Railway Management System

CSE 4301 – Final Project

Project Blueprint

Table of Contents

[Administrative Features 3](#_Toc51339817)

[Creating Stations 3](#_Toc51339818)

[Creating Trains 3](#_Toc51339819)

[Non-Administrative Features 5](#_Toc51339820)

[Making Bookings 5](#_Toc51339821)

[Additional Features 6](#_Toc51339822)

[How Bookings Are Made 6](#_Toc51339823)

[Viewing Bookings 7](#_Toc51339824)

[Viewing Station Timetables 7](#_Toc51339825)

[Memory Management 8](#_Toc51339826)

[Shortcomings 9](#_Toc51339827)

[Discarded Plans 11](#_Toc51339828)

**Railway Management System**

This project attempts to create a database management system to supervise, control and use the various parts of a railway system.

The project will allow a user to use the system in both administrative and non-administrative roles. As an administrator, the user will be able to take control of the management of stations and trains. As a customer, the system will provide an interface to manage seat bookings.

The system is organized as follows. Users in the administrative role are able to add stations and trains that exist in the real world into the database. Trains keep track of their own schedules, including information about which stations they pass through at what times, and which seats are occupied at different stations. Stations keep track of the trains that pass through them throughout the day. Users in non-administrative roles are able to make bookings based on their points of departure and arrival. They are shown different options, given that there are multiple valid trains that fulfil their requirements. The user can choose to make a booking, at which point their personal information is added to a ticket and stored on an external file. All of this information is kept track of and is visible to any administrators.

This project attempts to provide an organized, systematic and user-friendly way for both administrators and non-administrators to manage all activities related to the national railway system. The team behind this project hopes that their work can provide a basis for a more feature-filled and visually pleasing platform that can be used in real life.

## Administrative Features

Administrators are responsible for managing trains and stations in the database. As of now, it is not possible to actually remove anything from the existing database, so they can only add trains and stations.

### Creating Stations

Adding stations is as simple as giving the station a unique name. Stations are being stored in a prefix tree, with the last character of the station name containing a pointer to the actual object of the Station class.

The Station objects also contain an array of pointers to objects of the Train class, which are the trains that pass through that station. Train objects are not accessed directly, but are accessed through some Station object using this array.

When a new station is created, the name is stored in the external Stations.txt file. When the program is started, this file is checked for station names and the corresponding objects are created before passing control to the user.

### Creating Trains

Creating trains is a bit more complicated, since they contain all of the information that is used to make bookings. When adding a train, we must specify:

* the number of stations it passes through
* the names of those stations
* the departure and arrival times for each of those stations
* the distance between the stations

Each of these things is stored in a separate array, and the indices of the arrays are related to each other.

Additionally, there are two arrays for the number of available seats, for the two classes of seats, Business and Economy. These arrays work in the same manner, with the value at each index corresponding to the number of available seats at that station.

When adding trains, we must deal with a number of conflicts. Firstly, for each station the train passes through, we must check all of the trains in those stations to ensure that the time at which the new train is at the station does not conflict with any of the other trains. This is because the program assumes that a single station has only one train at any given time. We also need to check that there are no conflicts in the train's own schedule, i.e. it is not at two stations at the same time. However, as of now, this conflict is not resolved.

Once all the information has been verified, a Train object is created using the provided information. The train is also added to the array of Train pointers in each of the stations it passes through. This information is stored in the external Trains.txt file, and when the program next runs, the information will be retrieved into temporary arrays to create the Train objects again.

## Non-Administrative Features

### Making Bookings

Customers are able to make bookings by specifying their points of departure and arrival and the number of passengers. First, we retrieve a pointer to the departure Station object using the prefix tree. Once this is found, each of the trains in that station are checked for the arrival station's name. If a valid train is found, further checking is needed. We need to see if there are enough seats in either one of the classes at every station between the departure and arrival points. If there are, we consider this train a valid result. Note that one booking can be for only one class, i.e. the user will not be shown a result if there are enough seats in the train but not in just one of the classes.

All valid trains are shown to the customer, along with the time at which the trains depart and arrive and the price of tickets. The ticket price is calculated based on the distance between the points of departure and arrival and the class of ticket that the customer has chosen. If only one class is available, this is chosen automatically.

Once a train has been chosen, the customer is asked for their name and phone number. All of this information is used to create a Ticket object.

## Additional Features

### How Bookings Are Made

Bookings are managed using Ticket objects. Ticket objects are able to store information about:

* the departure station's name
* the ID of the train
* the index of the departure and arrival stations in the train's arrays
* the number and class of seats chosen
* the name and phone number of the customer

When a customer makes a booking, a Ticket object is created using this information. The name of the departure station is used to retrieve the Station object, and the train is found from the station's list of Train pointers using the ID of the train. Using the indices of the departure and arrival points, the required number of seats are decremented for all stations between those points.

Ticket objects also store themselves as objects in binary format in the external Tickets.dat file.

When the program first starts, temporary Ticket objects are read from the external Tickets.dat file and the information is again used to decrement seats as required, thus preserving bookings over multiple instances of the program.

Note that Ticket objects do not permanently exist in the database. They are created temporarily when needed, and deleted immediately afterwards.

### Viewing Bookings

Both administrators and customers are able to view bookings, albeit in different manners. Administrators view bookings for each train, so they are asked for the ID of a train. Customers on the other hand, can only view bookings for themselves. Thus, they are asked for the name and phone number they used when making the booking. A temporary Ticket object is created and data is read from the external Tickets.dat file one object at a time. When data is found that matches the input from the user, it is displayed.

### Viewing Station Timetables

All users have the ability to view the timetable of any station. This contains information about the IDs of the trains passing through the station, and the times at which the train arrives at and departs from the station. As such, this function is housed under the abstract User class, from which the Administrator and Customer classes are derived.

The process is simple enough, since the required station is retrieved from the prefix tree, and its array of Train pointers is checked to find the index for the station in each train. This is then used to find the corresponding arrival and departure times.

### Memory Management

When the program is closed, we need to invalidate all memory allocations for Station and Train objects. This is done with the help of the prefix tree. Each Station object is visited in turn, and its Train pointer array is traversed. Each station each of the trains visits is visited and the pointer in those stations for the corresponding train are invalidated, before deleting the Train object itself. If this is not done, then there will be invalid pointers that cannot be detected, which will cause the program to crash when we try to use them. When all the Train objects related to a Station object are invalidated or deleted, the Station object can be deleted.

## Shortcomings

enhancement

1. **There is no administrative login page.**

This is a security concern.

enhancement

1. **There is no way to remove trains and stations.**

Administrators should have the ability to remove trains and stations from the database.

Removing a station should not remove trains associated with that station, since those trains do not necessarily need to stop existing. However, due to the way the program is designed, we need to ensure that any trains associated with the station being deleted are also associated with at least one other station, so that we do not lose track of the Train object and cause a memory leak. We would also need to remove any information related to the station from each of the trains.

Removing a train should remove the train from all stations it is associated with. It should also discard any bookings made for that train.

1. **Conflicts where a train is at two stations at the same time are not checked when adding trains.**

bug

enhancement

1. **The scheduling system needs improvement.**

The current system only has a single, limited schedule that does not take dates into account. The program needs to be modified to accommodate a real-time and ever-changing schedule.

Additionally, there needs to be scope to edit schedules to accommodate changes and delays.

1. **The external data storage system needs improvement.**

enhancement

No security measures have been taken to protect the data being stored externally and no measures have been taken to prevent data loss. It would be best to switch to a proper database instead of using a flat-file system.

enhancement

1. **Limitations are imposed by usage of arrays.**

The use of arrays limits the capacity of the program. It would be best to replace these with vectors.

1. **Customers should be able to book tickets in different classes at the same time.**

enhancement

enhancement

1. **There should be a graphical interface.**

bug

1. **Revisions to the basic design of the program are needed.**

There are places where the code becomes extremely messy and confusing, especially with the number of indices of arrays and pointers thrown around.

## Discarded Plans

Initially the program had three classes, Station, Train and Route. Route was intended to keep an array of stations through which trains on a particular route would travel. This would mean storing all the Station objects in the route, which would take up a lot of space. To avoid this, pointers were considered. The theory was that every object of the Station class would have pointers to all the other Station objects connected to it and the route would follow the pointers until an invalid pointer was reached.

However, this would have caused two problems. Firstly, a large number of Station objects would mean a large number of pointers, which would in turn make the entire program very messy. Secondly, this theory falls apart if there are multiple connections to each Station object, as in the real world, since the route must be told which pointer to follow at every station, which is not possible.