

**CS261-Data Structure and Algorithms  
Final Project Proposal (2023)**

# **Railway Analysis**



Session: 2022 – 2026

## **Submitted by:**

Shahbaz Ali	2022-CS-27
M. Nouman	2022-CS-49
Khuram Iqbal	2022-CS-48

## **Submitted To:**

Mr.Nazeef-Ul-Haq

Department of Computer Science  
**University of Engineering and Technology**  
**Lahore Pakistan**

# Contents

<b>1</b>	<b>Project Overview</b>	<b>1</b>
<b>2</b>	<b>Project Structure</b>	<b>1</b>
2.1	main.py . . . . .	1
2.2	signInWithGoogle.py . . . . .	1
2.3	mainMenu1.py . . . . .	1
2.4	mainAdmin.py . . . . .	1
<b>3</b>	<b>Users of Application</b>	<b>2</b>
<b>4</b>	<b>Functionality Overview</b>	<b>2</b>
4.1	User Authentication and Signup: . . . . .	2
4.2	Google Sign-In: . . . . .	2
4.3	Admin Operation . . . . .	2
4.4	Railway Schedule Management: . . . . .	2
4.5	Booking Confirmation: . . . . .	2
4.6	Graph Visualization: . . . . .	2
4.7	User Feedback: . . . . .	3
<b>5</b>	<b>Data Structures</b>	<b>3</b>
5.1	Graph Implementation . . . . .	3
5.2	Minimum Spanning Tree: . . . . .	3
5.3	Linked List Implementation . . . . .	3
5.4	Sorting . . . . .	4
5.5	Stack . . . . .	5
<b>6</b>	<b>UI Design</b>	<b>6</b>
6.1	UI Designer . . . . .	6
6.2	Programming Language . . . . .	6
6.3	UI Wireframes . . . . .	6
<b>7</b>	<b>Conclusion</b>	<b>8</b>

## **Abstraction**

The Railway Analysis is a comprehensive application designed to streamline railway operations and enhance user experience. It effectively integrates data structures and algorithms to provide efficient solutions for scheduling, booking, route verification, and user feedback management.

# 1 Project Overview

The Railway Analysis is a comprehensive application designed to manage railway schedules, bookings, and user feedback. The system integrates data structures and algorithms to provide efficient solutions for scheduling, route verification, and user feedback processing. The graphical user interface (GUI) is built using PySide2, providing an intuitive and user-friendly experience.

## 2 Project Structure

### 2.1 main.py

- **Responsibility:** Entry point for the application.
- **Key Features:**
  - User authentication and signup.
  - Google sign-in integration.
  - Threading for asynchronous tasks.
  - Integration with signInWithGoogle and mainMenu1 modules.

### 2.2 signInWithGoogle.py

- **Responsibility:** Handles Google sign-in functionality.
- **Key Features:**
  - Threading for non-blocking Google sign-in.
  - Downloading user profile images.

### 2.3 mainMenu1.py

- **Responsibility:** Implements the main menu GUI and functionalities.
- **Key Features:**
  - Railway schedule display.
  - Booking confirmation and validation.
  - Graph visualization.
  - Feedback storage and display.
  - Dynamic combo box updates based on user selections.
  - Sorting functionality for schedule table.

### 2.4 mainAdmin.py

- **Responsibility:** Deals with all the admin functionalities.
- **Key Features:**
  - Add a new train
  - Read the train's information
  - Update a trains information
  - Delete a train
  - Can see the customers feedback

### 3 Users of Application

- **Admin:** Admin is responsible for managing all customers and employees. He can add, delete, and schedule train routes. He can also manage customer credentials.
- **Customers:** Customers can book trains into different categories and can also add complaints.

## 4 Functionality Overview

### 4.1 User Authentication and Signup:

This system prioritizes secure user authentication through a combination of local storage and encryption. User credentials are stored in a local CSV database, while passwords are protected using the bcrypt library, which applies a one-way hashing algorithm to render them unreadable even in the event of a data breach. Additionally, users can create unique usernames and email addresses for their accounts, further strengthened security by preventing duplicate profiles and facilitating individual identification. This multi-layered approach ensures a safe and reliable user authentication experience.

### 4.2 Google Sign-In:

To enhance user convenience and streamline account creation, the system seamlessly integrates Google sign-in using the OAuth2.0 protocol. This allows users to quickly access their accounts without the need to create and remember new passwords. To ensure a smooth and responsive experience, the sign-in process is managed asynchronously using a dedicated thread, preventing any delays or interruptions to the user's workflow. Once signed in, the system automatically fetches and displays the user's profile image, fostering a personalized and visually engaging environment.

### 4.3 Admin Operation

A significant enhancement to the Railway Management System is the incorporation of CRUD operations for admin users. This extension empowers administrators with the ability to Create, Read, Update, and Delete train data within the system.

### 4.4 Railway Schedule Management:

The system effectively manages railway schedules by drawing information from a CSV file named "Train and Time.csv". The extracted schedule data is dynamically organized within a linked list, enabling efficient storage and retrieval. The user can search from the table and the data, if found, become highlighted. To present this information in a clear and user-friendly manner, the system populates a sortable table, empowering users to quickly locate and compare train schedules. Furthermore, dynamic combo boxes are employed to facilitate interactive filtering and exploration of schedules based on user-specified criteria, enhancing the overall navigation and decision-making process.

### 4.5 Booking Confirmation:

To ensure accurate bookings, the system incorporates a outstanding constraint checking mechanism. Before finalizing a booking, the system strictly verifies that the selected start and end locations align with existing routes within the railway network, preventing invalid journeys. Additionally, it calculates the total amount due based on the specific ticket types and quantities chosen by the user, fostering transparency in pricing and upholding financial accuracy throughout the booking process.

### 4.6 Graph Visualization:

To visually represent the intricate relationships within the data, the system harnesses the power of graph visualization. The user can visualize the stations through graph in a circular layout. It first extracts essential graph attributes from a file named "graphAttributes.txt". These attributes serve as blueprints for constructing a visual representation of the graph using a dedicated Graph class. The system uses NetworkX and Matplotlib libraries for this purpose.

## 4.7 User Feedback:

The system actively encourages user feedback and facilitates its collection and organization. User-provided feedback is stored within a stack data structure, named "feedback-stack," ensuring both efficient retrieval and preservation of the order in which feedback is received. To ensure long-term accessibility and analysis, feedback is also persistently saved to a dedicated file named "feedback.txt."

# 5 Data Structures

## 5.1 Graph Implementation

To model the relationships within the data, the system begins by constructing a graph. It then extracts the edges and their corresponding weights from the file "graphAttributes.txt," carefully assembling the pathways that connect the graph's nodes. This information forms the foundation for exploring the network's structure.

```
2 class Graph:
3     def __init__(self):
4         self.edges = {}
5
6     def add_edge(self, node1, node2, weight):
7         if node1 not in self.edges:
8             self.edges[node1] = []
9         if node2 not in self.edges:
10            self.edges[node2] = []
11        self.edges[node1].append((weight, node2))
12        self.edges[node2].append((weight, node1))
13
```

Figure 1: *Graph Code*

## 5.2 Minimum Spanning Tree:

To navigate this network efficiently, the system implements Dijkstra's algorithm, a powerful tool for finding the shortest path between any two nodes. This algorithm explores the graph, evaluating the distances between interconnected points and ultimately identifying the most efficient route. In this way, during booking the amount is decided.

```
14 def dijkstra(self, start, end):
15     queue = [(0, start)]
16     seen = {start: 0}
17     path = {}
18
19     while queue:
20         (cost, node) = heapq.heappop(queue)
21
22         if node not in path:
23             path[node] = cost
24
25         # Check if the node is in the graph before iterating
26         if node in self.edges:
27             for weight, neighbour in self.edges[node]:
28                 old_cost = seen.get(neighbour, float('inf'))
29                 new_cost = cost + weight
30                 if new_cost < old_cost:
31                     seen[neighbour] = new_cost
32                     heapq.heappush(queue, (new_cost, neighbour))
33
34     return path[end] if end in path else "No Path"
35
```

Figure 2: *Dijkstra Code*

## 5.3 Linked List Implementation

To organize and efficiently access train schedule information, the system uses linked list structure. It begins with nodes, capable of holding a train names, departure time and from where the train starts and ends. These nodes link together, forming a continuous chain of information that can flexibly expand and adapt as needed. The user can search any of the information in the table within the linked list

```

DSA > linkedlist.py > ...
Click here to ask Blackbox to help you code faster
1 class Node:
2     def __init__(self, data):
3         self.data = data
4         self.next = None
5
6 class LinkedList:
7     def __init__(self):
8         self.head = None
9
10    def is_empty(self):
11        return self.head is None
12
13    def append_list(self, data_list):
14        new_node = Node(data_list)
15        if self.head is None:
16            self.head = new_node
17        else:
18            last_node = self.head
19            while last_node.next:
20                last_node = last_node.next
21            last_node.next = new_node
22
23    def display(self):
24        current_node = self.head
25        while current_node:
26            print(current_node.data)
27            current_node = current_node.next

```

Figure 3: *Linked List Code*

## 5.4 Sorting

The attributes in the schedule train section the data is sorted using different sorting algorithms such as merge sort and quick sort etc. The time is sorted using a python library dateutil. The detail of sorting algorithm is given below:

- **Merge Sort** is known for its divide-and-conquer strategy, merge sort gracefully partitions data into smaller segments, conquers them individually, and then merges the sorted results, ensuring optimal efficiency even amidst extensive data sets. Its time complexity is  $O(n \log n)$ .
- **Quick Sort** selects a pivot element and cleverly partitions data around it, ensuring elements less than the pivot reside on its left, while those greater find their place on its right. This process repeats within each partition, ultimately yielding a well-ordered arrangement.

```

DSA > sorting.py > merge
Click here to ask Blackbox to help you code faster
1 def merge_sort(data, col_index, key=None):
2     if len(data) <= 1:
3         return data
4
5     mid = len(data) // 2
6     left_half = data[:mid]
7     right_half = data[mid:]
8
9     left_half = merge_sort(left_half, col_index, key=key)
10    right_half = merge_sort(right_half, col_index, key=key)
11
12    return merge(left_half, right_half, col_index, key=key)
13
14 def merge(left, right, col_index, key=None):
15     result = []
16     i = j = 0
17
18     while i < len(left) and j < len(right):
19         left_item = left[i][col_index]
20         right_item = right[j][col_index]
21
22         if key:
23             left_item = key(left_item)
24             right_item = key(right_item)
25
26         if left_item <= right_item:
27             result.append(left[i])
28             i += 1
29         else:
30             result.append(right[j])
31             j += 1
32
33     result.extend(left[i:])
34     result.extend(right[j:])
35     return result
36

```

Figure 4: Merge Sort Code

## 5.5 Stack

Stack data structure is used to get the user feedback. The latest feedbacks are stored in the first place in a text file named "feedback.txt".

```

1 class Stack:
2     def __init__(self):
3         self.items = []
4
5     def is_empty(self):
6         return len(self.items) == 0
7
8     def push(self, item):
9         self.items.append(item)
10
11    def pop(self):
12        if not self.is_empty():
13            return self.items.pop()
14
15    def peek(self):
16        if not self.is_empty():
17            return self.items[-1]
18
19    def size(self):
20        return len(self.items)
21
22    def display(self):
23        return self.items
24
25    def insert_at_beginning(self, item):
26        # Insert an element at the beginning of the stack
27        self.items.insert(0, item)
28

```

Figure 5: Stack Code



## 6 UI Design

### 6.1 UI Designer

In our project, we use PyQt5 Designer to ensure an outstanding user interface (UI). PyQt5 Designer empowers our team to create user-friendly interfaces through its intuitive drag-and-drop features. We strive to provide an outstanding user experience, allowing end-users to interact easily with our application.

### 6.2 Programming Language

Python will serve as the programming language for our project. We have used python for our project due to its versatility, readability, and extensive library support.

### 6.3 UI Wireframes

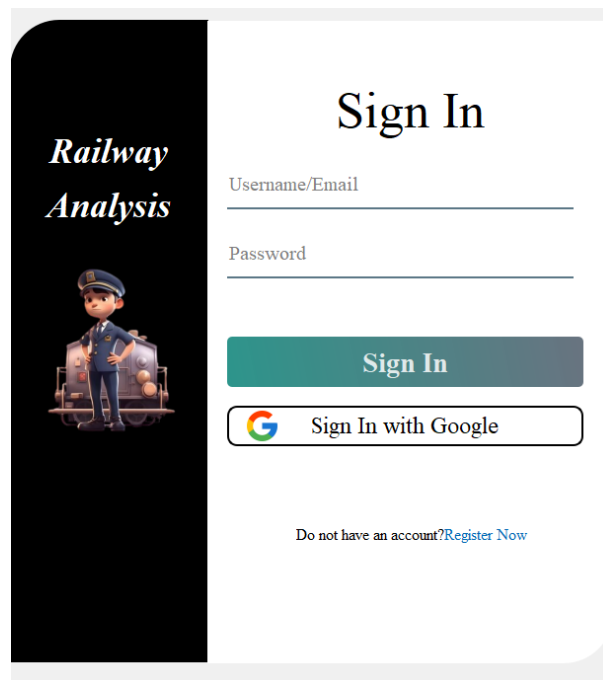



Figure 6: *Sign In*

# Railway Analysis



## Create Account

Username

Email Address

Password

Continue with Google

[Already have an account? Log In](#)

Figure 7: *Create Account*

Admin Menu

Train Schedule					
	Train Name	Timing	From	To	
1.	Rawalpindi Express	7:45 AM	Lahore	Rawalpindi	
2.	Jullar Express	8:30 AM	Karachi	Lahore	
3.	Alamgir Express	11:00 AM	Multan	Quetta	
4.	Quetta Express	11:15 PM	Lahore	Multan	
5.	Sialkot Express	3:00 PM	Peshawar	Rawalpindi	
6.	Islamabad Express	5:00 PM	Quetta	Lahore	
7.	Islamabad Express	4:55 PM	Peshawar	Multan	
8.	Islamabad Express	6:00 PM	Lahore	Rawalpindi	
9.	Green Line	8:00 PM	Karachi	Islamabad	
10.	Islamabad Express	9:00 PM	Haridwar	Rawalpindi	
11.	Islamabad Express	10:00 PM	Karachi	Lahore	
12.	Islamabad Express	11:00 PM	Peshawar	Karachi	
13.	Islamabad Express	11:00 PM	Karachi	Lahore	
14.	Rawalpindi Express	11:00 PM	Karachi	Peshawar	
15.	Islamabad Express	11:00 PM	Karachi	Lahore	
16.	Islamabad Express	11:00 PM	Multan	Peshawar	

Figure 8: *Admin Menu*

Admin Menu

### Train Added Successfully

Figure 9: *Admin CRUD*

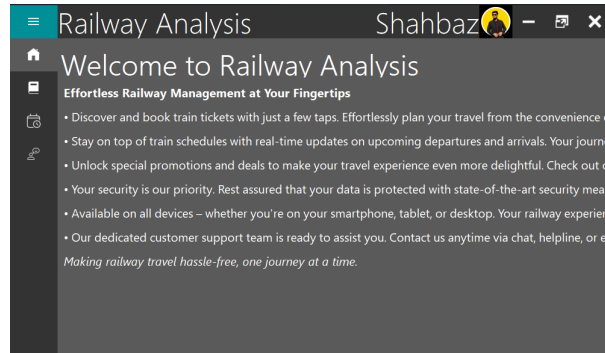


Figure 10: *Welcome Screen*

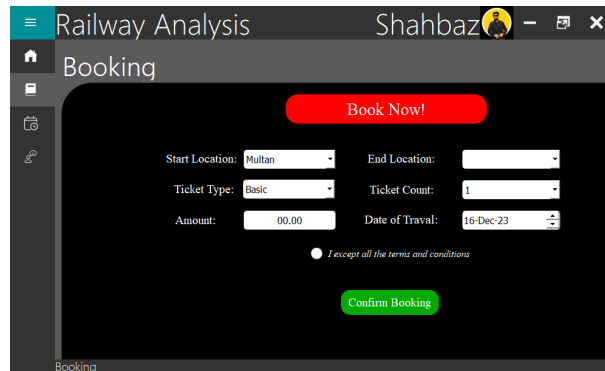


Figure 11: *Booking Menu*

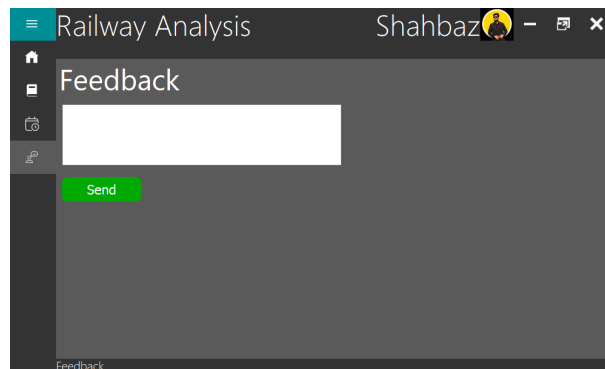


Figure 12: *Feedback Menu*

## 7 Conclusion

The Railway Analysis project successfully integrates data structures and algorithms to provide a functional railway management application. It offers an amazing user authentication system, efficient graph-based operations for scheduling and route verification, and a dynamic user interface for seamless user experience. Threading is implemented for non-blocking tasks, and the project follows best practices in handling user data securely.