1 Project Overview

The **Smart Route Assistance** system is a web-based application designed to solve the classic "Shortest Path Problem" in a user-friendly environment. Moving beyond traditional console-based applications, this project leverages a full-stack web architecture to visualize routes between cities across multiple countries, including Pakistan, India, Turkey, Saudi Arabia, and Palestine.

## 1.2 Objectives

The primary objectives of this project are:

- To implement a robust backend using **Dijkstra's Algorithm** for optimal pathfinding.
- To provide users with options to minimize either **Distance** (km) or **Cost** (currency).
- To design an interactive and aesthetically pleasing User Interface (UI) that visualizes the route.
- To manage dynamic data (Countries, Cities, Routes) using a relational database.

# Chapter 2

# System Architecture

The system follows a standard **Client-Server Architecture** typically deployed via XAMPP (Apache, MySQL, PHP).

**Technology Stack**

- **Frontend:** HTML5, CSS3 (Custom Styling), JavaScript (Fetch API).
- **Backend:** PHP (Logic Processing  API Endpoints).
- **Database:** MySQL (Relational Data Storage).
- **Server:** Apache HTTP Server (via XAMPP).

## 2.1 Data Flow

1. The **Client** selects a country, source, and destination.
2. JavaScript sends an asynchronous request (AJAX) to the **PHP API**.
3. The **API** queries the **MySQL Database** for all relevant nodes and edges.
4. The API runs **Dijkstra's Algorithm** to calculate the shortest path.
5. The result (Path & Total Cost) is returned as JSON and rendered on the screen.

# Chapter 3

# Database Design

The application utilizes a relational database named `smart_route_db`. The schema is designed to support hierarchical data selection (Country → City) and graph-based route calculations.

## 3.1 Entity Relationship Schema

**Countries Table:** Stores the names of supported nations.

**Cities Table:** Stores city names linked to a specific Country ID.

**Routes Table:** Represents the "Edges" of the graph. It links two cities (Source ID and Destination ID) and stores the *Distance* and *Cost* as weights.

## 3.2 SQL Structure

The following SQL commands were used to generate the core structure:

Listing 3.1: Database Schema Creation

```
CREATE TABLE countries (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE
);

CREATE TABLE cities (
    id INT AUTO_INCREMENT PRIMARY KEY,
    country_id INT,
    name VARCHAR(100) NOT NULL,
    FOREIGN KEY (country_id) REFERENCES countries(id)
);

CREATE TABLE routes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    source_city_id INT,
    dest_city_id INT,
    distance INT,
    cost INT,
```

```
    FOREIGN KEY (source_city_id) REFERENCES cities(id),
    FOREIGN KEY (dest_city_id) REFERENCES cities(id)
);
```

# Chapter 4

# Algorithm Implementation

The core intelligence of the system relies on **Dijkstra's Algorithm**. This is a greedy algorithm that finds the shortest path from a starting node to a target node in a weighted graph.

## 4.1 Why Dijkstra?

Unlike simple Breadth-First Search (BFS), Dijkstra accounts for edge weights. In our context:

- **Nodes** = Cities
- **Edges** = Routes
- **Weights** = Distance (km) OR Cost (currency)

## 4.2 Implementation Logic (PHP)

The backend fetches the graph data and processes it using a Priority Queue structure (min-heap).

**Algorithm Steps**

1. Initialize distances to all cities as `Infinity`, except the Source (0).
2. Insert the Source into a Priority Queue.
3. While the Queue is not empty:
   (a) Extract the city with the smallest distance.
   (b) For each neighbor of this city:
   (c) Calculate `new_dist = current_dist + edge_weight`.
   (d) If `new_dist < neighbor_dist`, update the neighbor and add to Queue.
4. Reconstruct the path by backtracking from the Destination to the Source using a `parent` array.

# Chapter 5

# User Interface

The user interface was designed with a "Vintage Explorer" aesthetic. It features a responsive card layout, dynamic dropdowns, and a visual route summary.

## 5.1 Design Features

- **Dynamic Loading:** Dropdowns for cities only populate after a country is selected, preventing user error.
- **Visual Feedback:** The route is drawn using nodes and arrows rather than simple text.
- **Theming:** A vintage map background with a "Parchment" style form container (`#E0D3C3`) and Ocean Blue accents (`#005F99`).

## 5.2 Project Screenshot

Below is the final rendered view of the Smart Route Assistance application.

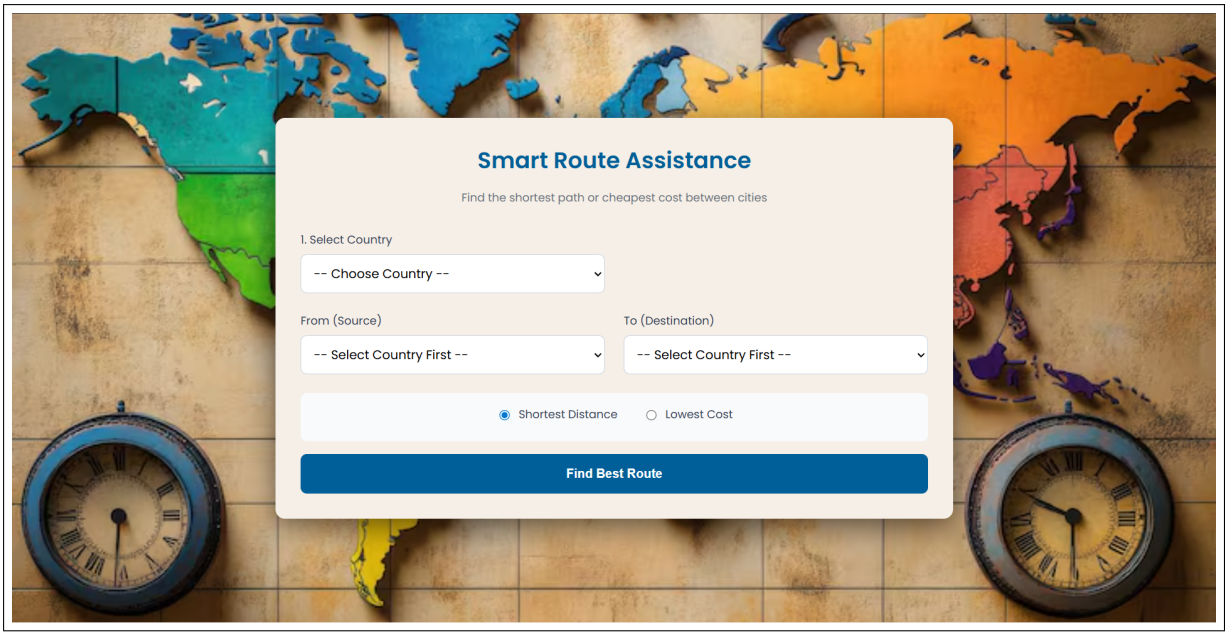


Figure 5.1: Final User Interface showing the Map Background and Search Form

# Chapter 6

# Conclusion

The "Smart Route Assistance" project successfully translates complex data structure concepts into a practical web application. By integrating a MySQL database with a PHP-based implementation of Dijkstra's algorithm, the system provides accurate and efficient routing solutions. The transition from a C++ console app to a GUI-based web interface significantly enhances usability and accessibility.