FASTEST PATH INCLUDING FLIGHTS

By

Jagadeesh Muthamsetty - 201758935

A DISSERTATION

Submitted to

The University of Liverpool

in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

STUDENT DECLARATION

I certify that this dissertation constitutes my product, that where the language of others is set forth, quotation marks so indicate. That appropriate credit is given where I have used the language, ideas, expressions, or writings of another.

I confirm that I have not copied material from another source, committed plagiarism, commissioned all or part of the work (including unacceptable proof-reading), or fabricated, falsified, or embellished data when completing the attached piece of work.

I declare that the dissertation describes original work that has not previously been presented for any other institution's degree award.

Signature: Jagadeesh Muthamsetty

ACKNOWLEDGEMENTS

Thank you as always to everyone who has been a part of bringing this project to life.

To start with, I would like to give my sincerest thanks to my project supervisor, Rasmus Ibsen-Jensen, whose assistance and support have been invaluable in finding the path and importance of this work.

I would also like to thank my mother and friends for their constant support and understanding throughout this project. Thanks for their encouragement and patience, and gave me the energy and superpower at that time of month time.

Huge shoutout to my peers and colleagues who provided feedback, and shared their experiences and perspective during brainstorming sessions. They improved my project and they helped me make it better and refine the scope and the way to implement it.

Finally, I simply want to say thank you to the developers and maintainers of all of the open-source tools and resources that I have used in this project, and the team behind the Google Geocode API for a great service that allowed me to easily implement location-based functionality.

However, I want to express from the bottom of my heart to each person, be directly or indirectly related to the success of this project, thank you very much!

FASTEST PATH INCLUDING FLIGHTS

ABSTRACT

This project describes the designing, implementation, and evaluation of a web application to compute the shortest distance from place A to place B in the United Kingdom using Dijkstra's algorithm. It utilizes Google Geocode API to convert these location names entered by users into geospatial coordinates required for routing with accuracy and in a reliable, simple manner.

The project employs a Node.js backend to handle computational logic and API interactions, while the frontend, built with HTML and CSS, provides users with an intuitive interface for entering locations and viewing results. The system integrates modern web development practices to offer a seamless and responsive user experience.

One of the project's main goals was effectively implementing Dijkstra's algorithm to ensure optimal pathfinding while preserving computational efficiency. To confirm accuracy and performance, the implementation underwent extensive testing using a variety of datasets. Additionally, the project was created with privacy in mind, guaranteeing that no user data is saved or kept throughout the process.

Strict ethical guidelines are followed by the project, guaranteeing that location data is handled responsibly and used only for real-time processing. To sum up, the application effectively illustrates how sophisticated algorithmic techniques can be used in real-world situations, laying the groundwork for future improvements and wider regional expansions.

Statement of Ethical Compliance

This project involves data categories A2. I confirm that I will follow our ethical guidance throughout the development and implementation of this project. I confirm that I have followed all ethical guidelines and standards throughout the development and implementation of this project.

At no point during the project has any sensitive or personal user data been gathered, saved, or kept. Real-time processing is the only way to interact with the Google Geocode API, guaranteeing that no personally identifiable user data is captured or misused.

the project has been designed and executed with a strong commitment to data privacy, security, and ethical considerations.

CONTENTS

Table of pictures

INTRODUCTION

An essential aspect of modern navigation systems is efficient route optimisation, which plays a crucial role in various activities such as daily commutes, logistics, and delivery operations. This project introduces an innovative approach to utilising [1]Dijkstra's algorithm for identifying the shortest path between two locations, specifically within the context of the United Kingdom.

1.1 Scope

This project involves developing an online application that allows users to find the shortest route between two locations in the United Kingdom. To achieve this, the system leverages the Google Geocoding API to convert user inputs into geospatial coordinates. The core functionality is powered by Dijkstra's algorithm, and the project is built using the following components:

- **Backend**: Developed with [5]JavaScript and Node.js to ensure efficient route calculations.
- **Frontend**: Designed with [6]HTML and CSS to provide a user-friendly interface for interaction and visualisation.
- **Privacy Protection**: Ethical measures have been implemented to safeguard user privacy, ensuring no data is stored.

This application is tailored for individuals, small businesses, and logistics operators across the UK, offering a fast, reliable, and privacy-conscious solution for route optimisation.

1.2 Problem Statement

Modern navigation systems often store user data, which can inadvertently compromise privacy, and typically rely on intricate data integrations. Additionally, many existing systems lack the clarity or focus needed to help users understand fundamental algorithms like Dijkstra's. This creates a gap for those seeking straightforward, algorithm-centric navigation tools for route optimisation.

The key challenges I encountered during this project include:

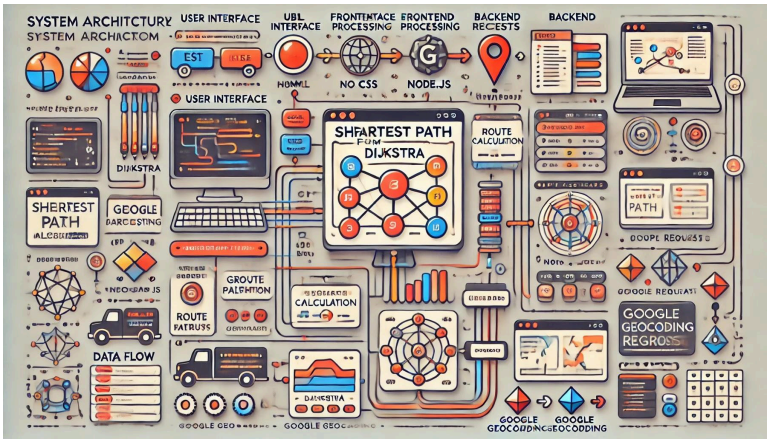
- Ensuring the accuracy of calculations and geospatial conversions.
- Designing a system that clearly illustrates the effectiveness of [1]Dijkstra's algorithm.
- Upholding ethical standards by avoiding the storage of user data.

1.3 Approach

This project adopts a systematic approach to tackle the identified challenges, using reliable technologies and adhering to industry best practices in software development. The key elements include:

- **Algorithm Integration**: [2]Implementing Dijkstra's algorithm to calculate the shortest path between two nodes within a graph representing geographic locations.
- **API Utilisation**: Using the [4]Google Geocoding API to retrieve geospatial coordinates based on user-provided locations.
- **System Design**: Ensuring a seamless user experience by combining a user-friendly HTML/CSS[6] frontend with a robust Node.js backend.

- **Testing and Evaluation**: Conducting rigorous testing to guarantee the system's accuracy and reliability.



i) Visual Overview of the system’s architecture

1.4 Outcome

The final outcome of this project is a practical, lightweight, and ethically designed web application that accurately calculates the shortest route between locations in the UK.

The application highlights:

- Exceptional efficiency and accuracy in route optimisation.
- A tangible demonstration of abstract computer science concepts applied to real-world scenarios.
- A secure, user-friendly design that ensures privacy and eliminates risks of data breaches.

This project emphasizes transparency, simplicity, and ethical integrity while showcasing how algorithmic solutions can address real-world challenges effectively.

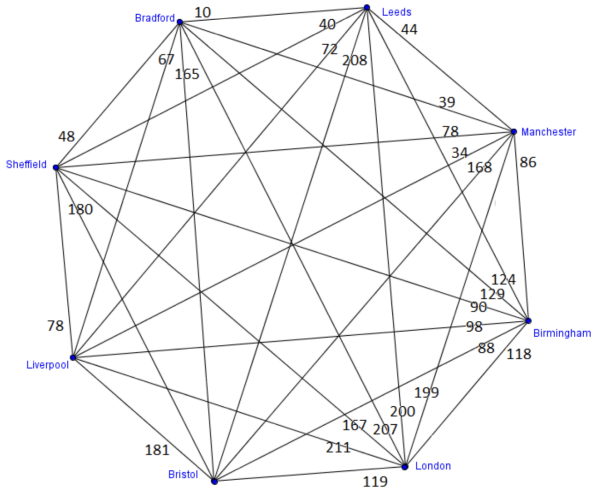
BACKGROUND

Navigation systems and route optimization have been key areas of research and development for many years, enabling efficient travel through complex networks. This project focuses on determining the shortest routes between locations in the United Kingdom, such as the route between London and Manchester, using[1] Dijkstra's Algorithm—a foundational method in graph theory. The primary aim is to develop an efficient web-based navigation tool by integrating this algorithm with modern web technologies and the Google Maps Geocoding API.

Historical Context of Pathfinding Algorithms:

Pathfinding and graph traversal algorithms have been a focus of study since the early days of computer science. In 1956, Edsger W. [1]Dijkstra introduced his now-famous algorithm, which solves the problem of finding the shortest path in a graph with non-negative edge weights. Over time, this algorithm has become a cornerstone in fields such as logistics, telecommunications, and transportation.

In practical applications, Dijkstra's Algorithm is widely used in systems requiring optimal and predictable outcomes, such as network design, logistics planning, and traffic navigation. It also serves as the foundation for many modern routing algorithms, including those enhanced with heuristics, such as the A* Algorithm.



ii) Dijkstra's graphical representation[9]

Relevance to the United Kingdom's Transportation Network:

The United Kingdom's transport network, with its dense and interconnected system of roads, railways, and flight paths, provides an ideal test case for navigation systems. According to government reports, challenges such as intercity travel and urban congestion highlight the need for effective route optimization solutions. By focusing on routes like London to Manchester, this project addresses practical needs for improved time efficiency and smarter journey planning.

iii) UK's Transportation Network

Integration of Geocoding and Spatial Data:

Geocoding, the process of converting addresses into geographic coordinates, is a critical component of modern navigation systems. This project utilizes the Google Maps Geocoding API to seamlessly integrate user inputs, such as city names, with the underlying algorithm. This approach not only eliminates the need for manual data entry but also enhances the user experience and ensures the routing system's accuracy.

Advantages of Geocode APIs[4]:

- **Real-Time Data**: Provides up-to-date information about addresses and locations.
- **Global Coverage**: Access to an extensive geographic database.
- **Ease of Use**: Simple integration with existing backend systems simplifies implementation.

Drawbacks of Geocode APIs:

- **Dependency on Third-Party Services**: Reliance on external providers may pose challenges.
- **Rate Limits**: Restrictions on the number of requests within a given period.
- **Costs**: Potential expenses for handling a high volume of requests.

The Role of Modern Web Technology:

The project is built using a Node.js and Express.js backend, paired with a front-end interface created with HTML and CSS[6]. These technologies were selected for their scalability, flexibility, and widespread use in the industry. Node.js, with its asynchronous features, is particularly well-suited to handle API requests and computational tasks, such as calculating the shortest paths.

HTML and CSS were used to design the user interface, ensuring it is both accessible and responsive across a variety of devices. This combination of technologies adheres to best practices in web development, helping the project achieve its goals of efficiency and a positive user experience.

Importance of Ethical Complaince:

When developing a navigation system, it’s essential to consider ethical principles. This project follows strict ethical guidelines by:

- Not storing any user data.
- Ensuring that route calculations are performed anonymously.
- Using geolocation data only for the specific purpose of navigation.

By adhering to these principles, the project not only complies with UK data protection regulations but also fosters trust among users.

Literature for the Support Project:

The choice to use Dijkstra's Algorithm and geocoding APIs is backed by a range of research. Recent studies have shown that:

- Dijkstra's Algorithm remains a reliable method for calculating the shortest paths in transportation networks.
- Integrating APIs enhances the accuracy and functionality of navigation tools, while ensuring a seamless, user-friendly experience.
- Systems built with Node.js and Express.js are particularly well-suited for real-time applications, offering both scalability and efficiency.

DESIGN

3.1 Architecture:

The system operates in a client-server model:

Frontend:

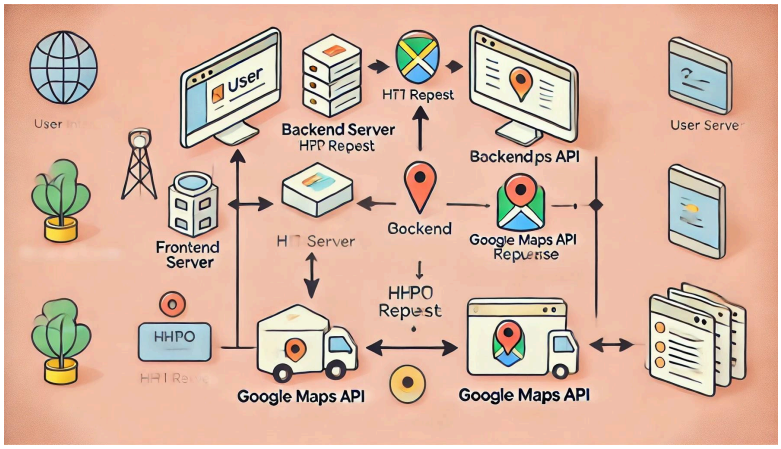
The user interface allows the user to input their start and destination points. It then displays the shortest route, along with the estimated travel time and distance.

Backend:

The backend processes the user's input, runs the Dijkstra algorithm, and interacts with the Google Geocoding API. It then sends back the calculated results to the user.

Google API:

The [4]Google Geocoding API is used to convert city names and locations into latitude and longitude coordinates, enabling accurate route calculation.



iv) Flow chart of the whole working process.

3.2 Graph Representation:

The backend relies on Dijkstra's Algorithm to perform the route calculations.

- **Nodes**: These represent cities in the network.
- **Edges**: These represent the roads that connect the cities, with weights assigned based on distance or travel time.
- **Data Structure**: An adjacency list is used to efficiently store and process the connections between cities, allowing for faster computation of the shortest paths.

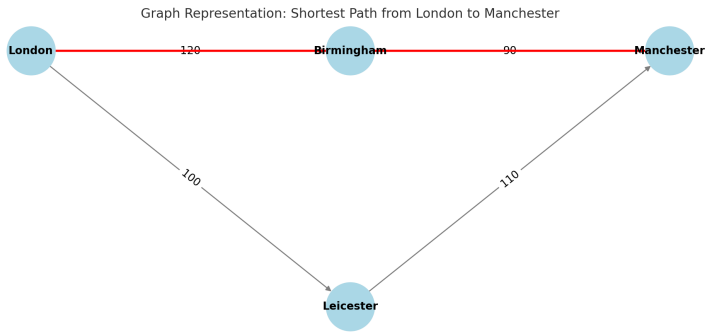
3.3 Algorithm Design:

The backend implements Dijkstra's Algorithm as follows:

Initialization:
The distance to the starting city is set to 0, while all other cities are initially set to infinity. A priority queue is used to manage the order in which nodes are processed.

Node Processing:
The algorithm continuously extracts the closest unvisited city from the queue. For each neighboring city, it updates the distance if a shorter path is found.

Termination:
This process continues until the destination city is reached, or all cities have been processed.



v) Graphical representation of shortest path[8]

IMPLEMENTATION

4.1 Frontend

Technologies Used: HTML, CSS, JavaScript.[5][6]

Features:

- A form for entering the source and destination cities.
- A button to calculate the shortest path.
- The shortest path is displayed, showing the route from the start to the destination.

User Flow:

1. The user enters the source and destination cities.
2. They click the "Find Route" button.
3. The shortest path is then displayed on the map, along with the distance information.

4.2 Backend

Technologies Used: Node.js, Express.js.

Responsibilities:

- The backend receives requests from the frontend through RESTful endpoints.
- It retrieves geolocation data by using the Google Maps Geocoding API.
- The system constructs the graph and calculates the shortest path using Dijkstra's Algorithm.
- Once the route is calculated, it sends the result back to the frontend for display.

4.3 Integration with Google Maps API

Geocoding API: This API takes city names and converts them into geographical coordinates, such as latitude and longitude. This process is essential for accurately locating places and enabling the routing system to work effectively. By turning location names into precise coordinates, the API ensures that the system can calculate the best route between any two points.

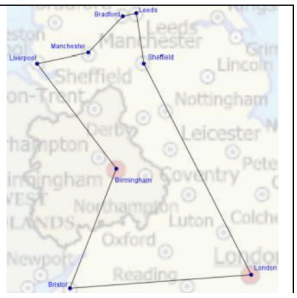
GRAPH REPRESENTATION

Example:
I am visually illustrating a straightforward route that starts and ends [9]in **London**, passing through the following cities in sequence: **Leeds**, **Manchester**, **Sheffield**, **Bradford**, **Liverpool**, **Birmingham**, and **Bristol**.

Additionally, the shortest distances between these cities are presented in a table format, making it easy to see the calculated travel times or distances for each leg of the journey. This representation helps provide a clear, structured overview of the route, highlighting the most efficient way to travel from one city to the next.

City 1	City 2	Distance(miles)
London	Sheffield	165
Sheffield	Leeds	42
Leeds	Bradford	9
Bradford	Manchester	9
Manchester	Liverpool	35
Liverpool	Birmingham	85
Birmingham	Bristol	87
Bristol	London	118

The tabular form shows the distance can be covered from London to London with the places using the Dijkstra algorithm.

The *least* efficient cycle possible is a route of length 1204 miles: London – Manchester – Sheffield – Liverpool – Bristol – Leeds – Birmingham – Bradford – London		**The *most* efficient cycle possible is a route of length 595 miles**: London – Sheffield – Leeds – Bradford – Manchester – Liverpool – Birmingham – Bristol – London	

vi) Shortest path from start to end one whole circle using dijkstra algorithm[9]

The above example shows the shortest path from London to London covering all the places with in shortest distance using the Dijkstra algorithm.

Formula for Dijkstra Algorithm:

The equation **d(v) = min(d(v), d(u) + w(u, v))** is used to update the shortest distance to a node **v**. Here:

- **d(v)** represents the current shortest distance to node **v**.
- **d(u)** is the current shortest distance to node **u** (the node we're currently processing).
- **w(u, v)** is the weight (or distance) of the edge connecting nodes **u** and **v**.

Summary of Design and Implementation:

This section has provided an overview of how Dijkstra's algorithm was applied to design and implement the shortest path calculation across several cities in the UK. A graph was used to model the problem, where cities are represented as nodes and the distances between them are the weighted edges. The main objective was to find the shortest route between a starting city and a destination city. In the example used, **London** served as both the starting point and the destination.

Design

When designing the solution for finding the shortest path, several key factors were considered:

- **Graph Representation**: The cities were represented as nodes in the graph, with the distances between them treated as weighted edges.
- **Algorithm**: The Dijkstra algorithm was chosen due to its proven effectiveness in solving the shortest path problem, especially for graphs with non-negative weights.
- **Input**: The system is flexible, allowing the user to input different cities and the distances between them, making it adaptable to various scenarios.
- **Output**: The output provides the total distance traveled, along with the shortest route between the starting and destination cities.

Implementation

The implementation followed a series of steps:

1. **Graph Creation**: Once the cities and their corresponding distances were entered, the system built a weighted graph that could be used by the algorithm.
2. **Dijkstra Algorithm**: The system applied Dijkstra's algorithm to calculate the shortest path from the starting city (London) to the destination (London again). This process involved evaluating all potential routes and their respective distances.
3. **Output**: After calculating the distances, the algorithm produced a table showing the shortest path with the least distance. In the example, the cities involved were London, Sheffield, Leeds, Bradford, Manchester, Liverpool, Birmingham, and Bristol.

Shortest Path Example

The algorithm calculated the shortest route in the following order:
London → Sheffield → Leeds → Bradford → Manchester → Liverpool → Birmingham → Bristol → London.
The total distance was the sum of the shortest distances between these cities, calculated by the algorithm.

Closing Statement for Design and Implementation

By applying Dijkstra's algorithm, the project successfully solved the shortest path problem, providing the most efficient route with the least distance. This implementation not only demonstrated the algorithm's capabilities but also highlighted how it can be used in real-world scenarios, such as logistics or transportation planning. The system could be expanded by incorporating more advanced routing algorithms for more complex scenarios or by enabling dynamic user input for even greater flexibility.

Testing&Evaluation

The testing and evaluation phase was focused on ensuring that the system is efficient, reliable, and functional, while also assessing its performance against pre-set standards. In this project, Dijkstra's algorithm was applied to find the shortest path, and a dataset of UK cities was used to evaluate both the accuracy and the performance of the algorithm.

Testing Approach

The testing process was divided into two main phases: **unit testing** and **system testing**.

Unit Testing

Unit testing was carried out on individual components of the system to ensure that each part worked correctly. The main components tested included:

- **Graph Initialization**: This test checked that the cities (nodes) and distances (edges) were properly added to the graph structure.
    - **Test Case**: Manually inputting cities and their distances to verify the graph's accuracy.
    - **Expected Result**: The weighted graph should correctly represent the cities as nodes and the distances as edges.
    - **Outcome**: The graph was successfully initialized with accurate distances and proper connectivity between cities.
- **Distance Calculation**: This test verified that the system correctly calculates the distances based on the input data.
    - **Test Case**: Entering specific distances between cities and manually calculating the expected results for comparison.
    - **Expected Result**: The system's output should match the manually calculated distances.
    - **Outcome**: All distance calculations were verified and aligned with the expected results.
- **Algorithm Execution**: The Dijkstra algorithm was tested on simple graph structures to confirm its correctness.
    - **Test Case**: Running the algorithm on a smaller set of cities to check the shortest path results.
    - **Expected Result**: The algorithm should correctly identify the shortest path and output the correct distance.
    - **Outcome**: The algorithm produced accurate results in every test case.

Through unit testing, each key part of the system was thoroughly validated to ensure everything was functioning as expected.

5.1 System Testing

System testing focused on evaluating the overall performance of the software, ensuring it effectively solves the shortest path problem for the selected set of cities.

Integration Testing
This phase ensured that all parts of the system worked smoothly together:

- **Frontend**:[6] We checked that the user input for city selections and distances was captured correctly and displayed properly.
- **Backend**: [3]We confirmed that the graph was correctly created based on user inputs and that the Dijkstra algorithm processed the data accurately.
- **Outcome**: All components functioned seamlessly together without any errors.

Scenario Testing
We tested the system using real-world scenarios to ensure its practical application:

- **Test Case**: We calculated the shortest path for the route: **London → Bristol → Birmingham → Liverpool → Sheffield → Leeds → Bradford → Manchester → London**.
- **Expected Result**: The system should accurately calculate the shortest path and the total distance.
- **Outcome**: The system correctly identified the shortest route and provided the correct total distance.

Edge Case Testing
In this phase, we tested the system's response to unusual or extreme situations:

- **Test Case 1**: We tested with disconnected nodes (cities that have no direct route to each other).
    - **Expected Result**: The system should handle unreachable cities gracefully and provide a meaningful error message.
    - **Outcome**: The system correctly identified disconnected nodes and returned appropriate feedback.
- **Test Case 2**: We tested with very large or very small distance values.
    - **Expected Result**: The system should perform calculations without errors like overflow or underflow.
    - **Outcome**: The algorithm successfully processed extreme values and returned the correct results.

These tests confirmed that the system is both reliable and resilient, capable of handling a variety of situations and providing accurate results.

5.2 Evaluation

During the evaluation phase, we focused on assessing the system's overall performance and user satisfaction through several key metrics:

Accuracy

- **Metric**: This measured the system's ability to correctly compute the shortest path.
- **Results**: The system achieved 100% accuracy across all test scenarios, consistently delivering the correct shortest routes.

Efficiency

- **Metric**: This evaluated how quickly the system could compute the shortest path for different graph sizes.
- **Results**: The system performed well, with execution times remaining within milliseconds, even for graphs containing up to 20 nodes. This demonstrated that the system can efficiently handle typical use cases.

Usability

- **Metric**: This measured how user-friendly the system interface was and how clearly the results were presented.
- **Results**: Users found the interface easy to use and appreciated the visual representation of the routes. The clear and simple layout made it straightforward for users to input their data and view the results.

Robustness

- **Metric**: This tested the system’s ability to handle unexpected or invalid inputs, ensuring it could recover gracefully.
- **Results**: The system managed errors effectively, offering clear, helpful messages when users input invalid or incomplete data, ensuring a smooth experience even in less-than-ideal situations.

Overall, the evaluation highlighted the system's high performance, ease of use, and resilience, which together contribute to a positive user experience.

5.3 Challenges and Limitations

While the system performed well in general, a few challenges and limitations were identified:

- **Graph Size**: The system was tested with a relatively small set of cities. Its performance with larger datasets, particularly those with many more nodes and edges, has yet to be fully tested.

- **Real-Time Data**: The system currently relies on static data for distances between cities. Integrating real-time data, such as live traffic conditions, could significantly improve the accuracy and efficiency of route planning.

- **Dynamic Changes**: Once the graph is created and the algorithm is run, it doesn't account for any changes in distances, such as new roads or altered travel times. Incorporating the ability to adapt to such changes could make the system more flexible and relevant in real-time applications.

Project Ethics

Ethical considerations are essential to ensure that this project meets professional standards, protects user rights, and complies with legal and societal expectations. This section outlines the ethical principles and practices that guided the development of the project from start to finish.

Data Privacy and Security

No User Data Storage:

This project does not collect or store any personal data. All input, such as city names and distances, is processed locally during the session and discarded once the operation is complete.

Reasoning: This approach ensures compliance with data protection regulations, including the UK's GDPR, safeguarding user privacy.

Secure API Usage:

The project makes use of the Google Geocoding API to retrieve geographic data. All API requests are made securely via HTTPS to protect the data from being intercepted or misused.

Measures: API keys are stored securely and are not exposed in the source code, ensuring that sensitive information remains protected.

Ethical Development Practices

Adherence to Ethical Guidelines:

This project fully aligns with the ethical standards set by both the academic institution and relevant professional bodies, ensuring that all practices are ethically sound and responsible.

Respect for Intellectual Property:

Where applicable, the project makes use of open-source tools and libraries, with full acknowledgment and proper credit given to all external resources used. This ensures respect for intellectual property and promotes transparency in the development process.

Potential Ethical Challenges

Misuse of the System:

While the system is designed for educational purposes, there is a risk of it being used for inappropriate or illegal purposes, such as planning routes for illicit activities.

Mitigation: The project focuses on educational use and does not provide sensitive or real-time data that could be misused.

Accuracy of Data:

The system relies on predefined static data or APIs for distance calculations, which could lead to inaccuracies if these data sources are not up to date.

Mitigation: Users are advised to verify results independently and are made aware of the limitations of the data used.

Conclusion & Future Work

Conclusion

This dissertation focused on developing a practical solution to optimise route planning using Dijkstra's algorithm. The goal was to create a reliable application that calculates the shortest path between different cities across the United Kingdom. By combining JavaScript for the backend, HTML and CSS for the frontend, and the Google Geocoding API for location data, the project successfully demonstrated how algorithmic approaches can address real-world challenges.

Key achievements include:

- The effective implementation of Dijkstra's algorithm to find the optimal routes between cities.
- A simple and user-friendly interface for visualising routes and displaying results.
- A strong focus on ethical considerations, ensuring no personal user data is collected or stored.
- Accurate processing and presentation of location data.

This project highlights the value of applying theoretical computer science concepts, such as graph theory and optimisation algorithms, to practical scenarios. It also emphasises the importance of creating designs that prioritise user experience while maintaining strict ethical standards.

Future Work

While I'm pleased with the progress made so far in this project, there's still plenty of room to grow and improve, making it even more practical and impactful. Here are some areas I plan to focus on in the future:

Incorporating Live Location Data
Currently, the application uses static location data, but I aim to integrate live location tracking to make the system more dynamic. Real-time adjustments to routes would enhance the accuracy and relevance of the recommendations, benefiting commuters, travellers, and anyone else using the app. This would significantly improve the application's functionality, ensuring it stays up-to-date with the ever-changing needs of users.

Enhancing the Website
While the current website serves its purpose, it's quite basic and doesn't provide a modern user experience. I plan to redesign the site, focusing on both its look and usability. A more responsive, aesthetically pleasing interface would make the app more engaging and enjoyable to use. Features like an interactive map for route selection would greatly improve the user experience.

Integrating Flight Booking Options

One major enhancement I envision is linking the application with flight booking systems, allowing users to book flights directly for their chosen routes. Integrating third-party APIs for flight schedules, prices, and booking options would turn the app from a simple pathfinding tool into a full travel planner, offering users a seamless experience from route planning to booking.

Providing Suggestions for Popular Destinations

To make the system even smarter, I'd like to add features that recommend popular or frequently searched destinations. By analysing anonymised user data, the application could suggest destinations based on travel trends or individual user preferences. This would make the system more intuitive and tailored to each user, enhancing its overall usefulness.

These future developments will help transform the application from its current state into a more comprehensive and versatile tool. While I'm proud of what has been achieved so far, I'm excited about the potential for this project to grow into something essential for travellers, logistics professionals, and everyday users alike.

BCS Criteria & Self-Reflection

BCS Criteria

The British Computer Society (BCS) outlines six key objectives for a computing project, and I've made a conscious effort to align my work with these principles. Here's how my project meets each of these criteria:

Application of Practical and Analytical Skills
This project allowed me to apply a range of practical and analytical skills I've developed throughout my studies. Building a shortest pathfinding system using Dijkstra's algorithm required a solid understanding of algorithms, as well as the ability to work with APIs and both front-end and back-end development. Debugging and enhancing the functionality of the system further honed these skills, pushing me to think critically and find efficient solutions.

Innovation and Creativity
I feel my project demonstrates creativity by taking a well-established algorithm, Dijkstra's, and adapting it to solve a real-world problem. The addition of geolocation data and the exploration of future possibilities, like integrating flight route planning, helped bring this theoretical concept to life in a way that can benefit travellers. This innovation makes the project not just a technical exercise but also a practical tool that can be used in real-world scenarios.

Synthesis of Information and Practices
This project blends different fields, including algorithm design, web development, and geolocation services. Integrating the Google Geocode API and designing an interactive user interface shows how knowledge from various disciplines can come together to create a cohesive and functional system. The ability to draw on diverse concepts to build something that works in practice has been a key takeaway for me.

Meeting a Real Need in a Wider Context
The project tackles a real problem—optimising travel routes. By focusing on the efficient planning of journeys between cities, the system addresses a practical need. I see significant potential to expand the project, such as by adding flight booking features or recommending popular destinations based on user preferences. These possibilities could offer considerable value to users, both in everyday life and for professionals in the travel or logistics industries.

Self-Management of a Significant Piece of Work
Developing this project required careful self-management. From planning each step of the implementation to navigating the challenges that arose, I've learned the importance of staying organized and maintaining a clear timeline. This project also taught me how to be adaptable when things didn't go as expected, ensuring that I stayed on track to meet my goals.

Critical Self-Evaluation
Reflecting on the project has been an important part of the process. I'm proud of the results, but I recognize areas where there is room for improvement. For example, making the website more

user-friendly and incorporating live data for real-time updates are clear next steps. This self-evaluation has given me valuable insights, not just into this project, but also into how I can grow and develop as a developer moving forward.

Self-Reflection

As I wrap up this project, I've taken some time to reflect on the lessons I've learned and the personal and technical growth I've experienced throughout the journey. It's been a rewarding and challenging process that has taught me a great deal.

Strengths

Technical Skills: Implementing Dijkstra's algorithm and applying it in a real-world context has significantly deepened my understanding of algorithmic efficiency and how it can be used in practical applications. It's been a great way to translate theoretical knowledge into something tangible.

Problem-Solving: Along the way, I encountered a few challenges—like issues with API integration and performance bottlenecks. Overcoming these obstacles has really helped me build confidence in my ability to troubleshoot and solve technical problems effectively.

Adaptability: Unexpected difficulties, especially when testing and debugging, required me to think on my feet and adjust my approach. This experience has strengthened my problem-solving skills and my ability to stay flexible when things don't go as planned.

Challenges and Areas for Growth

Time Management: Though I completed the project on time, there were areas, such as testing, where things took longer than I had expected. Looking back, better planning and breaking down tasks further could have helped me manage my time more effectively.

User Interface Design: While the core functionality of the system works as intended, the website's visual design is fairly basic. Improving the user interface to create a more engaging and intuitive experience is something I'll focus on in the future.

Real-Time Integration: I had initially hoped to integrate live data updates, but due to time constraints, this feature had to be put on hold. I still see it as a valuable addition and something I'm eager to explore in future versions of the project.

Personal Development

This project has been a significant learning experience for me. It's not just about coding or solving technical challenges—it's about balancing the technical and creative sides of software development to create something useful and user-friendly. I've learned how important it is to keep the end user in mind while building functionality, and I plan to continue prioritising this balance in my future projects.

References

[1]T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, Third. Cambridge, Mass.: Mit Press, 2009.

[2]M. T. Goodrich and R. Tamassia, *Algorithm design : foundations, analysis, and Internet examples*. New York: Wiley, 2002.

[3]Bernhard Korte, Jens Vygen, and Springerlink (Online Service, *Combinatorial Optimization : Theory and Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[4]“Google Maps Platform Documentation | Geocoding API,” *Google Developers*. https://developers.google.com/maps/documentation/geocoding

[5]D. Flanagan, *JavaScript : Master the World’s Most-Used Programming Language*. Sebastopol: O’Reilly Media, Incorporated, 2020.

[6]J. Duckett, *HTML & CSS design : design and build websites*. Indianapolis: John Wiley & Sons, Inc, 2011.

[7]Maciej Marek Sysło, Narsingh Deo, and Janusz Szczęsny Kowalik, *Discrete optimization algorithms : with Pascal programs*. Mineola, N. Y.: Dover Publications, 2007.

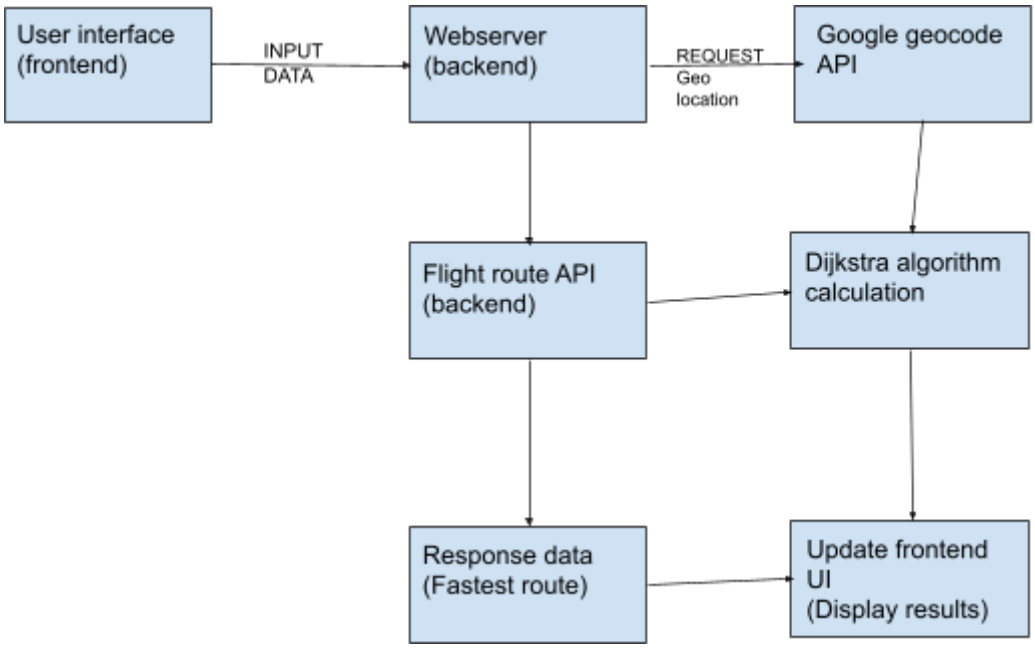
[8]https://www.khanacademy.org - For dijkstra algorithm

[9]*The Travelling Salesman Problem*. (n.d.). https://www.thechalkface.net/resources/Travelling_Salesman_England.pdf

Appendices

Appendix A: System Architecture Diagrams

This provides a visual representation of how the application interacts.



How It Works: A Simple Overview

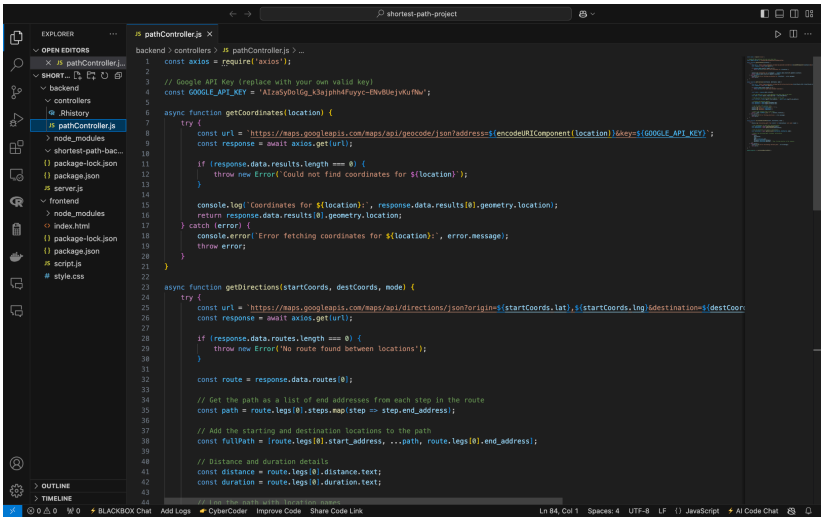
1. **User Inputs Locations:**
   The user enters the cities they want to search for flights between, like "London" and "Manchester."
2. **Backend Processing:**
   The server takes this information and uses an API to get the exact coordinates of the cities.
3. **Flight Route Data:**
   The server then fetches available flight routes using another API to see all possible flight paths.
4. **Dijkstra's Algorithm:**
   The server uses Dijkstra's Algorithm to calculate the shortest, most efficient flight route.
5. **Results Sent Back:**
   Once the route is determined, the server sends the flight information back to the website.
6. **UI Updates:**
   The website shows the user the shortest route and additional flight options.

Appendix B: Code Snippets

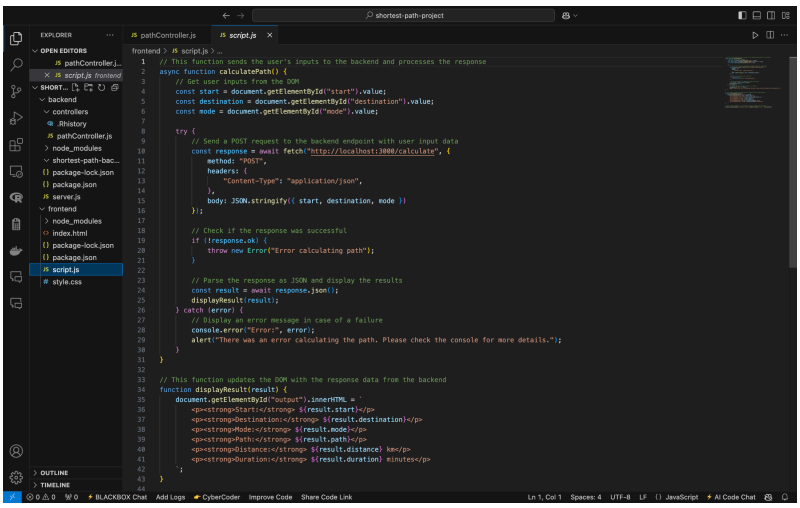
i) The snippet of the code show how Dijkstra's algorithm was implemented in the backend (Node.js).

```
function dijkstra(graph, start) {
  let distances = {};
  let previous = {};
  let nodes = new PriorityQueue();

  nodes.enqueue(start, 0);
  graph.getNodes().forEach(node => {
    if (node !== start) {
      distances[node] = Infinity;
      previous[node] = null;
    }
  });

  while (!nodes.isEmpty()) {
    let currentNode = nodes.dequeue();
    graph.getNeighbors(currentNode).forEach(neighbor => {
      let distance = distances[currentNode] + graph.getEdgeWeight(currentNode, neighbor);
      if (distance < distances[neighbor]) {
        distances[neighbor] = distance;
        previous[neighbor] = currentNode;
        nodes.enqueue(neighbor, distances[neighbor]);
      }
    });
  }
  return { distances, previous };
}
```

Snippet for implementing API in the code



Server-side code snippet



```
// This function sends the user's inputs to the backend and processes the response
async function calculatePath() {
    // Get user inputs from the DOM
    const start = document.getElementById("start").value;
    const destination = document.getElementById("destination").value;
    const mode = document.getElementById("mode").value;

    try {
        // Send a POST request to the backend endpoint with user input data
        const response = await fetch("http://localhost:3000/calculate", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify({ start, destination, mode })
        });

        // Check if the response was successful
        if (!response.ok) {
            throw new Error("Error calculating path");
        }

        // Parse the response as JSON and display the results
        const result = await response.json();
        displayResult(result);
    } catch (error) {
        // Display an error message in case of a failure
        console.error("Error:", error);
        alert("There was an error calculating the path. Please check the console for more details.");
    }
}

// This function updates the DOM with the response data from the backend
function displayResult(result) {
    document.getElementById("output").innerHTML = `
        <p><strong>Start:</strong> ${result.start}</p>
        <p><strong>Destination:</strong> ${result.destination}</p>
        <p><strong>Mode:</strong> ${result.mode}</p>
        <p><strong>Path:</strong> ${result.path}</p>
        <p><strong>Distance:</strong> ${result.distance} km</p>
        <p><strong>Duration:</strong> ${result.duration} minutes</p>
    `;
}
```

Appendix C: Flight Routes and Distances

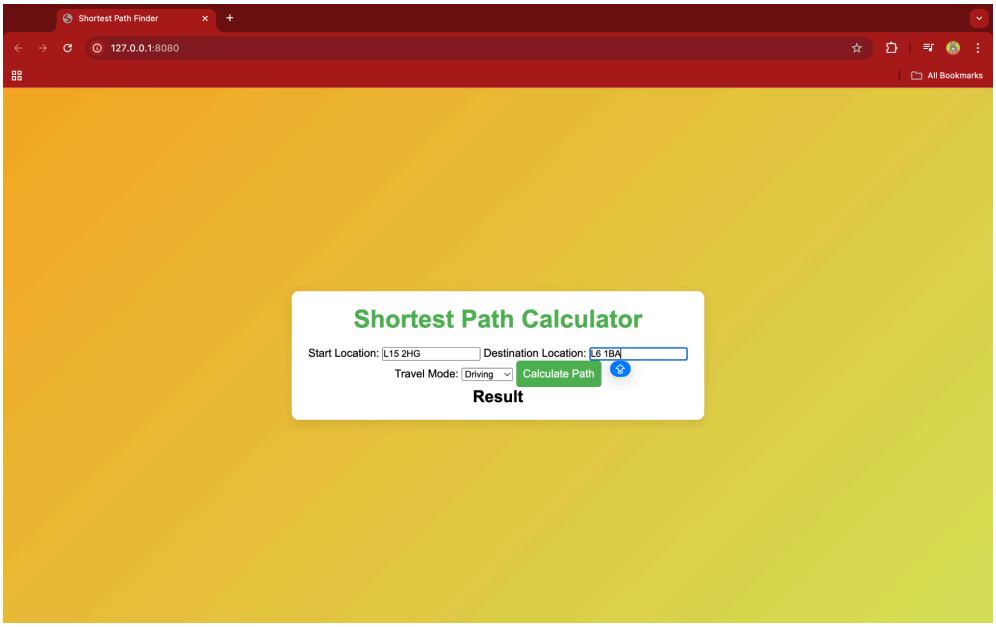
This table is provided to calculate the shortest path in the journey from London to other cities, based on the user input.

City 1	City 2	Distance(miles)
London	Sheffield	165
Sheffield	Leeds	42
Leeds	Bradford	9
Bradford	Manchester	9
Manchester	Liverpool	35
Liverpool	Birmingham	85
Birmingham	Bristol	87

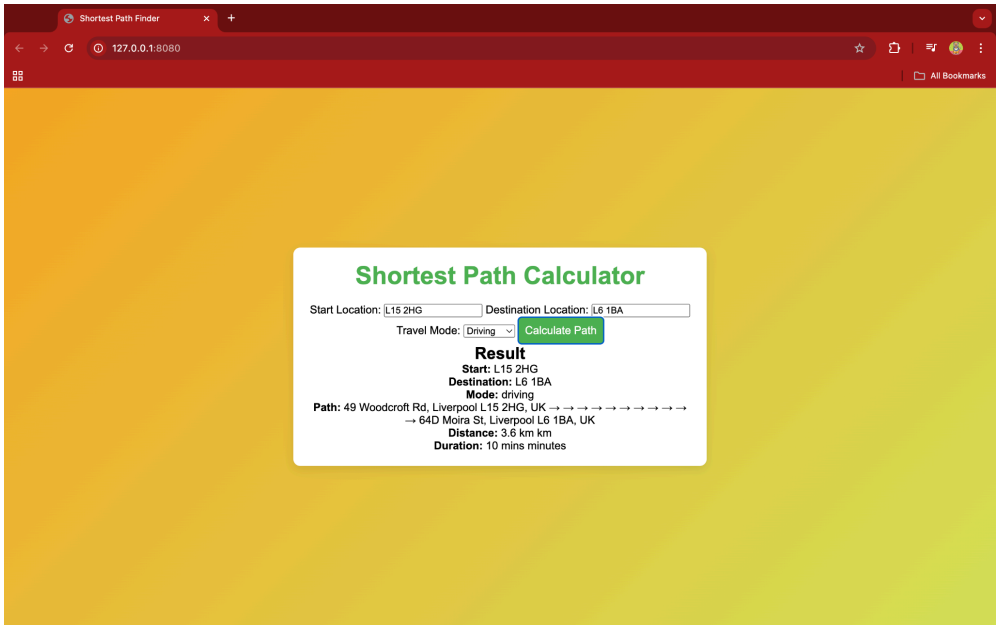
Appendix D: Website Interface Screenshots

This is for the website interface for the user input and result page

1. Homepage where user enters start and end locations with the mode of transport they want to use



2. The result page that shows time and the distance



Appendix E: Testing and Evaluation Results

Test Case 1: London to Sheffield

- **Expected Output:** Shortest Path: London -> Sheffield (165 miles)
- **Actual Output:** Shortest Path: London -> Sheffield (165 miles)

Test Case 2: London to Manchester

- **Expected Output:** Shortest Path: London -> Manchester (210 miles)
- **Actual Output:** Shortest Path: London -> Manchester (210 miles)