Community Based Train Locating System (CBTLS)

D.N.H Senevirathna

139180A

Dissertation submitted to the Faculty of Information Technology, University of Moratuwa, Sri Lanka for the partial fulfillment of the requirements of the Master of Science/ Post Graduate Diploma in Information Technology.

March 2016

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

D. N. H. Seneviratha
Name of the Student

Nadeesha
Signature of the Student

Date: 2016/04/27

Supervised by

UOM Verified Signature

Name of the Supervisor

Signature of the Supervisor

Date: 27/04/2016

Dedication

This thesis is dedicated to my parents, Mr. D.S. Senevirathna and Mrs. A.P.P Karunasingha for their endless love, encouragement and support.

Acknowledgements

First and foremost I would like to offer my sincere gratitude to my research supervisor Mr. S. C. Premaratne, for his guidance, supervision, encouragement and support throughout this study.

I would also like to thank all the lecturers of Faculty of Information Technology – University of Moratuwa, for their guidance and encouragement to get maximum use of knowledge and capabilities.

I am grateful to the management and colleagues at PricewaterhouseCoopers Sri Lanka for their kind support, encouragement and understanding during this work.

My special thanks must go to Dr. Dilani Wickramaarachchi at University of Kelaniya, for her kind support, guidance, motivation and encouragement throughout the project.

Finally I would like to extend my deepest gratitude to my parents and family, for their continuous support given in every possible way to make this project a success.

Abstract

Rail transportation has been considered as a main mode of transportation in Sri Lanka since a long time. Therefore it is important to further develop and enhance railway transportation as an alternative method of transportation, especially considering the traffic congestion that could be observed in city areas. With the advancement of information technology, over the past time there have been many attempts to enhance the quality of railway services, but despite of them, some major concerns for the train passengers in Sri Lanka still remaining unsolved to date.

The main objective of this project is to propose and implement a crowdsourced real time train tracking system based on GPS named Community Based Train Locating System (CBTLS), for the benefit of train passengers and train transportation of Sri Lanka, aiming to address the major concerns and enhance the railway service.

CBTLS is a community based (crowdsourced) system, therefore data is retrieved from the train passengers, and then organized, processed and analyzed by the system, and resulting information and predictions is given back to the train passengers.

The proposed system consists of a native Android mobile application and a Web application. Any train passenger with a smart mobile device or a computer would be able to access the system through internet, update the train locations, compartment details, and view current and/or last known locations of a train, view analysis, predictions and suggestions on train schedules. Other than static train schedules, rest of the data required for system's functionality is acquired from the train passengers, hence the system is community based.

As an additional feature, a location aware alarm clock is integrated into the native android application, for the use of passengers to indicate when their destination has been reached.

Other than train passengers, the system consists of an administrative functionality as well. System administrators hold responsibility to control and overview the user accounts created by train passengers and manage static master data.

With this system, it is expected to facilitate train passengers to make better travelling decisions by providing required information for them, hence facilitating efficient usage of railway services.

Table of Contents

List of Figures/Tables

Chapter 1

CBTLS – Community Based Train Locating System

1.1 Introduction

This chapter mainly focuses on the motivation, aims and objectives of the community based train locating system. Here it is described some of the key problematic areas in the current train transportation system that could be observed in Sri Lanka, mainly in the train passengers perspective, hence identifying the problem to be addressed. At the same time, it is briefly explained the proposed method of addressing the problem in this chapter.

1.2 Background and Motivation

In today's context, in city areas, especially around and in city of Colombo, a heavy traffic congestion could be observed daily on the roads, and it has become one of the major concerns in country as well. School students, University Students, Government and public sector employees, and general public have been facing a crisis when it comes to travelling in and out from Colombo daily.

As the study done by A. Kumarage indicates, there can be different approaches available to solve this issue involving both short term and long term strategies. One of the key and less time and resource consuming approach would be to enhance and develop already existing alternative methods of transportation for general public. The valuable man hours and other resources which are wasted on roads daily could be easily preserved by introducing proper alternative methods of transportation and by enhancing the efficiency, reliability and quality of currently available public transport systems [1].

As it indicates in the work by Jayasinghe and Pathiranage, it is very important to support and enhance railway transportation as an alternative method of transportation[2].

When considering such already available alternative transportation methods, Rail transportation has been considered as a main mode of transportation in Sri Lanka since a long time.

The significance of enhancing train transportation service has been indicated in the annual report 2012 of Central Bank of Sri Lanka. In the annual report 2012 of Central

Bank of Sri Lanka, it has indicated that the Sri Lanka Railway has a great potential to improve their current services for both passengers and freight, and it will cause the city traffic congestion to be reduced to a great extent. Further, the report indicates that is has been already identified that the current issues in the service like inadequate coverage, lack of carriages, and most importantly the key point "inefficiency". It indicates that due to the said issues, general public are to seek for other modes of transportation, causing heavy traffic congestion, and ultimately leading to loss of productive man hours and energy utilization[3].

As mentioned above, the productive man hours, energy could be saved, and the heavy traffic congestion could be avoided to a certain extent, especially around city areas, by enhancing rail transportation service. The current issues mentioned above, inadequate coverage and lack of carriages should be addressed by providing required physical infrastructural resources for the service.

The issue "inefficiency" could be considered as a main reason for general public to consider other modes of transportation in place of trains. The main objective of the system proposed here (CBTLS) would be to provide a means for the general public to use this "inefficient service" efficiently.

When considering the statistics provided by Ministry of Internal Transport - Sri Lanka, which is given in the table below, no. of passengers who has chosen train, increasing annually[4].

	2010	2011	2012	2013
Total trips operated (Both passenger and Goods trains)	116,912	119,392	121,782	122,269
No.of Passengers Carried (in millions)	101.45	96.11	106.05	118.71
Length the passengers carried on (Km in million)	4,352.83	4,574.19	5,039.45	6,257.38

Table 1.1 - Sri Lanka Railways - Operational Statistics[4]

Considering data shown in Table 1 above, the important figure in this context would be the "No.of Passengers Carried". The variation of this figure is given below,

Figure 1.1 - No.of Passengers Carried (in millions) over 2010 - 2013 period

The increased number of passengers over the years indicates the increasing demand for the train as a mode of transportation.

In order for Sri Lanka Railway Service to draw and retain more passengers with their service, there are several issues which need to be immediately addressed. Delay of Trains in Sri Lanka is a very common situation, which train passengers have to face daily. It has turned in to be an unavoidable scenario over the time and the train passengers who are frequently using the service, have got used to it. As a transportation service, railway should maintain its reliability, and it's important in country's economic aspects as well. The mentioned issues here should be resolved within Railway Department of Sri Lanka after doing a proper analysis of the causes, but the aim of the proposed system (CBTLS), here is to facilitate the train passengers to use this existing inefficient service efficiently with the aid of the system.

Usually, a train might get delayed from few minutes to many hours in Sri Lanka and in certain commonly observable scenarios, trains gets cancelled as well. This is without any prior notification for the train passengers. According to media, even in recent history in Sri Lanka, in such situations clashes has occurred among the passengers and the railway administration. But the problems still remain unresolved to date.

A fixed schedule is maintained by the Railways department of Sri Lanka on train arrival and departure, it is available online as a web application and also as mobile applications. The major issue with the static schedule is, it does not get updated based on potential delays and cancellations.

As a result the commuters face many problems and waste time and energy that could have been used more productively. For people who are using trains for their daily travelling, there's no means of recovery of their time in scenarios which trains are delayed or cancelled.

If the passengers could know beforehand, whether the train they expect to travel is on time or not, preferably before coming in to the station, they would be able to make a better decision on their method and time of transportation. An ideal situation for them would be to know the current location of the train.

In railway administration's point of view, if they could collect the train details on each and every schedule daily, along the entire route, that data could be used to analyse the existing issues in the system, the reasons for the train delays, locations where trains gets delayed. Then that analytical information can be used to identify the issues, find solutions to them and finally enhance the service.

Similarly, such information is important for passengers as well, especially when deciding which train to travel on, since the expected time of arrival at the destination indicated in the railway's timetable would be somewhat different from the actual time of arrival.

There are passengers who uses trains for their daily travelling, or frequently, who are much familiar with the railway system, specially the stations. There are some passengers who might seldom use railway transportation, especially like tourists. Such people might not be aware of the location of destination stations they want to travel to.

Such scenarios could be observed while travelling in train, people get in to the wrong train, which will not stop at their destination station, or people who have missed their destination station.

Usually in train stations in Sri Lanka, an announcement is made when a train is arrived in the station, indicating the next set of stations in where it would stop along with the final destination. But these announcements are not very clear sometimes, and most of the time is in Sinhala language only in smaller stations. Therefore people like tourists, who are not familiar with Sinhala language, would face issues when finding the correct train and destination stations. If there was a way for them to get an indication when they are reaching their destination, it would be a great help and they would grow in confidence to use railway service more frequently.

1.3 Aim and Objectives of the CBTLS

The aim of this research is to provide a comprehensive software application solution - named as Community Based Train Locating System (CBTLS), for the train passengers in Sri Lanka, which would help them for an efficient usage of current train transportation service in Sri Lanka.

CBTLS would be aiming at enhancing the usage of rail transportation service in Sri Lanka for passengers, by introducing new features for them which are not available in current systems as given below,

- Facility for the passengers to update train's current location actively or passively
- Searching and locating trains in real time
- Providing information about the passenger density in each compartment of the selected train
- Predicting and suggesting most suitable train to take based on destination, and time of arrival at destination desired by the passenger.
- Analysis of data collected over a period on a given train, and indicate more accurate travelling times.
- Location based alarm to indicate if the passenger has reached the destination.

The proposed system will also be an enhancement and combination over the features available in currently available systems for the same purpose.

Additionally, the CBTLS would facilitate the storage and analysis of historical data related with each train by storing them in a centralized database. With this facility, authorized users would be allowed analyze patterns of train travelling daily, hence the delays could be observed.

As a community based system, it would allow registered users to post their comments, criticisms and suggestions regarding a selected train. Authorized users would be allowed to view these comments, criticisms or suggestions by the passengers.

1.4 CBTLS Implementation – how will it address the issues

The proposed system would be consist of a web application and a mobile application. Web application would cater as the backend for the mobile application, while facilitating all the functionalities available in the mobile application as well. Web application backend and native android mobile application would communicate through a REST (Representational State Transfer) API (Application Program Interface).

When it comes to the web application, when actively updating train locations and compartment details (while travelling in the train), the user's location would be tracked through the web browser in contrast to the native android application, in which the location would be gathered through GPS and the Network Location Provider of Android.

Initially, the available static train schedule details from Sri Lanka Railways would be inserted to the system as master data. This data would be considered as a base line through the rest of the application. For this purpose, the data integration module is available in the system and the web service available from Railway Department through Information and Communication Technology Agency of Sri Lanka (ICTA) is used.

A proper database structure is defined in order to store these kind of master data and the data received from passengers on each occurrence of this train schedule. Design of this data structure is a key part of the system.

A location aware android mobile application consisting of a train details (location, compartment details) update and a train details view part would be developed for the use of passengers.

Considering train details update part for the system from passengers,

- A simple user interface would be provided for them to indicate if a selected train has arrived or not at their location, when updating the train locations, it could be done in two ways in the system
    - Actively update – The passengers who are already on board the train could actively update the train's location. This could be done once or else a facility is available for the users to keep updating the location

automatically along the entire journey. In this process, the train location would be determined based on user's current location

- Passively update – The passengers or anyone who's aware of train's location (e.g. People who live around train stations, rail roads) could use this facility to update train's location, and in this process, the train location would not be determined based on user's current location, rather it would be determined based on last train station, along with the data, if train is in the station, moving or stopped after the station.

- The data inserted would be validated against the predefined geo coordinates of the selected rail route, before accepting into the system.
- A simple user interface would be provided for passengers to indicate the passenger density of their current compartment, or the overall density in the entire train. Sometimes a user may not aware of the current compartment number he is in, in such situations, a general indication could be updated. The total number of compartments is also retrieved from user, therefore an average number would be indicated as the total number of compartments in the given train turn.
- The crowd density states are predefined as levels – Low, Medium, High, Very High, and user can easily select the compartment number and select a level and update.
- The compartment number could be selected within a given range, and in the same way, total number of compartments in the train could also be selected.
- Since these data is updated by general public, there's a great possibility for entering inconsistent and inaccurate data in to the system, and to clean such data validations are placed in every possible scenario.
- The train passengers could setup a location aware alarm using the native android application, by choosing the destination train station and at which time the alarm should notify – before a certain distance to the destination station, at the train station or after the train station.
- A user interface would be provided for passengers to enter their comments/suggestions/criticisms into the system regarding a selected train, this is supposed to be cater as a review for other passengers. This data is stored on a train schedule basis rather than a single occurrence of a train schedule, this data

could be provided to the users along with other analytical data available regarding the train schedule.

- Using this application, passengers would be able to mark their frequently used train schedules as "Favorites", and after that they would have easy access to the most frequently used train schedules, rather than searching always.
- A user will have to login to the system to use certain customized functionalities like favorites, and also to update train locations. A single user account could be created using a user name and a password, and using that, a user can log in to the system using any android mobile device. With this design, the user's preferences would be saved across multiple android devices and/or web application.

Considering data view part of the system for passengers,

- A UI would be provided to search for a train, this is similar to the functionalities of existing systems, that's is to select a start station, end station and if preferred a date and a time range. With the given criteria, a set of train schedules will be loaded, from master data obtained from Sri Lanka Railway services.
- With the "Favorites" facility, for regular passengers this search functionality could be skipped and the required train schedules could be directly loaded to the system.
- The rest of system's functionality will mainly base on the schedule selected here.
- The passengers would be able to view last known location of the train to the system (must have been reported by another passenger), for a selected train, together with the crowd density details reported for each compartment.
- The scheduled time for the train to reach the user's station (or the nearest station for the user), and the general deviation of it along the time, and predicted reaching time would be displayed for the passenger.
- When viewing details reported by other passengers, an indicator about the confidentiality would also be displayed for the passenger

A web application would be developed specifically for the admin functionalities like maintaining master data, moderating user comments in addition to the functions available in the mobile application. It would also provide the analytical functions

related with data mining, in order to analyze the patterns of transportation of a single train. This would allow the authenticated users to determine any delays at specific points of journey. The same set of users would be able to see the feedback from passengers regarding the selected trains.

The mobile application would be a native android application, and the web application would be developed mainly based on Java EE. Spring and Hibernate Frameworks are used in the web application, and for the UI, bootstrap and JQuery is used.

Railway transportation service has been popular among various types of passengers. Based on how frequently the train has been used, various types of passengers can be categorized roughly as below,

- Daily Users - train is used as the main method of transportation daily

  e.g.:-

  - Workers in both private and government sectors, travelling daily to the working places
  - School students,
  - University students

- Weekly users - train is used as the main method of transportation weekly

  e.g.:-

  - Workers, of whom the working places are in Colombo, travelling from faraway places like Galle, Kandy, and Anuradhapura.

- Occasional users - train is used as the main method of transportation occasionally

  e.g.:-

  - Tourists

When the passenger type - daily users is considered, it can be observed most of the time their travelling pattern has been similar along time. The train, the compartment, sometimes even the seat row which is being used to travel has been the same. The system which is proposed here has mainly targeted this passenger type.

Other user types also could be highly benefitted through this system, since its features like location aware alarm, and analysis of train schedules.

When the factors mentioned above, the importance of rail transportation, and the increasing demand and usage of rail transportation in Sri Lanka, are considered, any contribution to enhance it as a service for general public would be of great value. The system proposed here has been aimed to be a contribution for that.

1.5 Structure of the Dissertation

The next chapter (Chapter 2) describes the review of similar systems currently available and the similar systems proposed to address the same issue. There are several systems available currently for the same purpose as CBTLS – to enhance Railway Transportation system in terms of efficiency and reliability, also there are researches done for the same purpose. Some of these were reviewed before proposing the CBTLS and the review details are indicated in this chapter. After that in Chapter 3, the technologies adapted in this CBTLS system would be explained. The overall architecture of the system, and the reasons to justify the selection of technologies would be described here. Chapter 4 would describe the proposed CBTLS system's approach to address the efficiency and reliability issues in current railway transportation system which are described in detail in the section above "aims and objectives of CBTLS", along with the technologies adapted which is described in Chapter 3. In chapter 5, system design is described for the community based train locating system. The wireframes for the mobile UI, the class diagram, ER diagram and the sequence diagrams are included and described in this chapter. Also the modules available in the system and the interaction among them is also described in detail here. Chapter 6 would describe the implementation details each module which was described in chapter 5. Here the details about software, hardware used for the system is mentioned, and at the same time details of critical algorithms used in system are described. In Chapter 7 the evaluation methods used for this system will be described in detail. The design of the questionnaire, selection of users and final results would be described in this chapter. Chapter 8 is about the overall achievements of CBTLS, problems that had to be addressed while developing the system, and limitations of CBTLS and how it could be enhanced further in the future.

Current approaches available to address the Issues in Railway Transportation System

2.1 Introduction

Railway Transportation service in Sri Lanka is owned by the government of Sri Lanka and functions as a public service offered to citizens by Sri Lanka railways Department. To use this public service efficiently, the availability, and easy access of information regarding the service is a critical factor for train passengers. Based on the available information, the passengers would be able to make decisions on their travel plans.

Since the railway transportation service in Sri Lanka is owned by the public sector of the country, the government authorities have been seeking methods to improve the efficiency of this service. The main objective of such efforts is to provide a better service to the train passengers.

According to the work done by G. Bradley, it suggest that in most of countries the governments have already recognized the potential and importance of the implementation of Information Communication Technology in the key areas of their services for general public. At present, Information Communication Technology is playing a key role as a main tool used to enhance the quality and allow easy access to government services with the aim of providing a better and efficient service for the general public[5].

As a result, e-Government and m-Government like concepts have been introduced to use Information Communication Technology as an interface to provide services offered by the public sector as well as to distribute the required information to general public of the country.

As it is indicated in the work by S.Rainford, at present, in Sri Lanka most of the key public services has been integrated with ICT already and as a still ongoing project, rest of the services are also planned to be integrated in the future. As an example, public services like revenue license issuance, wildlife bungalow reservation service are already available as e-Services, furthermore public information services like the exam result publishing service, vehicle information service, and train schedule information

service are available as e-Services. Through the currently available service, public has access to the static train schedules[6].

Based on this e-Service provided by Sri Lanka railways Department, with the support of Information and Communication Technology Agency of Sri Lanka (ICTA), there are several applications build, both mobile and web applications for the benefit of train passengers. Some of these systems are reviewed in this section.

In addition to the currently available system, there are some proposed systems available for the purpose of enhancing the railway services, and they would be also reviewed here.

2.2 Currently Available Systems for general public in railway transportation services

When the currently available methods of information retrieval by train passengers are considered, certain drawbacks could be seen in them. A list of such services that could be found online is listed below

2.2.1 eService by The Department of Railways [7]

This is the main service offered by Department of Railways, with the support of Information and Communication Technology Agency of Sri Lanka (ICTA), even for CBTLS, the master data was obtained from this system. The backend for this service is hosted as a web service, and same service is used in both mobile and web applications. Here the web application is analysed and the mobile application is reviewed in the section below.

This provided e-Service can be accessed via the given url here - http://www.eservices.railway.gov.lk/schedule.

Same service has been implemented as several different mobile applications and they are listed after this.

This service is aimed mainly at displaying the static train schedule for the users – anyone who has access to the service. According to the instructions given in the service below are the steps to use this service and information which could be obtained from it.

An enquiry can be placed by providing start and destination stations. This service allows viewing Train schedules with time/ station details and ticket prices according to the search criteria.

In this service, mandatory details to search for train would be start and end railway stations as in schedules.

Search Train Schedule Procedure in the system is given as below,

- Start station should be selected from drop down menu –mandatory.
- End station should be selected from drop down menu –mandatory.
- If required Start and end time can be selected, otherwise if the date is current date, train schedules would be displayed from next available train.
- If required to search for a train on different date other than current date, a search date from the calendar could be selected.
- After clicking on 'Search' button, a list of train schedules would be displayed.

Figure 2.1 – Initial Screen of e-Service Offered by Railway Department[7]

The above System will display Train time table with following details

- Direct Trains

- Arrival time
- Departure time
- Destination/ Time
- End station/ Time
- Frequency
- Name
- Type
- Available Classes &
- Train number
- Connecting Trains(if available) – Same as above details
- Ticket Prices
- Class name
- Price (Rs.)
- Total Distance

Figure 2.2 – Train Detail Screen of e-Service Offered by Railway Department[7]

Drawbacks as observed,

In this system, only the static schedule data is displayed, and there's no way of confirming if the train is available or not in real time. The system does not offer a method to view train delays. And there's no way to locate the trains in real time.

2.2.2 Android Mobile Applications available in the Google Play marketplace;

- Sri Lanka Train Schedule [8]

This application is provided by Railway Department of Sri Lanka with the help of Information and Communication Technology Agency of Sri Lanka (ICTA). As mentioned in the above section, this is the mobile presentation of the service offered by government. This application can be installed in android mobile devices, as same as the web application, it requires an active internet connection to work. Since this is a mobile application, it facilitate the train passengers to access train schedules while they are moving.

The application is available in google play store in the following location: - https://play.google.com/store/apps/details?id=lk.icta.mobile.apps.railway

The following information is available in the google play store regarding the application,

"Sri Lanka Train Schedule application is developed under the initiative of delivering government e-services which are connected to Lanka Gate through smart phone mobile interface. From this application you can get Train Schedule and Ticket Price information from Sri Lanka Railways."[8]

Drawbacks as observed,

In this system, only the static schedule data is displayed, and there's no way of confirming if the train is available or not in real time. The system does not offer a method to view train delays. And there's no way to locate the trains in real time.

- Train Schedules of Sri Lanka [9]

Train Schedules of Sri Lanka provides another mobile user interface for the same service as above. Additional features are added to this system such as the facility to add train schedules to favorites and to store last 10 searches in history.

The application is available in google play store in the following location: - https://play.google.com/store/apps/details?id=com.aselalee.trainschedule

Drawbacks as observed,

In this system, only the static schedule data is displayed, and there's no way of confirming if the train is available or not in real time. The system does not offer a method to view train delays. And there's no way to locate the trains in real time.

- Train Guide - Sri Lanka [10]

Train Guide - Sri Lanka is another mobile application which is to provide another mobile user interface for the web service offered by Railway Department of Sri Lanka. To this system, certain additional features are added as search history to be available offline access, and the location awareness to find the nearest train station.

The application is available in google play store in the following location: - https://play.google.com/store/apps/details?id=k.dw.timetable

Drawbacks as observed,

In this system, only the static schedule data is displayed, and there's no way of confirming if the train is available or not in real time. The system does not offer a method to view train delays. And there's no way to locate the trains in real time.

In addition to the mobile applications indicated here, there are several more applications, but all of them share the same basic set of features and therefore share same drawbacks. The above mentioned systems have integrated some additional customized functionality to the system in addition to the features available in the offered service from Railway Department of Sri Lanka.

2.2.3 GPRS based Railway Traffic Optimisation System (RTOS) by Sri Lanka Railway with University of Colombo [11]

In 2014, a new train tracking system named Railway Traffic Optimisation System (RTOS) was launched for the coastline enabling the general public to view train movements by the Railway Department Sri Lanka with the association of University of Colombo's School of Computing (UCSC) [12]. The system was available for the general public via the link - www.slrail.info. It has been indicated that the system was developed after a study done on technical issues. The research study has been carried

out by joined effort of undergraduates from UCSC and the National Research Council of the Department of railways of Sri Lanka. As a solution for the issues encountered in the research, which are train delays and safety issues, the Railway Traffic Optimisation System (RTOS) has been developed.

The Railway Traffic Optimisation System (RTOS) has been mentioned as based on General Packet Radio Service (GPRS), and information such as the train destinations, train speeds, train schedule and information on cancellations provided through this system. Additionally to general public, the information from the system is available for the internal staff like engine drivers and station masters, facilitating them to monitor train movement.

This system could be accessed via - http://www.slrail.info/ , but on a special note, at present the system is unavailable via the given URL (Uniform Resource Locator). Below is a screen shot of the system taken while it was available online.

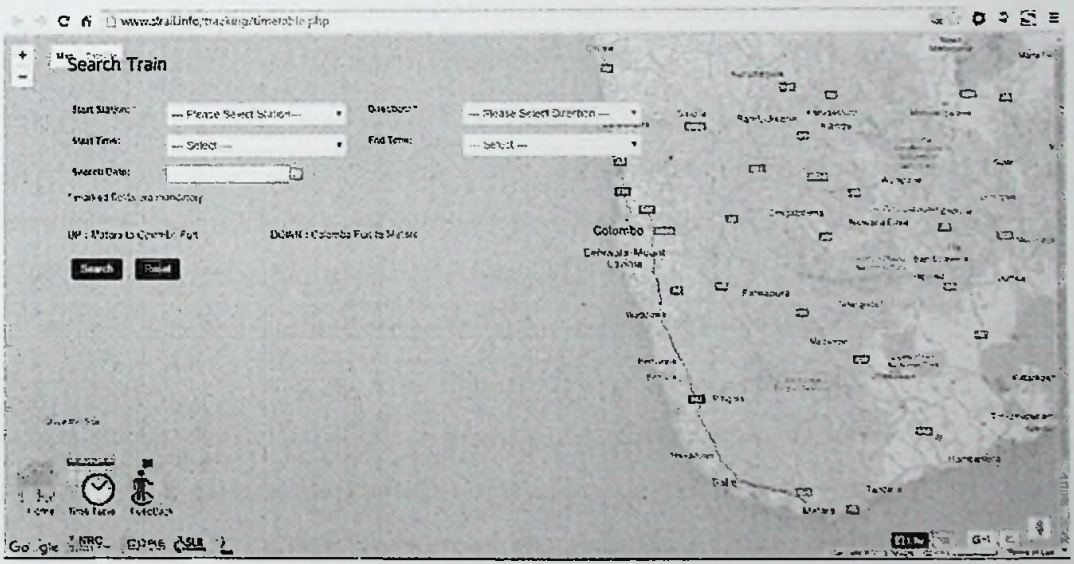


Figure 2.3 – Search Train Screen of GPRS based Railway Traffic Optimisation System (RTOS) by Sri Lanka Railway with University of Colombo [11]

The system is implemented only for Coastal Line, and the current location of the train could be seen on a map. An enquiry can be placed by providing start position and destination stations. Compared to the other system available for general public to access information on train details, this system contains considerable advanced features such as to view train locations in real time and get details about train delays

Drawbacks as observed,

Implemented only for Coastal Line and currently the system is not functional via the given URL, system maintain ace is not properly in place as it is observed.

2.2.4 A proposed system - GPS/GSM based train tracking system – utilizing mobile networks to support public transportation [13]

Dileepa Jayakody, Mananu Gunawardana and coworkers have proposed an intelligent train tracking and management system to be implemented in Sri Lanka for the purpose of improving the existing railway transportation system. According to their study, the proposed system is a combination of technologies like Global System for Mobile Communication (GSM), Geographical Information System (GIS), Global Positioning System (GPS) and a custom software. The train location is to be identified using the Global Positioning System (GPS) technology, and for this purpose a GPS module is proposed to be installed inside the train. Furthermore, the obtained train location using the installed GPS module inside the train is proposed to be transferred to a central system using the Global System for Mobile Communication (GSM) technology. Once the data of train's current location is received, the data is proposed to be processed using the custom software, and provide a visual positioning of train on maps using Geographical Information System (GIS) technology. In their study, they have mentioned that with the availability of this information, the administrative staff of Railway Department, like train controllers would be able to obtain more accurate details about train location and hence take more accurate decisions. At the same time, due to the availability of accurate, real time information including speeds of trains, the administrative staff is to be able to identify and address safety issues more effectively which occur in railway transportation system in a considerable frequency in Sri Lanka. Their study also shows that the collected data using the proposed system could be used for accurate scheduling considering the train arrival time and departure time at each station[13].

This system can be considered as a comprehensive solution for the current issues observed in the train transportation system. It is proposed to facilitate the real time train tracking, and to provide collected data to the railway administration to enhance the efficiency and safety of their service. But it mainly focuses on train administrative staff rather than the passengers, and also the cost of implementation and infrastructure cost will be considerably large. Furthermore, this system should be implemented within the railway department itself.

2.2.5 GPS based tracking system for trains in Sri Lanka[14]

Gunasekara, N.S has proposed a system named "trianTracker" to function as an auxiliary system inside the control center of Railway Department of Sri Lanka, for the use of internal technical staff. The system is proposed to monitor the train movements electronically using Global Positioning System (GPS) technology. In his study, he has pointed out the importance of having the exact location of a train, especially during disastrous situations. In the proposed "trianTracker" system, the locations of trains are to be displayed on a digital map, for the reference of staff inside the train control center in of Railway Department of Sri Lanka. For this purpose, the retrieved location of a train using Global Positioning System (GPS) technology, is to be transferred using the Short Message Service (SMS) service of the wireless telecommunications service provider[14].

The main disadvantage of the proposed system here is, it is available for the train control staff only. In this work, the train passengers have not been taken in to consideration. In contrast, the main objective of the CBTLS is to provide train location information to the general public.

2.2.6 Sri Lanka Railways - Future plans - Information Technology [15]

In the official web site of Sri Lanka Railways developed with the association of Information and Communication Technology Agency of Sri Lanka (ICTA), under the section of Future Plans and under the subsection of Information Technology, it has mentioned a planned project named "Train Tracking and Operating Information System". According to the information available in their official web site, Sri Lanka Railways is planning this project with the association of Information and Communication Technology Agency of Sri Lanka (ICTA), and the planned duration is mentioned as two (02) years. It has been proposed for the system to be installed at the railway control center at Maradana Train Station. As the technologies planned to use, it has mentioned the Global Positioning System (GPS) technology to get the location of the train and, Global System for Mobile Communication (GSM) to transfer the captured location and Speedometer Units to get the data on train speeds. The objective of the proposed system is mentioned as to enhance the functionality of train controlling and at the same time to provide exact train location to the general public[15].

This proposed system is a comprehensive solution for the issues identified in the current railway transportation system in this project. Since the system is implemented by Sri Lanka railways itself, the reliability and maintainability will be high and the conflicts that could arise when introducing such a new system would be much less, compared to an outside party doing the implementation.

The major concern regarding this proposed system is that still it is not implemented or available. According the official web site of Sri Lanka Railways, the last updated date of the site is mentioned as Tuesday, 20 September 2011, since the estimated time for the project is mentioned as two (02) years, the system should have implemented by now.

2.2.7 Different Types of Vehicle tracking systems

Benjamin Coifman, David Beymer, and coworkers have proposed a real time computer vision system for vehicle tracking and traffic surveillance on the basis of video image processing. Their work is focused on a feature-based tracking system for detecting vehicles under challenging conditions[16]. A similar study has done by D.J. Dailey and his coworkers to extract vehicular speed information from a given sequence of real-time traffic images. According to their study, the existing systems for similar purpose have problems with accurately tracking vehicles[17]. Therefore the purpose of Benjamin Coifman, David Beymer, and coworkers is to develop a system to address these problems. They have proposed to track vehicle features instead of tracking entire vehicles, making the system robust and the system less sensitive to the problem of partial occlusion. In their work, they have developed an algorithm and by tracking in daylight and nighttime conditions, the system itself will choose the most appropriate features for the given conditions. The resulting vehicle trajectories from this system can be used to provide traditional traffic parameters as well as new metrics such as lane changes. This vehicle tracking system is suited both for permanent surveillance installations and for short term traffic studies[16].

Noppadol Chadil and his coworkers, they have proposed an open source GPS tracking system named as Goo-Tracking system, using hardware and open source software [18]. It is a different approach than of system proposed by Benjamin Coifman, David Beymer, and coworkers based on image processing[16]. Their proposed system includes a Global Positioning System (GPS) module to locate vehicle and a General

Packet Radio Service (GPRS) for message transmission, Multi Media Card (MMC) to temporary store location information, and an 8-bit AVR microcontroller. Their system is claimed to have shown great stability when it was tested, and by using the robust message transfer protocol most of locations were accurately acquired and transmitted to the server in real-time. They have proposed the Goo-tracking system to be used in fleet management in the future, and as a further enhancement, they have proposed it to be used for lost vehicle tracking by integrating with a car alarm system. The sensors are to report vehicle status information to the server, which will be useful for information processing and for intelligent tracking management[18].

By comparing two different approaches on vehicle tracking, the second approach based on GPS/GPRS appears to be simpler and feasible to implement with minimum effort and cost.

The above study shows certain limitations of both approaches on vehicle tracking.

Research	Limitation
Real time computer vision system for vehicle tracking (Coifman et al., 1998)	One major limitation of this system is the cost related with implementing
GPS tracking system (Goo-Tracking system) (Chadil et al., 2008)	A limitation of this system in practical usage is, it requires a GPS module to be implemented inside the vehicle. Therefore this method will not be suitable in applications like public traffic management.

2.3 Summary

According to the review of other similar work done here, it is clear there are number of solutions already implemented and available, and also some comprehensive systems have been proposed to try and address the same set of issues the proposed system Community Based train locating System (CBTLS) is trying to address in this work. But, as discussed above, still there's not a single solution successfully implemented to address the current issues in Railway Service Sri Lanka. As indicated above, each one

of them are having its own unique features which is beneficial for the train passengers as well as train controlling staff. But the lack of one comprehensive system, including all good features of all above systems is still a pending requirement.

The many mobile application which are available in Google Play, are based on same service offered by Information and Communication Technology Agency of Sri Lanka (ICTA) and Sri Lanka Railways, therefore all of them share the same issue, not having real time updated data.

CBTLS is addressing this issue by keeping data from ICTA and Sri Lanka Railways as master data, and get updated on them through the community. By maintaining its own data store, CBTLS is capable of handling the issues which other systems described above are not able to handle.

In contrast to the systems mentioned above, the implementation cost would be minimum for CBTLS.

The following chapter – Chapter 3 will describe the tools and technologies used to implement the CBTLS.

CBTLS – chosen technologies to cater real time data

3.1 Introduction

In the previous chapter, various existing systems and proposed systems to address the same issues which CBTLS is supposed to solve were analyzed. Each of their features and disadvantages were listed.

In this chapter, the technologies, and architectural features will be described regarding the proposed community based train locating system. CBTLS should be designed in a way to allow maximum possible number of users to access the system. The system will receive data from a large number of users simultaneously, and therefore should be capable of handling such large amounts of requests through the web clients as well as the mobile clients.

When considering the mobile application, the system requires internet access to operate its functionality properly, but at least the cached static data of recent searched should be available in the mobile device as well. Therefore a temporary database would be stored in mobile device as well. A more detailed description of such concerns and technologies adapted to address them are described in this chapter.

3.2 Technologies Available

3.2.1 Web application (User interface and Backend Service)

When considering the proposed web application, there are number of technologies available to which could be adapted in to the proposed system. There are several commonly used and popular techniques such as PHP, ASP.net, Java EE used. For this project Java EE has been selected. Considering the requirements, ability to cater a large amount of users simultaneously, the scalability of the system is crucial, at the same time with the solid Object Oriented Programming nature of Java, it facilitates the modeling of real world object into the system more conveniently. Also there are many web based frameworks available for Java, facilitating rapid developments once properly configured. Since Java is open source and freely available it does not contribute to the cost of software. At the same time powerful and widely used open source IDEs (Integrated Development Environments) like Eclipse are available freely for

development. These factors contribute to the cost of implementation and therefore open source software were selected in every scenario.

The backend service was developed as a REST (Representational State Transfer) service, this is because both mobile and web applications shares the same functionality. For the user interface development of the web application, JQuery, Bootstrap and CSS was used. The purpose of using bootstrap is to provide a similar look and feel across most browsers available. The purpose of using JQuery for Ajax calls and other normal JavaScript functions as well is to support similar functionality across all the main browsers available.

In order to support rapid development by maintain the coding standards like design patterns, Spring framework and Hibernate framework has been used. In Spring framework, the available features, Spring security, Spring MVC (Model View Controller), Spring web module, Spring tag library, Spring support for hibernate, Spring REST services and Spring core container has been selected to be used in CBTLS.

This application requires a user authentication module, and since Spring provides an easily configurable, yet very powerful security module, it has been used in the system.

Since CBTLS consists of two parts, a mobile application and a web based application, it necessarily requires the separation of Model – View – Controller components. Due to this requirement, the MVC module of Spring framework was used in CBTLS.

For the web application, the Web module of Spring and the Spring tag library was required. The spring web module was required for the rest services as well. In addition to the Spring tag library, the JSTL tags, and FMT (formatting tags) were also used for common functionalities like to iterate a collection inside JSP(Java Server Pages), to format date like functionalities.

In this system, Hibernate framework has been integrated as an ORM (Object Relational Mapping) tool to facilitate the easy mapping as access of RDBMS (Relational database management systems) tables to the domain objects available in java. Spring hibernate support module is added to support the integration of spring framework with java.

Spring REST services has been used in the system to facilitate the system integration with the mobile application, through the provided REST (Representational State Transfer) API (Application Program Interface), the mobile application communicate

with the web application. The spring core container has been integrated to handle the dependency injection and inversion of control of beans in the system.

MySQL was used as the relational database for the system, it is free and open source software with a considerable community support. It caters the most crucial features like scalability and flexibility. Open source tools like MySQL workbench is available to access and view data in MySQL conveniently. It also provides features like reverse engineering of databases. MySQL databases supports complete ACID (atomic, consistent, isolated, durable) robust transactions. MySQL databases can be easily clustered and hence support large scale systems. The convenient integration support available with java of MySQL was another main reason to be selected as the database for CBTLS.

3.2.2 Mobile Application

When deciding on the technology to be used to develop mobile application, the main factor which was taken in to consideration was the current usage percentage of the technology and the future trends. There are few popular mobile application platforms in the world at present including Android, Apple IPhone, Windows mobile and Blackberry. Of these, in general in Sri Lanka and also in other countries, Android has a considerable market share compared to other platforms. The main purpose of CBTLS is to facilitate as much as possible train passengers to use the system, and therefore the widely available mobile platform was selected. Additionally, it provides strong built in functionalities and as well as has larger community support. Development using android is relatively low cost compared to platforms like IPhone, which requires their specific Operating System and tools for development. The development tools for android (ADT – Android development Tool built upon Eclipse IDE) is freely available to download.

3.3 Design Considerations

Following design considerations were made during the design process of CBTLS User Interface Design

3.3.1.1 Usage of MVC pattern

As described above, for the web application - Model View Controller (MVC) Pattern was used with the aid of Spring MVC. Since both mobile and web client should be catered using same services, the presentation layer is separated from the business layer.

3.3.1.2 Integrating localization to the system

Since one of CBTLS systems main objective is to address the difficulties faced by train passengers due to linguistic issues, both mobile and web applications are developed in a way to support localization. Both Spring and Android platforms are supporting this feature by default.

3.3.1.3 Supporting major browsers available

Since CBTLS is a community based application, it supposed to be accessed by a large variety of users, and therefore it is a main requirement for the system to support the main browsers and their latest versions. For this purpose, a user interface framework like bootstrap, and JavaScript framework like JQuery is used.

3.3.1.4 Variety of Mobile device support

Again, for the purpose of catering large number of users, the native android mobile application would support earliest possible android version, in order to cater as much as devices.

3.3.1 Programming considerations

When developing the system, in every possible location, design patterns would be used. Most of this mentioned design patterns are already available with the integration of spring and hibernate frameworks. The Business component uses Data access Object (DAO) pattern, Factory Pattern, Singleton Pattern. The presentation layer uses MVC pattern. The purpose of using design patterns is to increase the reusability and maintainability at the code level.

3.3.2 Database Design Considerations

3.3.2.1 Data model

The physical data model is done considering MySQL database. The selection RDBMS (Relational Database management system) is done due to the requirement of maintaining the relationships of data for the easy analysis later.

3.3.2.2 Connection Pooling

A Connection pool is used for obtaining database connections & those connections are released back to the pool after usage. This process is handled automatically with the introduction of Hibernate as Object Relational Mapping and data access framework to

the system..This is to cater the simultaneous usage of the system by large number of users.

3.3.2.3 Database Transaction and rollback handling

In order to maintain the ACID (atomic, consistent, isolated, durable) properties of the database transactions, the transaction layer has been moved to the service layer of the application. It ensures no partially committed transactions are available.

3.3.2.4 Support and Facilitating Concurrent access of database

Since data in CBTLS are be concurrently accesses in a very high frequency. Therefore to handle concurrent updates and to avoid data being overwritten from other sessions optimistic locking of database records through the version column will be used.

3.3.2.5 Clustering support for scaling up

The application is designed to support clustering, to facilitate scaling up when required. To facilitate this the business classes are made to be stateless and singleton in nature. Therefore the application supports single instance or cluster environment deployment.

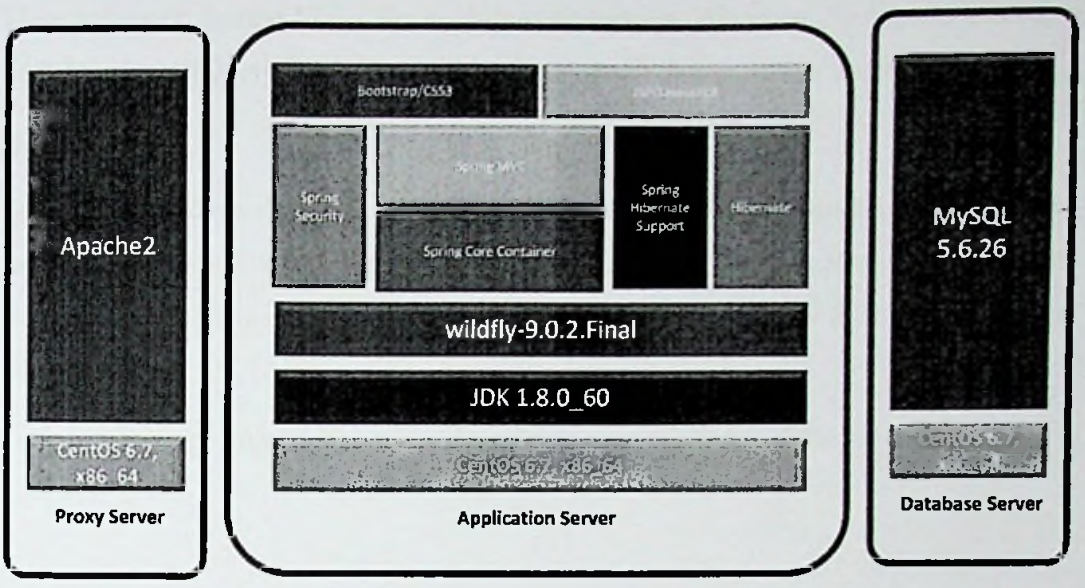
3.3.3 Logging Facilities for debugging

Log4j logging framework will be integrated to the system to capture system logs into file system for any kind of operation in the system. This feature is required to fix the bugs raising after the systems launch.

3.3.4 Security

The web application is secured with Spring security framework and only authorized users will be able to access the system, and even the authorized users will have restrictions when it comes to system's functionality.

3.4 Technology Stack



CBTLS Technology Stack

Figure 3.2 – Technology Stack

Component	Description
Proxy Server	The physical server dedicated to handle requests from client and do load balancing, which is crucial in systems like CBTLS
CentOS 6.7,x86_64	64 bit CentOS- a Linux operating system as the OS in servers
Apache2	To be used as the proxy server
Application Server	The dedicated physical server/servers to handle application server - WildFly
JDK 1.8.0_60	Java framework version 8, on top which CBTLS is implemented and will execute,
Wildfly Application Server	Application server which hosts the CBTLS Web application

Table 3.2 – Each component of Technology Stack of CBTLS

Usage of the other components available in the above diagram - Spring security, Spring Core container, Spring MVC, Spring hibernate support, Hibernate, Bootstrap/CSS and JSP/Javascript/jQuery, MySQL are describe in detail earlier in this section.

3.5 Summary

In this chapter the technologies used in CBTLS and main factors which were taken into consideration when designing the system were described. Furthermore, along with the selected technologies in the implemented system, the reasons for their selection was also described.

In the following chapter, the approach to implement CBTLS will be described.

Crowdsourced system approach for real time train information

4.1 Introduction

In the previous chapter the design considerations and technologies selected to be used in implementation of CBTLS was described.

In this chapter, the approach taken to solve the mentioned issues in chapter 1 and chapter 2 is described in terms of inputs to the CBTLS, expected outputs, processes available inside the system, additional features and also the targeted set of users of the system.

4.2 System Structure of CBLS

The proposed system would consist of a web application and a mobile application. Mobile application would be used to collect data about trains from passengers and the same is used to display data upon enquiries. Same functionality is available in the web application as well, and additionally administrative functionality. The mobile application would be a native, location-aware application for Android which would support geo locating the user. Therefore this mobile system would only be available for android users. Since the web application consists of all the functionality of mobile application, rest of users can access the web application if required.

4.3 Inputs for the Community Based Train location System

Initial data of selected train schedules would be fed to system using the data integration module. This static schedule data is retrieved from the web service available from ICTA and Department of railways. Once the master data on train schedules is available, users would be able to look up schedules initially. The proposed system is mainly based on data provided by general public (the community of train passengers), on each occurrence of train schedule. Therefore the critical data required for system's functionality is captured from rain passengers who would choose to use the system. Therefore, a challenging part of the system would be to validate the received data before it gets displayed for other users.

For this validation purposes, and for data analytical purposes, geo coordinates of train stations along the selected route, and the geo coordinates of the selected rail route would be have to be inserted into the system along with master data. Since such data is not already available, it should be done manually using the features available in Google Map API. Unlike for a normal route, for rail roads in Sri Lanka, the series of geo coordinates is not available.

The users of CBTLS can update the current location of a selected train using the mobile application or web application. This could be done actively or passively, and for each two methods different parameters will be taken in to the system as inputs. Active update is available for the users who are already inside the train, they could either update the location once or can allow the system to keep track of the location continuously. Here the location of the user would be captured. Passive update is for the users who are outside the train, but still aware of the location of the train. They are allowed only to update once, and when updating, instead of their lactation data, the last station passed, the located time and current moving status of the train should be provided.

Furthermore, users can update the compartment details of the train as well, and this is in terms of crowd density. They can provide one of the predefined crowd density status for a selected compartment and for the overall train. They should provide the compartment number of their reporting and the total number of compartments in the whole train as well.

As an additional feature, a location aware alarm clock is integrated and users can set the alarm based on their preference.

4.4 Outputs

Initially, similar to the currently available systems which were reviewed in chapter 2, users will only have access to view the static train schedules as provided by the web service offered by ICTA and Department of railways. Once the system is updated by train passengers, the real time data would be available for the general public.

The CBTLS facilitate users to view real time train locations on a map, and also allows to view compartment details of a selected train. Additionally, it provides the facility to view analysis and predictions on a selected train schedule.

If the location aware alarm is set by the user, it will be activated once the set destination has been arrived.

4.5 Process

The initial static train schedule data is integrated in to the system using the data integration module. By using either mobile or web application, users could search for train schedules. For this purpose, a basic search and an advanced search both will be available for the convenience of users, once they view the train schedules, they could access the list of recommendations for the same criteria. This recommended list is prepared by analyzing the historical data collected in the system.

The user authentication module will authenticate users by the backend service, allowing to use same credentials to be used both in web and mobile applications.

In the mobile system, the user location would be acquired through GPS and Android's Network Location Provider and is sent to the web application as a series of geo coordinates. The retrieved location data into the web backend is to be validated against a set of predefined geo location data set before being recorded in the system. As an additional measure, the data will be validated against user's ranking in the system as well.

4.6 Users

Three major types of users are identified in the CBTLS system as anonymous users, registered train passengers (normal system users) and system administrators. Based on the type of user, access levels to certain functionalities of the system is varying. Only the system administrators are allowed to view and use the administrator module, and only the registered users are allowed to update the system with data. Anonymous users are allowed to use the viewing functionalities only, this measure has been taken ensure the reliability of the system.

4.7 Features

The most distinguished feature of CBTLS among the reset of services available or proposed for the same purpose is its source of information that is the community based nature of the system. Due to this factor, the implementation cost is kept at a minimal rate compared to the rest of systems using GPS/GPRS like technologies. Since no involvement from railway Department is required, the implementation would not be complicated. Once the web system is hosted and mobile application is added to the Google Play app store, general public can easily access and use the system. Only an initial cost for the hosting environment is to be applied. Over the time CBTLS is

expected to be grown mature, since large amount of valuable data will be collected through the system, which is not already available. Such data could be used for the analytical purpose and to generate new knowledge.

Although CBTLS contains static train schedule data, unlike the existing systems which are calling the remote service each time a user access the system, CBTLS would be consists of its own data set after the initial integration. This is to prevent the dependency on a third party service, specially a service which is not reliable. For the sake of accuracy and updated data, the integration process could be done periodically. Therefore CBTLS is expected to be a self-managing system without depending on any of external systems.

4.8 Summary

In this chapter the community based train locating system has been described in terms of its inputs and outputs, users who are involved and the system's specific features and processes.

With the detailed description of the system's key aspects in this chapter, in the next chapter analysis and design part of CBTLS would be described along with diagrams to aid. The design diagrams would be incorporated in to this document at the appendix.

Chapter 5

Analysis and Design of CBTLS

5.1 Introduction

In the previous chapter, the approach to develop CBTLS was described in terms of its key components, expected inputs to the system, expected outcomes, planned processes and features of the system.

In this chapter detailed design of CBTLS would be illustrated. For the design purposes, the UI wireframes of mobile application, the Entity relationship diagram of CBTLS, the class diagram of CBTLS, sequence diagrams related with mobile application functionalities are illustrated below.

5.2 Detailed Architecture Diagram Of CBTLS

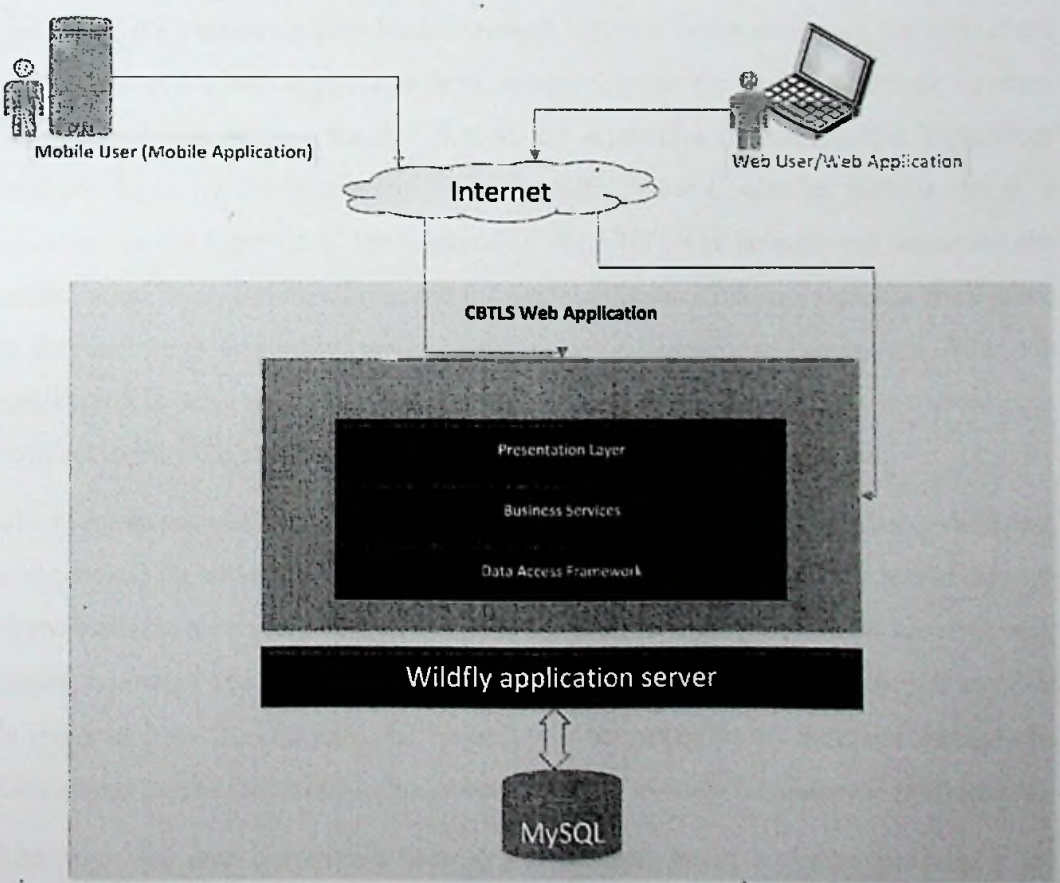


Figure 5.1 – Overall Architecture of CBTLS

Component	Description
CBTLS Web application	Used by the general public through web clients and as well as mobile clients
Wildfly Application Server	Application server which hosts the CBTLS Web application
MySQL	Relational database to store CBTLS data
Mobile User/Mobile Application	Native android application which would communicate with the web application
Web user/Web application	Web application as the client itself

Table 5.1 – Each component of Overall Architecture of CBTLS in Figure 5.1

As it is explained in the chapters 1 and 3, the CBTLS consist of a mobile application and a web application. The web application caters as the backend for the mobile application as well. As it is indicated in the Figure 5.1, the mobile application will connect to the backend system hosted through internet. In the same way, the web client component of the web application is to be connected to the backend through internet. This feature has become feasible due to the separation of presentation logic from business logic by the implementation of Model View Controller pattern. As it is indicated in the Figure 5.1, the backend of the CBTLS is layered and separated the presentation layer, business layer and the database layer. This approach has been taken to facilitate high flexibility, maintainability, and scalability of the system. The web application is designed to be hosted in wildly application server, and the system is to connect to MySQL database through data access layer of the application.

The request received by the presentation layer of the web based application is validated, and checked for authentication and based on the validity of request, it is passed through to the business layer. Business layer of the application would contain the core business logics to process and respond to the received request, and for this processing if any data is required from the database, the request will be passed to the database through the data access layer of the application, where it would manage the database transactions.

The requested data is returned back to the business layer, and after processing the received request is responded with the processed information through the presentation layer. According to the design of CBTLS, the business layer and the data access layer

would be stateless, and the user session related data would be maintained at the presentation layer of the system, hence it would be stateful.

As it was mentioned in chapter 1, the core functionalities of the system would be exposed to the mobile and web application through a REST (Representational State Transfer) API (Application Program Interface).

5.3 User Interface Wireframes of mobile application

Please refer Appendix A for the User Interface design diagrams of the mobile application of CBTLS. In the section, each user interface is explained in detail related to its functionality and the navigation from one user interface to another.

5.4 CBTLS entity relationship diagram

Please refer Appendix B for the entity relationship diagram (ER Diagram) of CBTLS.

5.5 Class diagram of CBTLS

Please refer Appendix C for the class diagram of Community based train locating system.

5.6 Sequence Diagrams of CBTLS

Please refer Appendix D for the class diagram of Community based train locating system.

5.7 Summary

This chapter was mainly focused on design diagrams and design details of CBTLS. Here the architectural design of the system was explained in details along with reasons for the design. Also, the user interface design of the mobile application (this is applicable to web application as well) were explained in detail with the navigation process and purpose of each interface design. Furthermore, rest of the design diagrams of the system like class diagram, entity relationship diagram, and sequence diagrams of CBTLS is listed in this section.

The following chapter is mainly focused on implementation details of CBTLS, it contains the curtail code segments,

Chapter 6

Implementing CBTLS for real time information

6.1 Introduction

In the above chapter, chapter 5, the design details of CBTLS was described in detail. In this chapter, it will be explained the details of implementation done based on that design.

Here the implementation details would be discussed in terms of software, hardware used for the implementation, important algorithms and code segments used for the implementation of CBTLS.

6.2 Implementation Plan for demonstration purpose

For the demonstration purpose of research, only a single train route (Colombo –Puttlam line) was selected, this is in terms of integrating master data including static train schedules, to geo coordinates of the train route. Of that train route, train schedules are selected to cover both weekdays and weekends, for office times where the trains are mostly crowded, and to cover regularly crowded times, when generating data to stimulate data that should be retrieved from the train passengers.

6.3 Software and Hardware used in CBTLS implementation

At the time of implementation a set of tools were used for the development purposes as described in this section. To maintain the code base for both mobile and web application, the decentralized version controlling system – GitHub was used. As the git client for windows "TortoiseGit 2.0.0.0" software was used together with "git version 2.6.2" for windows.

For the web application, as the integrated development environment (IDE), Eclipse "Mars Release (4.5.0)" was used. This is because. Since CBTLS is developed using Java "jdk1.8.0_60", of Eclipse IDEs available, the latest version Eclipse Mars is the only Eclipse IDE which fully supports java 8. Eclipse IDE was selected for

development as it is a comprehensive development tools with a large number of plugins available and a widely used IDE.

For the mobile application the Android Developer Tools (ADT) built upon Eclipse was used. For the application development, the android version 5.0 was used (Lollipop - API level 21). Even though a latest version of android was used for development, it facilitates the backward compatibility for android devices.

Since MySQL database was used in CBTLS, for the development, MySQL Community Server - version: 5.5.28 was used. To access this server via a graphical user interface, MySQL workbench 5.2.47 was used.

To run and test the web application, wildfly "9.0.2.Final" version was used. The application was run and tested using Firefox web browser "41.0.2" version.

To run and test the mobile application, an android device with android version 5.1 (Lollipop OS) was used.

At the development stage, the application server and database server both were in the same computer, but at the actual deployment they could be easily separated as described in the session below named "Deployment View". Since the application was hosted in the local environment (localhost), for the mobile device to access the REST web service, both devices were connected to a same router in a local area network.

6.4 CBTLS web application implementation

Initially the project is set up using the technologies explained in the chapter 3. The project setup inside the eclipse is viewed as below in figure 6.1.

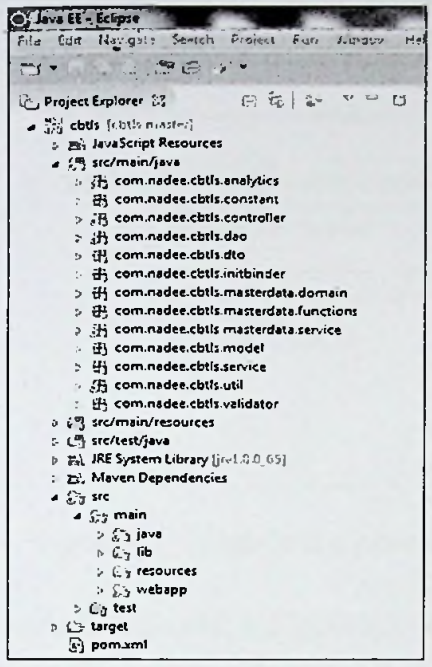


Figure 6.1 – CBTLS Web Application Structure in Eclipse

The static schedule data was integrated into the system initially using the web service provided by Railway Department and ICTA, the web service URL is given below.

"http://103.11.35.13:9080/services/RailwayWebServiceV2Proxy.RailwayWebServiceV2ProxyHttpSoap12Endpoint"

In order to call this web service, the following method was used,

```
private static SoapObject callWebService(String actionName, SoapObject request) throws Exception {
    SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
    envelope.setOutputSoapObject(request);
    AndroidHttpTransport androidHttpTransport = new AndroidHttpTransport(ENDPOINT);
    androidHttpTransport.call(actionName, envelope);
    System.out.println("result" + envelope.bodyOut.toString());
    SoapObject results = null;
    try {
        results = (SoapObject) envelope.bodyIn;
    } catch (ClassCastException e) {
        SoapFault fault = (SoapFault) envelope.bodyIn;
        throw e;
    }
    return results;
}
```

Figure 6.2 – CBTLS Web Application method to call web service

After the initial master data is included in the system, users can search for train schedules. As an example code to represent rest of the services as well, in below Figure 6.3, Figure 6.4 and Figure 6.5, the code samples to represent the flow inside the system from presentation layer (Figure 6.3) to business layer (Figure 6.4) to the persistent layer (Figure 6.5) is indicated.

```
@Controller
public class TrainStationScheduleController {

    @Autowired
    private TrainStationScheduleService trainStationScheduleService;

    @RequestMapping(value = "/searchTrainSchedules", method = RequestMethod.POST)
    public @ResponseBody List<TrainStationScheduleDTO> searchTrainSchedules(
            @RequestBody TrainScheduleSearchDTO trainScheduleSearchDTO ){
        List<TrainStationScheduleDTO> list=new ArrayList<TrainStationScheduleDTO>();
        try {
            System.out.println("trainScheduleSearchDTO :" + trainScheduleSearchDTO);
            list=trainStationScheduleService.serachTrainStationSchedules(trainScheduleSearchDTO);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return list;
    }

}
```

Figure 6.3 – Code in the presentation layer (controller) to receive the request

As it is indicated in Figure 6.3, with the use of annotation "@ResponseBody" this controller will function as a rest service provider.

```
@Service("trainStationScheduleService")
@Transactional(propagation = Propagation.SUPPORTS, readOnly = true)
public class TrainStationScheduleServiceImpl implements TrainStationScheduleService {

    @Autowired
    private TrainStationScheduleDAO trainStationScheduleDAO;

    @Autowired
    private CommonDAO commonDAO;

    @Override
    public List<TrainStationScheduleDTO> serachTrainStationSchedules(TrainScheduleSearchDTO trainScheduleSearchDTO)
            throws Exception {
        List<TrainStationSchedule> stationSchedules=trainStationScheduleDAO.serachTrainStationSchedules(trainScheduleSearchDTO);
        if(!(stationSchedules==null)){
            List<TrainStationScheduleDTO> list=new ArrayList<TrainStationScheduleDTO>();
            for (TrainStationSchedule trainStationSchedule : stationSchedules) {

                TrainStationScheduleDTO trainStationScheduleDTO=new TrainStationScheduleDTO(trainStationSchedule);
                trainStationScheduleDTO.setTicketPrices(new ArrayList<TicketPriceDTO>());
                List<TicketPrice> ticketPrices=trainStationScheduleDAO.getTicketPrices(trainStationSchedule.getTrainStationScheduleId());
                for (TicketPrice ticketPrice : ticketPrices) {
                    trainStationScheduleDTO.getTicketPrices().add(new TicketPriceDTO(ticketPrice));
                }
                list.add(trainStationScheduleDTO);
            }
            return list;
        }
        return null;
    }
}
```

Figure 6.4 – Code in the business layer (service) to process request

As it has been indicated in Figure 6.4, after service layer receives request from controller as indicated in Figure 6.3, it will access the persistent layer indicated as "TrainStationScheduleDAO" and "CommonDAO" in Figure 6.4, get the requested information, process them and pass back to the controller layer, so it could give response back to the web client or the mobile client. This flow has been clearly described in chapter 4 and 5. With use of "@Transactional" annotation, the database transaction management is taken in to the service layer.

```
@Repository(value="trainStationScheduleDAO")
public class TrainStationScheduleDAOImpl implements TrainStationScheduleDAO {

    @Autowired
    private SessionFactory sessionFactory;

    @Override
    public List<TrainStationSchedule> serachTrainStationSchedules(TrainScheduleSearchDTO trainScheduleSearchDTO)
            throws Exception {
        Criteria criteria = sessionFactory.getCurrentSession().createCriteria(TrainStationSchedule.class);
        criteria.createAlias("trainSchedule", "trainSchedule");
        criteria.createAlias("fromTrainStation", "fromTrainStation");
        criteria.createAlias("toTrainStation", "toTrainStation");

        Conjunction and=Restrictions.conjunction();
        and.add(Restrictions.ge("arrivalTime",trainScheduleSearchDTO.retrieveFromTime()));
        and.add(Restrictions.le("departureTime",trainScheduleSearchDTO.retrieveToTime()));
        and.add(Restrictions.eq("fromTrainStation.trainStationId",trainScheduleSearchDTO.getFromStationId()));
        and.add(Restrictions.eq("toTrainStation.trainStationId",trainScheduleSearchDTO.getToStationId()));
        criteria.add(and);

        List<TrainFrequency> list=trainScheduleSearchDTO.retrieveTrainFrequencies();
        if(!(list==null || list.isEmpty())){
            Disjunction or = Restrictions.disjunction();
            for (TrainFrequency trainFrequency : list) {
                or.add(Restrictions.eq("trainSchedule.trainFrequency",trainFrequency));
            }
            criteria.add(or);
        }
        return criteria.list();
    }
```

Figure 6.5 – Code in the persistent layer (service) to process request

Here the persistent layer is accessing database with the aid of Hibernate framework and fetch data based on defined criteria and returns the requested data back into the service layer. When considering the structure of "TrainScheduleSearchDTO" as indicated in above figures (Figure 6.3, Figure 6.4 and Figure 6.5), it is a Data transfer Object (DTO) used for the sole purpose of passing data through a layered architecture. The structure of the "TrainScheduleSearchDTO" is shown is below, Figure 6.6.

```
public class TrainScheduleSearchDTO implements Serializable {
    private static final long serialVersionUID = 1L;

    private long fromStationId;
    private long toStationId;
    private String searchedDate; // in dd/MM/yyyy
    private String fromTime; // in HH:mm
    private String toTime; // in HH:mm

    public Date retieveFromTime() {
        Date fromTimeDate = null;
        if (StringUtils.isNotEmpty(fromTime)) {
            SimpleDateFormat tf = new SimpleDateFormat("HH:mm:ss");
            try {
                fromTimeDate = tf.parse(fromTime);
            } catch (ParseException e) {
                e.printStackTrace();
            }
        }
        return fromTimeDate;
    }

    public Date retieveToTime() {
        Date toTimeDate = null;
        if (StringUtils.isNotEmpty(toTime)) {
            SimpleDateFormat tf = new SimpleDateFormat("HH:mm:ss");
            try {
                toTimeDate = tf.parse(toTime);
            } catch (ParseException e) {
                e.printStackTrace();
            }
        }
        return toTimeDate;
    }
```

Figure 6.6 – Structure of the data transfer object

The flow of the rest of the functionalities in the system would be identical to the structure shown in above figures (Figure 6.3, Figure 6.4 and Figure 6.5 and Figure 6.6).

The domain classes of CBTLS serves as the backbone of the system, all the logics, code structuring is done based on them, and with the use of Hibernate framework, the table structure is also auto generated from the defined domain classes with the use of annotations.

Please refer Appendix E for the structure of domain classes of Community based train locating system.

One of the crucial feature of CBTLS is to maintain the validity of its information, since data is retrieved from general public. For this purpose, the geo coordinates received from a user is validated in the system as given in below Figure 6.27,

```
@Component("gPSValidator")
public class GPSValidator {

    //within 50Km
    private static final double standarRadius=0.5;

    boolean isInRectangle(double centerX, double centerY, double radius, double x, double y) {
        return x >= centerX - radius && x <= centerX + radius && y >= centerY - radius && y <= centerY + radius;
    }

    // test if coordinate (x, y) is within a radius from coordinate (center_x,
    // center_y)
    public boolean checkIfPointInCircle(double centerCoordX, double centerCoordY, double radius, double coordX, double coordY) {
        if (isInRectangle(centerCoordX, centerCoordY, radius, coordX, coordY)) {
            double doubleXCoord = centerCoordX - coordX;
            double doubleYCoord = centerCoordY - coordY;
            doubleXCoord *= doubleXCoord;
            doubleYCoord *= doubleYCoord;
            double distanceSquared = doubleXCoord + doubleYCoord;
            double radiusSquared = radius * radius;
            return distanceSquared <= radiusSquared;
        }
        return false;
    }

    public boolean checkIfValidLocationInput(GeoLocation inputLocation, List<GeoLocation> geoLocations){
        boolean isValid=false;
        for (GeoLocation centerLocation : geoLocations) {
            if((checkIfPointInCircle(centerLocation.getLatitude(), centerLocation.getLongitude(),
                    standarRadius, inputLocation.getLatitude(), inputLocation.getLongitude()))){
                isValid=true;
            }
        }
        return isValid;
```

Figure 6.27 – Validation code for GPS coordinates

Since CBTLS allows the users to be ranked, that parameter is used to validate users as given blow in Figure 6.28.

```
//threashhold user ranking
private static final float minUserRanking=2.5f;
/*
 *  The formula for calculating the weighted rating of a user, Bayesian estimate:
    weighted rating (WR) = (v Ã· (v+m)) Ã— R + (m Ã· (v+m)) Ã— C
    where:
    * R = average for the user (mean) = (Rating)
    * v = number of votes for the user = (feedbacks)
    * m = minimum votes required to be considered as a trusted user - eg:- 10
    * C = the mean vote across the whole report
    for the Top 250, only votes from regular voters are considered.
 */
public boolean checkIfUserIsTrusted(long userId){
    float avergeUserRanking=systemUserRankingService.getAverageSystemUserRanking(userId);
    if(avergeUserRanking>=minUserRanking){
        return true;
    }
    return false;
}
```

Figure 6.28 – Validation code for Users

6.5 CBTLS mobile application implementation

The project was set up using the technologies explained in the chapter 3. The project setup inside the eclipse is viewed as below in figure 6.29

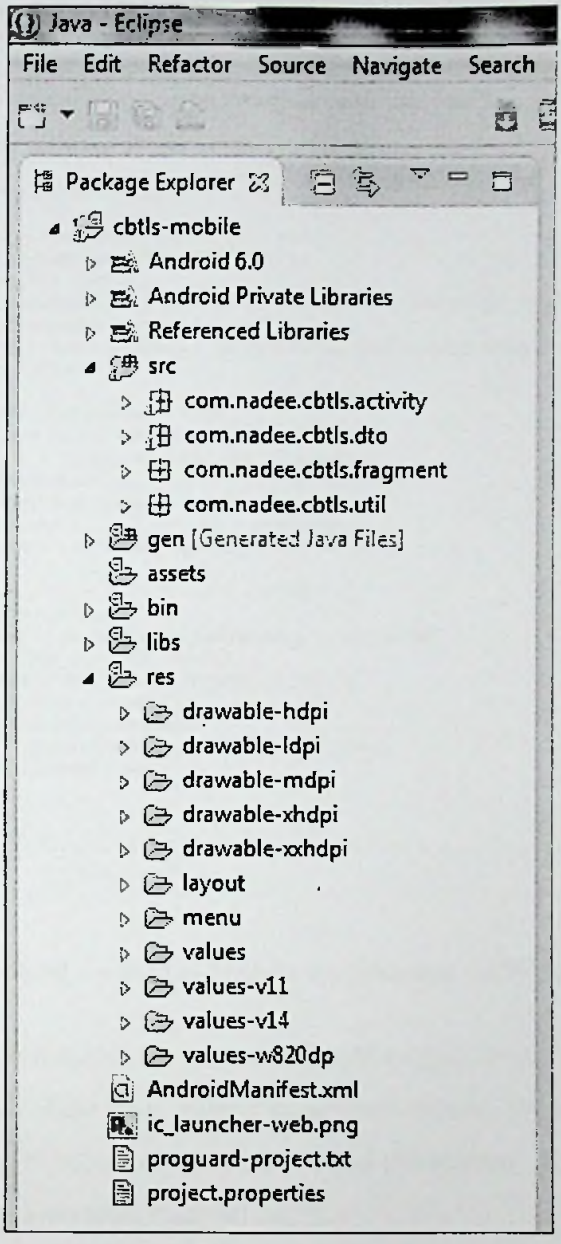


Figure 6.29 – CBTLS Mobile Application Structure in Eclipse ADT

The user location would be acquired through GPS and Android's Network Location Provider. Although GPS is most accurate, it only works outdoors, it quickly consumes battery power, and doesn't return the location as quickly as desired. Android's Network Location Provider determines user location using cell tower and Wi-Fi signals, providing location information in a way that works indoors and outdoors, responds faster, and uses less battery power. Therefore, to obtain the user location in this application, both GPS and the Network Location Provider are used as shown in figure 6.30.

```
//Minimum distance to change updates in meters
private static final long MIN_DISTANCE_CHANGE_TO_UPDATE = 10; // 10 meters

//Minimum time between updates in milliseconds
private static final long MIN_TIME_BETWEENW_UPDATES = 1000 * 60 * 1; // 1 minute
/**
 * Try to get my current location by GPS or Network Provider
 */
public void getCurrentLocationOfUser() {
    try {
        locationManager = (LocationManager) mContext
                .getSystemService(LOCATION_SERVICE);
        // check and get GPS status
        isGPSEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);
        // check and get network status
        isNetworkEnabled = locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER);
        // Try to get location
        if (isGPSEnabled) {
            this.isGPSTrackingEnabled = true;
            // GPS is used to get GPS coordinates
            provider_info = LocationManager.GPS_PROVIDER;
        } else if (isNetworkEnabled) { // Try to get location
            this.isGPSTrackingEnabled = true;
            // Network State is used get GPS coordinates
            provider_info = LocationManager.NETWORK_PROVIDER;
        }
        // Application can use GPS or Network Provider
        if (!provider_info.isEmpty()) {
            locationManager.requestLocationUpdates(provider_info,
                    MIN_TIME_BETWEENW_UPDATES,
                    MIN_DISTANCE_CHANGE_TO_UPDATE, this);
            if (locationManager != null) {
                location = locationManager
                        .getLastKnownLocation(provider_info);
                updateGPSCoordinates();
            }
        }
    } catch (Exception e) {
```

Figure 6.30 – CBTLS Mobile Application GPS Tracker

The web application contain all the features of mobile application, except for the facility to provide the location of the train or the compartment details. The general public would be able to search for trains, get the estimation and predictions. At the same time they could provide feedback on each selected trains.

In the web application, there would be a part with restricted access for admin functionalities.

Please refer Appendix F for user interfaces available in the web application.

6.6 Deployment View

The following diagram illustrates the deployment environment for the CBTLS, required resources and networks.

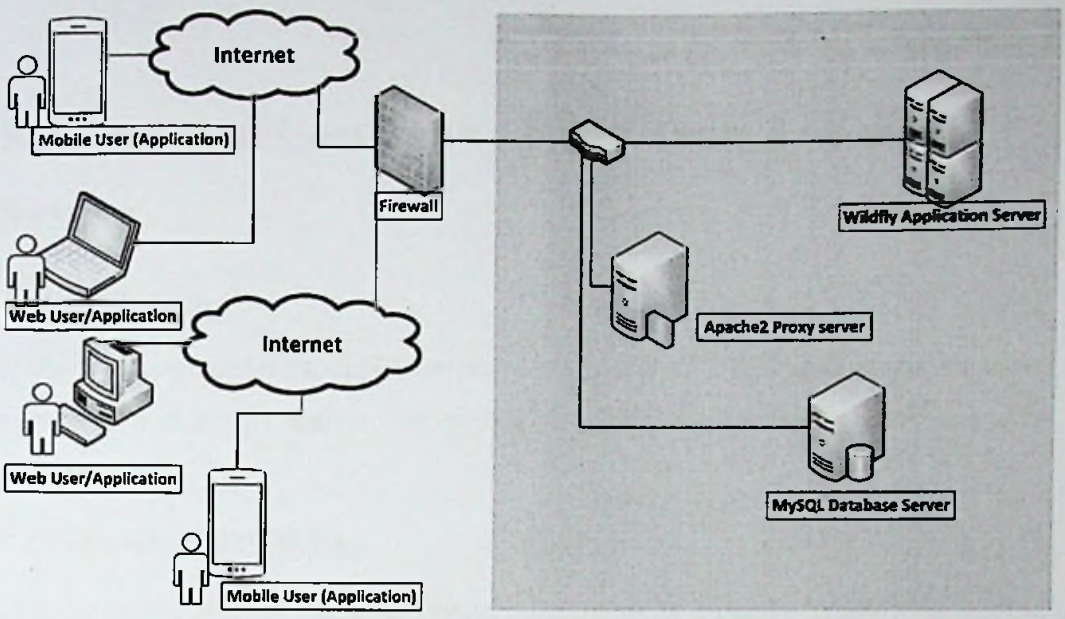


Figure 6.49 – CBTLS Web Application Deployment Diagram

Above figure depicts the deployment architecture of the servers and the services required for the CBTLS Application. As it has been indicated in Figure 6.1, Apache 2 server is integrated in to the deployment structure for the load balancing purposes.

As it was stated in chapter 3 as well, the web based system is hosted in Wildfly application server, based on the requirements and growth of the system, this application server could be clustered and load balancing techniques could be introduced.

Since MySQL database is used for the implementation of CBTLS, it will be hosted in a separate server in the production environment and that server could be separately clustered or replicated based on the requirements. In Figure 6.1 this MySQL server has been labelled as "MySQL" database server.

6.7 Summary

In this chapter, it was explained in detail how the implementation done for CBTLS, based on the design which was describe in Chapter 5. In next chapter it will be discussed how this implemented system was evaluated by a selected set of users.

Chapter 7

Evaluation of Community Based Train Locating System

7.1 Introduction

In the previous chapter the implementation details of the CBTLS system was discussed in details, and in this chapter, the evaluation of the implemented system would be discussed.

7.2 Evaluation Methodology

In order to evaluate the complete system's functionality properly by allowing a set of users, it would require the system to be deployed in an actual production environment. Therefore, the methodology used here is to allow a set of selected users to use and give a feedback on system using a predefined set of question in a questionnaire. For this evaluation purpose, the system was deployed in a local environment. Since CBTLS consists of a mobile and a web based application, both systems were evaluated separately, using the same set of questions.

The following areas were evaluated in the system evaluation.

- Web Application Evaluation
    - In terms of system functionality (5 questions)
    - In terms of Usability (5 questions)
    - In terms of Overall Impression (5 questions)
- Mobile Application Evaluation
    - In terms of system functionality (5 questions)
    - In terms of Usability (5 questions)
    - In terms of Overall Impression (5 questions)

For each question on system evaluation sections, following evaluation factors were assigned.

1 - Very poor (1), 2 – Poor (3), 3 – Average (6), 4 - Good (8), 5 - Excellent (10)

The target of the evaluation is to make a decision through a statistical analytical method. A critical line is defined and the calculated average pint for each question will be checked against the critical line to make a decision.

Number of users who has given the feedback - U

Number of questions on evaluation feature – Qi

Number of items marked as Very poor – VPi

Number of items marked as Poor – Pi

Number of items marked as Average - Ai

Number of items marked as Good - Gi

Number of items marked as Excellent - Ei

Average points per evaluation item= (Ei*10 + Gi*8 + Ai * 6 + Pi * 3 + VPi *1)/U

The critical line = 40%

7.3 Evaluation Forms

Please refer Appendix G for evaluation forms

7.4 Final Evaluation Results

Mobile Application: -

Total number of users who provided their feedback: - 17

Evaluation factors	Weight assigned to each option	Usability Evaluation	System Functionality Evaluation	Overall Impression of the system
No of questions in each section		5	5	5
Very poor	1	1	0	5
Poor	3	5	5	10
Average	6	50	55	50

Good	8	20	15	15
Excellent	10	9	10	5
Average points		33	33	29
Average Percentage		66%	66%	59%

Table 7.4 – Final Evaluation Estimation of Mobile Application

Web Application: -

Total number of users who provided their feedback: - 17

Evaluation factors	Weight assigned to each option	Usability Evaluation	System Functionality Evaluation	Overall Impression of the system
No of questions in each section		5	5	5
Very poor	1	1	1	0
Poor	3	3	6	2
Average	6	45	51	20
Good	8	25	13	40
Excellent	10	11	14	23
Average points		34	33	39
Average Percentage		69%	67%	79%

Table 7.4 – Final Evaluation Estimation of Web Application

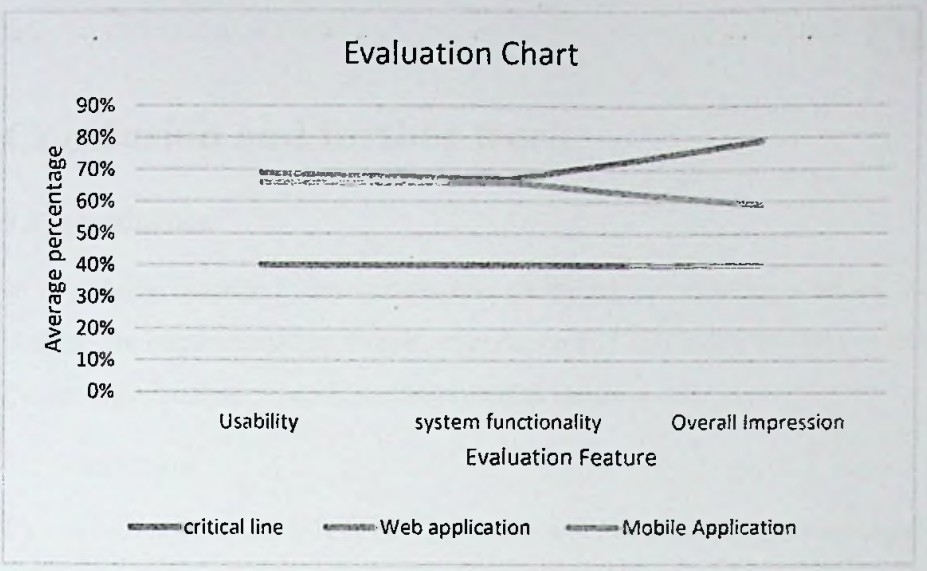


Figure 7.1 – Evaluation Chart

According to the evaluation chart given here, in Figure 7.1, all the evaluation points are above the defined critical point and therefore system has approached its target evaluation points.

7.5 Summary

According to the evaluation done in this chapter, the system has maintained its evaluation above the critical point in all aspects of the evaluation for both mobile and web application. In the following chapter, the conclusion and the future possible enhancements for CBTLS would be discussed.

Chapter 8

Conclusion and further work

8.1 Introduction

In the above chapter 7, the system was validated by selecting a set of users. In this chapter the implementation details, experience and validation results are discussed and the identified enhancements are described as future work.

8.2 Conclusion

The party whom are mainly benefited by this system would be the passengers. With the currently available systems, they are only capable to see the static train schedules. This new system will provide the following information to a passenger, regarding a selected train.

- Indication if the train is available or not (for current trains).
- The current position of the desired train.
- The crowd density in each compartment.
- Recommendations for trains based on historical data
- Facility to provide the passenger's suggestions, comments and criticisms regarding a selected train
- A location aware alarm to indicate when the desired destination is reached
- Facility to add train schedules in to a "Favorites List" for easy access

The above features would allow passengers to save the waiting time at the stations for a train. It would also allow to select an alternative method of transportation, in case a train is not available for the desired time, or the train has been delayed or cancelled. This can save many productive man hours for the country.

A set of authorized users for the web application, would be able to analyze patterns of train transportation and to identify the points where delays occur. The past stored data could be also analyzed and studied to provide a better train transportation service which will serve the need of passengers better.

The main risk for the system would be the possible inaccuracy and reliability issues of the data retrieved from the passengers. The inaccuracy could be reduced by through validation, but the reliability of data could only be determined through the amount of

similar data retrieved from different sources. To enhance the reliability, a rating system for users has been introduced.

The location awareness of the mobile application for passengers would require GPS or Android's Network Location Provider activated in the mobile application. Also the application would require an active internet connection to use this application. These could be considered as limitations of the system, since some users would not be agreed with these terms.

All the existing and proposed systems mentioned currently available, would require an involvement from Sri Lanka Railways (SLR), mainly for locating trains. It has been done by placing a GPS tracking device inside the train. In this CBTLS, there won't be a requirement for any involvement of SRL, since the data is expected to be fed by train passengers.

For the passengers, to use the mobile system, it would require an android mobile device with either GPS or Android's Network Location Provider activated, and an active internet connection.

For the web application, it would require a high performance server to host the application to handle the expected large amount of requests, since this is accessed by general public. At the same time, since the current locations of trains should be updated at regular intervals, requests for locations updates would be sent frequently to the server.

8.3 Future work

Though the CBTLS system, a set of data regarding each train would be collected daily, and this would be a new set of data, which were not available before in Sri Lanka. Therefore such a set of data could be used to generate new knowledge.

If the railway services would want to involve or accommodate this system, they could have done it without having to bear an additional cost, or at a minimal cost, since only requirement would be to place a smart mobile device inside the train.

To analyses and study the pattern of train transportation varying with the factor of weather conditions, or to determine if there's a relationship between weather conditions and train delays, another dimension could be integrated in to this system as to collect

weather information along with train location updates. For this purpose, a third party service could be easily integrated with the system.

Through the facility to provide their feedback regarding the train, passengers would be able to convey their suggestions, comments and criticisms for other passengers. This feature could be used by some responsible authority to get to know the feedback of general public on their services.

8.4 Summary

In this chapter the overview of CBTLS project was discussed along with the evaluation results. Furthermore an insight on future enhancements was also discussed.

References

[1] A. Kumarage, "Urban traffic congestion. the problem and its solutions," 2002.

[2] A. B. Jayasinghe and N. Pathiranage, "Centrality measures' as a tool to identify the transist demand at railway stations: the case of railway network, Sri Lanka," 2015.

[3] "Economic and social infrastructure - Central Bank of Sri Lanka - ANNUAL REPORT 2012," CENTRAL BANK OF SRI LANKA, ANNUAL REPORT 2012, Mar. 2013.

[4] "Statistics, Ministry of Internal Transport," *Ministry of Transport and Civil Aviation - Sri Lanka*, 28-Oct-2015. [Online]. Available: http://www.transport.gov.lk/web/index.php?option=com_content&view=article&id=141&Itemid=113&lang=en. [Accessed: 13-Mar-2016].

[5] G. Bradley, International Association for Development of the Information Society, and Albert-Ludwigs-Universität Freiburg, Eds., *Proceedings of the IADIS International Conference ICT, Society and Human Beings 2010: part of the IADIS Multi Conference on Computer Science and Information Systems 2010 ; Freiburg, Germany, July 29 - 31, 2010.* Lisboa: IADIS Press, 2010.

[6] S. Rainford, "e-Sri Lanka: An integrated approach to e-government case study," *Reg. Dev. Dialogue*, vol. 27, no. 2, pp. 209–218, 2006.

[7] ICTA, "Sri Lanka Railways - Train Schedule," *Sri Lanka Railways*, 2011. [Online]. Available: http://eservices.railway.gov.lk/schedule. [Accessed: 13-Mar-2016].

[8] G. Bhashitha Nadun, "Sri Lanka Train Schedule - Android Apps on Google Play," *Google Play*, 04-Mar-2014. [Online]. Available: https://play.google.com/store/apps/details?id=lk.icta.mobile.apps.railway. [Accessed: 13-Mar-2016].

[9] Leelaratne, "Train Schedules of Sri Lanka - Android Apps on Google Play," *Google Play*, 14-Oct-2014. [Online]. Available: https://play.google.com/store/apps/details?id=com.aselalee.trainschedule. [Accessed: 13-Mar-2016].

[10] K. Mobiles, "Train Guide - Sri Lanka," *Google Play*, 30-Jul-2014. [Online]. Available: https://play.google.com/store/apps/details?id=k.dw.timetable. [Accessed: 13-Mar-2016].

[11] "Railway Traffic Optimisation System," *Sri Lanka Railways*, 01-Aug-2014. [Online]. Available: www.slrail.info/tracking/timetable.php. [Accessed: 22-Nov-2015].

[12] Prasanna, "How to search where the train is in Sri lanka (system to keep track of trains)," *Synergy Y*, 17-Jul-2014.

[13] D. Jayakody, M. Gunawardana, N. W. Surendra, D. G. Jayasekara, C. Upendra, and R. De Silva, "GPS/GSM based train tracking system – utilizing mobile networks to support public transportation," 2011.

[14] N. S. Gunasekara, "GPS based tracking system for trains in Sri Lanka." 07-Jan-2006.

[15] ICTA, "Future Plans - Information Technology," *Sri Lanka Railways*, 11-Sep-2011. [Online]. Available: http://www.railway.gov.lk/web/index.php?option=com_content&view=article&id=126&Itemid=180&lang=en#IT. [Accessed: 13-Mar-2016].

[16] B. Coifman, D. Beymer, P. McLauchlan, and J. Malik, "A real-time computer vision system for vehicle tracking and traffic surveillance," *Transp. Res. Part C Emerg. Technol.*, vol. 6, no. 4, pp. 271–288, 1998.

[17] D. J. Dailey, L. Li, T. Northwest, and others, "Video image processing to create a speed sensor," Washington State Department of Transportation, 2000.

[18] N. Chadil, A. Russameesawang, and P. Keeratiwintakorn, "Real-time tracking management system using GPS, GPRS and Google earth," 2008, pp. 393–396.

Appendixes

Appendix A – User interface designs wireframes for mobile application

UI Wireframes of CBTLS mobile application

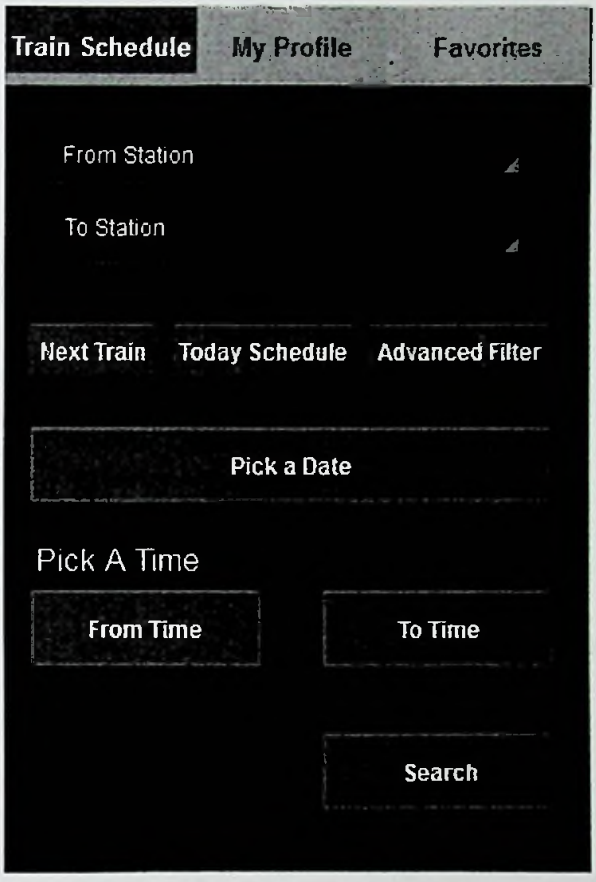


Figure 5.1 – CBTLS mobile application initial UI wireframe

Figure 5.1 would be the initial UI loaded to the mobile user, and this can serve as other existing system to search for train schedules. This UI was built to preserve that functionality and user experience. From here, by picking a start station and an end station, and an optional date and time range, users can search for train schedules.

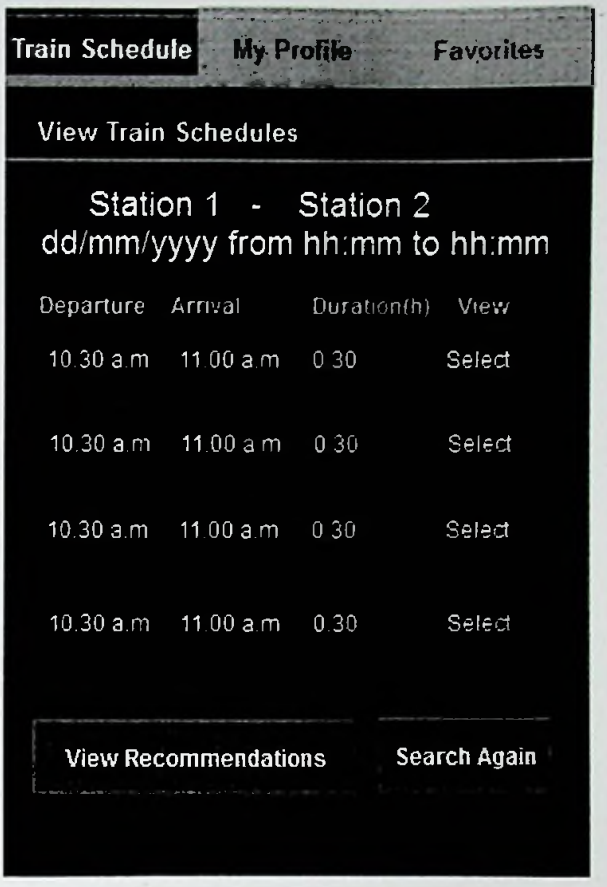


Figure 5.2 – CBTLS mobile application view train schedule wireframe

Once the schedules are searched through the initial user interface (in Figure 5.1), they will be get listed like in the above Figure 5.2. Up to this level, system behaves as an ordinary existing train schedule searching mobile application. Through this UI, advanced features like recommendations user interface (in Figure 5.3), and real time location UIs could be accessed. If the user accessed the "select" feature listed in each schedule, user will be directed to the user interface "View train details" (Figure 5.4)

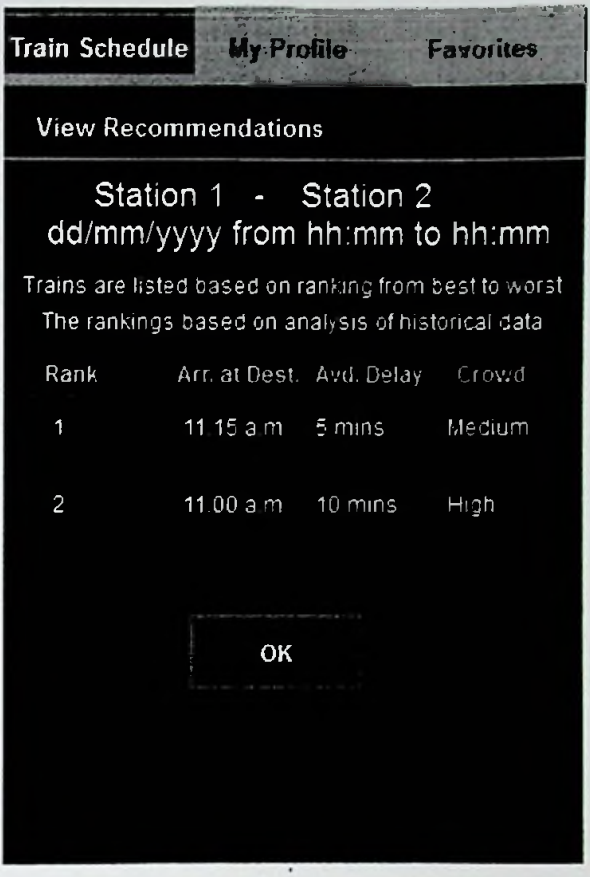


Figure 5.3 – CBTLS mobile application view recommendations wireframe

In the user interface indicated in Figure 5.3 here through an internal algorithm, system will order the train schedules for user, the considered facts are included like daily delay, crowd density, time of arrival at destination.

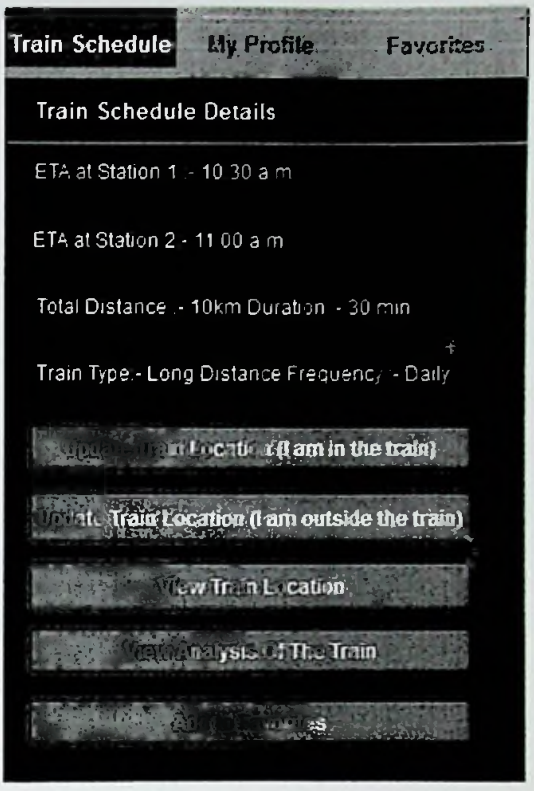


Figure 5.4 – CBTLS mobile application view train schedule details wireframe

In this user interface indicated in Figure 5.4, the user has selected a single train schedule. Therefor user can view the static data regarding selected schedule, and at the same time this UI provides access to the other major UIs as indicated.

The button named "Update train location (I am in the train)" would provide the access to the user interface (in Figure 5.5) where the user can actively update train location.

The button named "Update train location (I am outside the train)" would provide the access to the user interface (in Figure 5.8) where the user can passively update train location.

"View Train location" button would provide access to the user interface where user can view current location of the train (in Figure 5.9).

The button named "View analysis of the train" would provide access to the user interface (in Figure 5.11) where user can view analytical details of the train.

"Add to favorites" button will direct user to the interface where favorite schedules are listed for the user (in Figure 5.14).

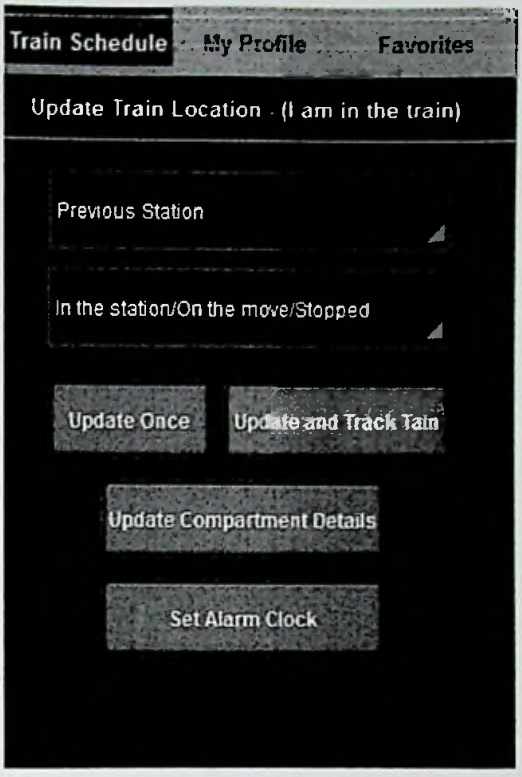


Figure 5.5 – CBTLS mobile application active update train location wireframe

The user interface in Figure 5.5 is mainly to actively update the current location of the train. Additionally it provides access to other user interfaces like update compartment details (in Figure 5.6) and set alarm clock (in Figure 5.7).

Rest of the UIs wireframes are listed below,

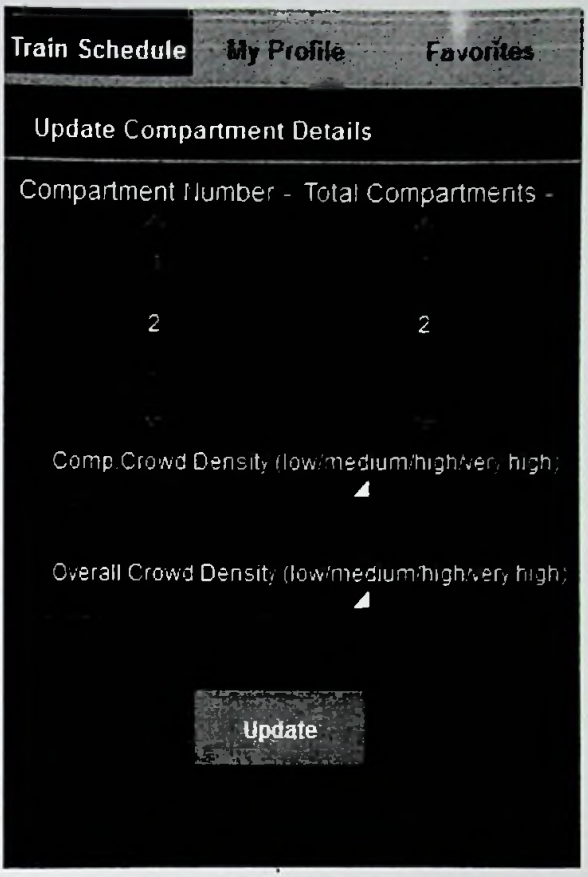


Figure 5.6 – CBTLS mobile application active update compartment details wireframe

The user interface in Figure 5.6, facilitate user to update compartment details, by selecting appropriate values from a predefined set as indicated in Figure 5.6 . "Update" button will sent the selected details to the backend server and direct user to the user interface named "View current compartment details" (Figure 5.10).

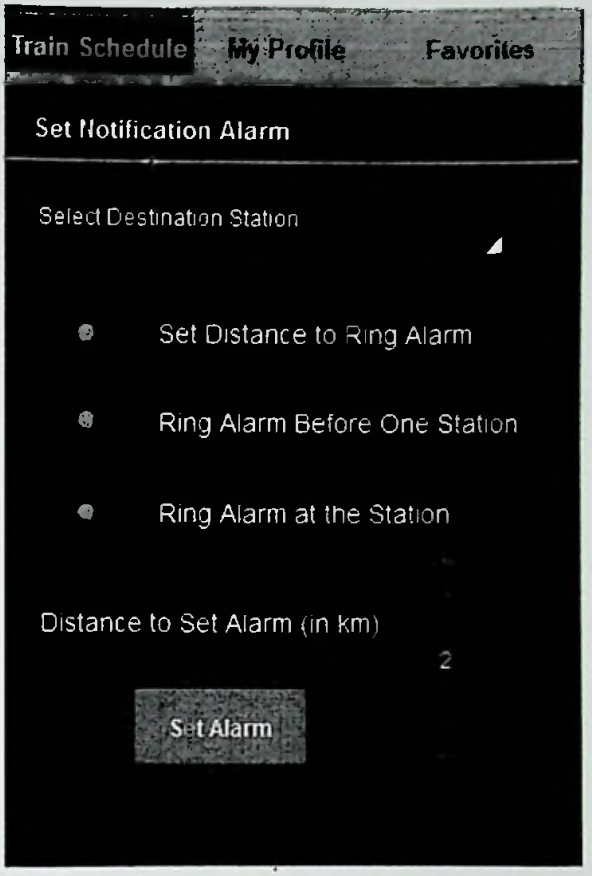


Figure 5.7 – CBTLS mobile application set notification alarm wireframe

The user interface in Figure 5.7 is to set the location based alarm clock for the user.

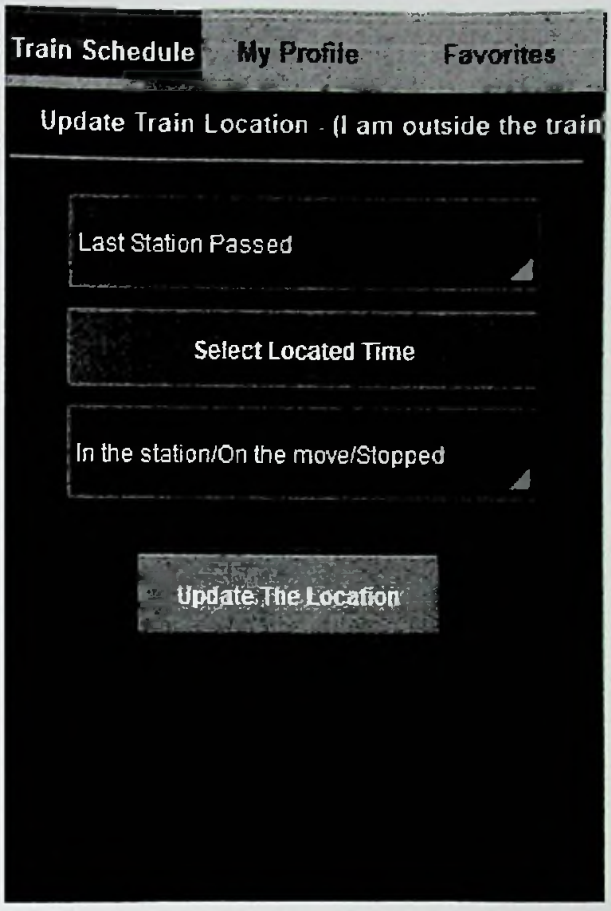


Figure 5.8 – CBTLS mobile application passive update train location wireframe

The user interface indicated in Figure 5.8 is to update the train location passively. Through this interface the location is not captured of the user instead it allows user to give an approximate location of the train.

Figure 5.9 – CBTLS mobile application view real-time train location wireframe

In the user interface indicated in Figure 5.9 the current location of the selected train will be indicated on a map, for current trains if the data is available with the system.

The "Refresh" button is to manually refresh the map for the user.

The button "View Compartment details" will direct the user to user interface available to view compartment details, in figure (Figure 5.10);

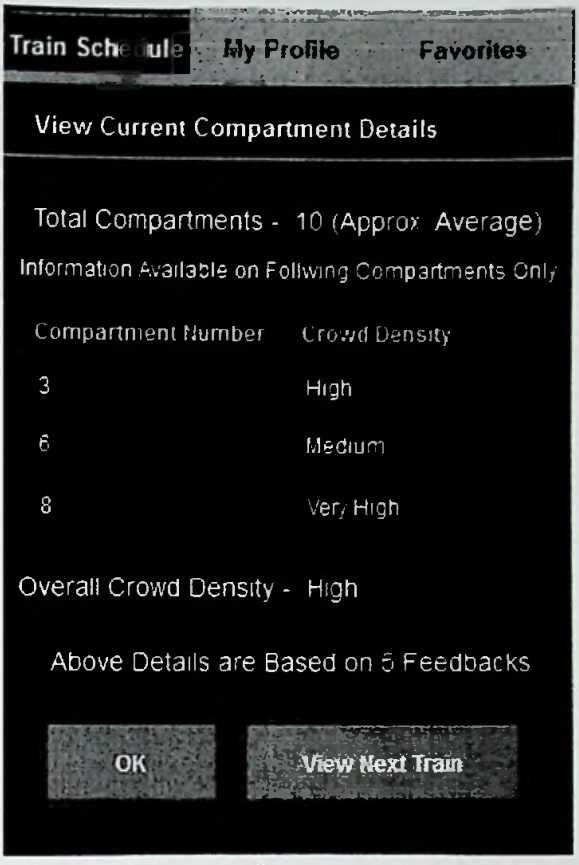


Figure 5.10 – CBTLS mobile application view compartment details wireframe

This user interface indicated in Figure 5.10 is to view the current reported compart details of the selected train.

"View Next Train" button will lead the user back in to the Figure 5.2 - "View Train Schedule" interface, where user can select a different train if required.

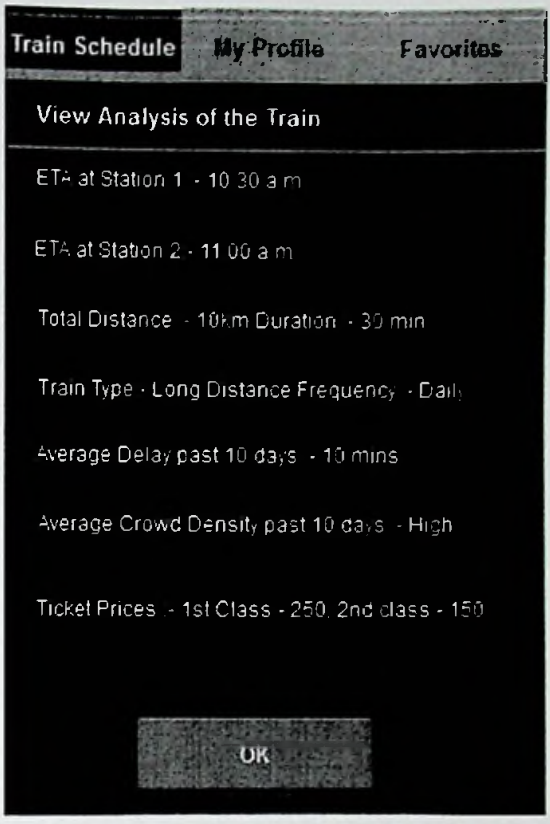


Figure 5.11 – CBTLS mobile application view analysis of train wireframe

The user interface indicated in Figure 5.11 facilitate the user to view further details of the selected train.

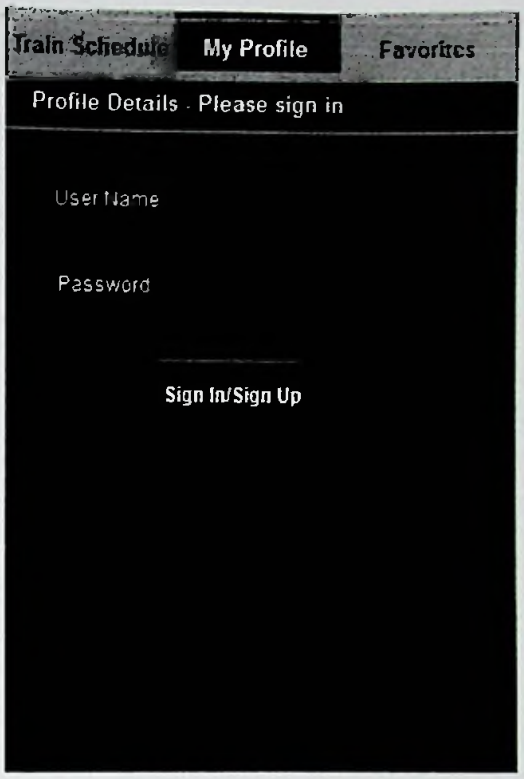


Figure 5.12 – CBTLS mobile application user login wireframe

Figure 5.12 user interface is to provide the login functionality for the user, in order to access limited functionalities of the system, user will have to login through this interface.

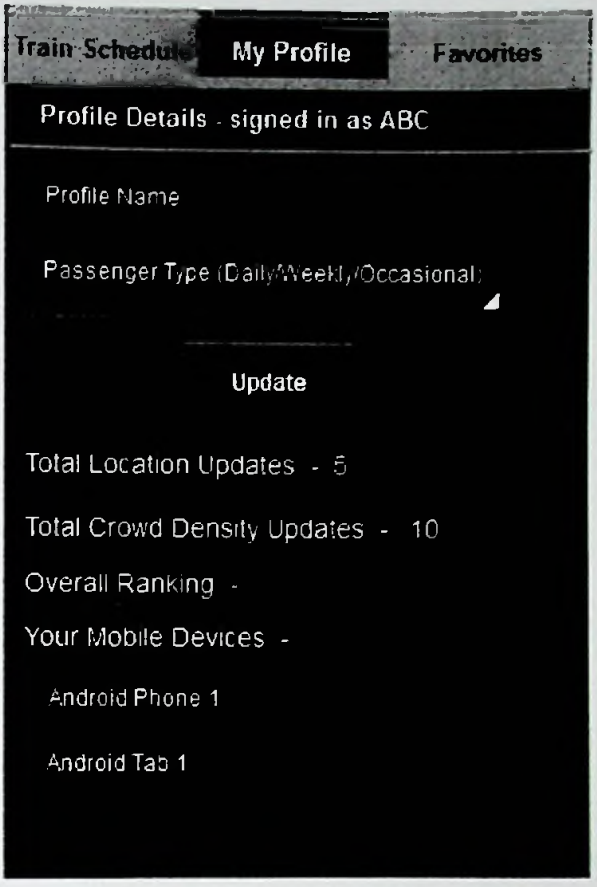


Figure 5.13 – CBTLS mobile application user profile details wireframe

As it indicated in Figure 5.13, this user interface facilitate the logged in user to update his/her profile.

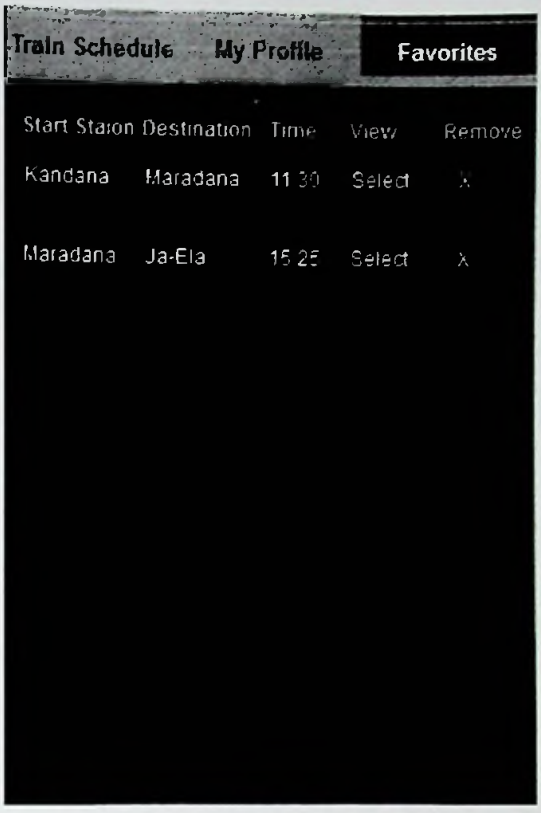


Figure 5.14 – CBTLS mobile application favorite trains wireframe

The user interface indicated in "Figure 5.14" is to list down the train schedules marked as "favorite" by the user, it is purely for the convenient usage of the system user.

Entity Relationship diagram of Community Based Train Locating System

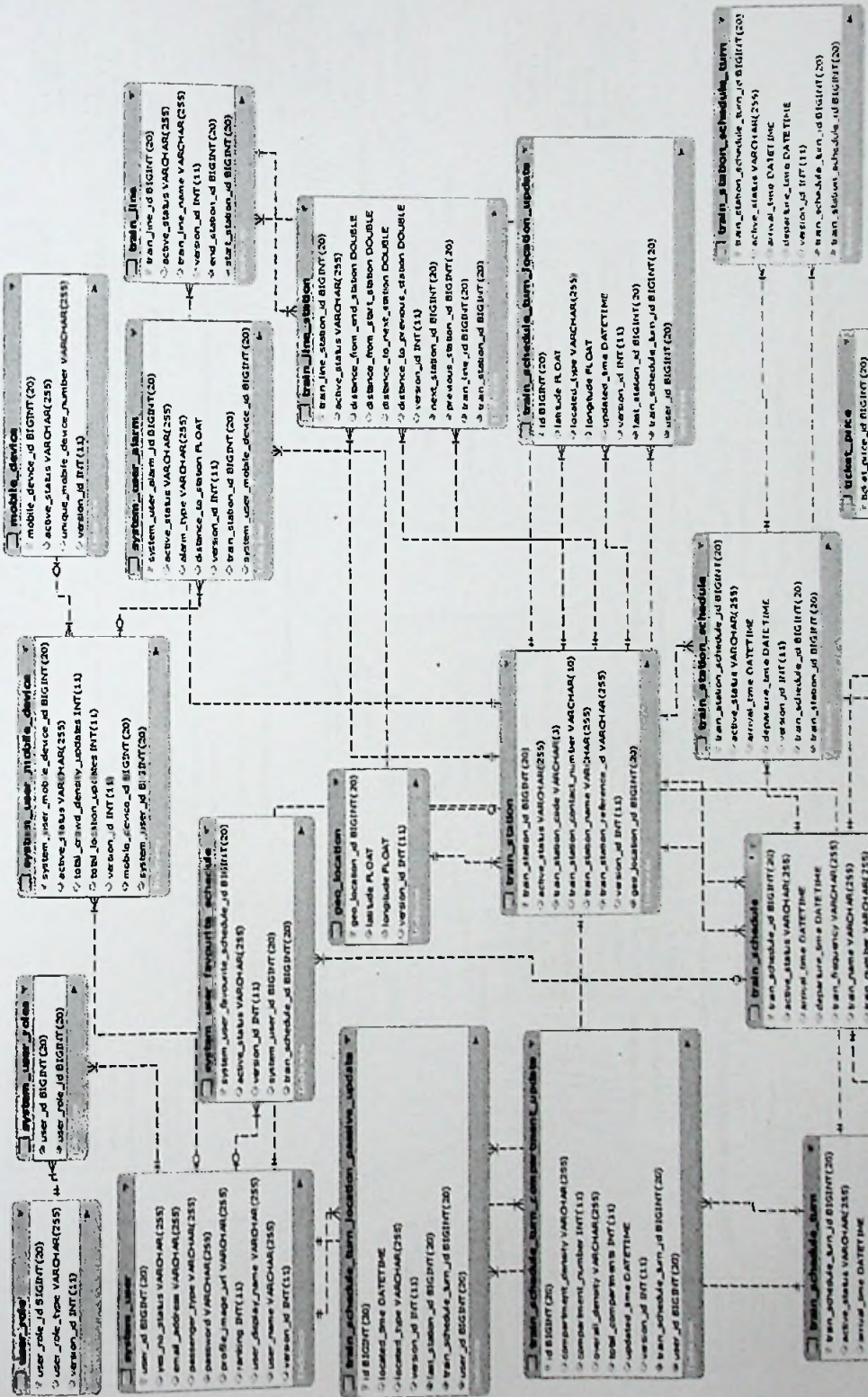


Figure 5.15 – CBTLS ER diagram

Class diagram of Community Based Train Locating System

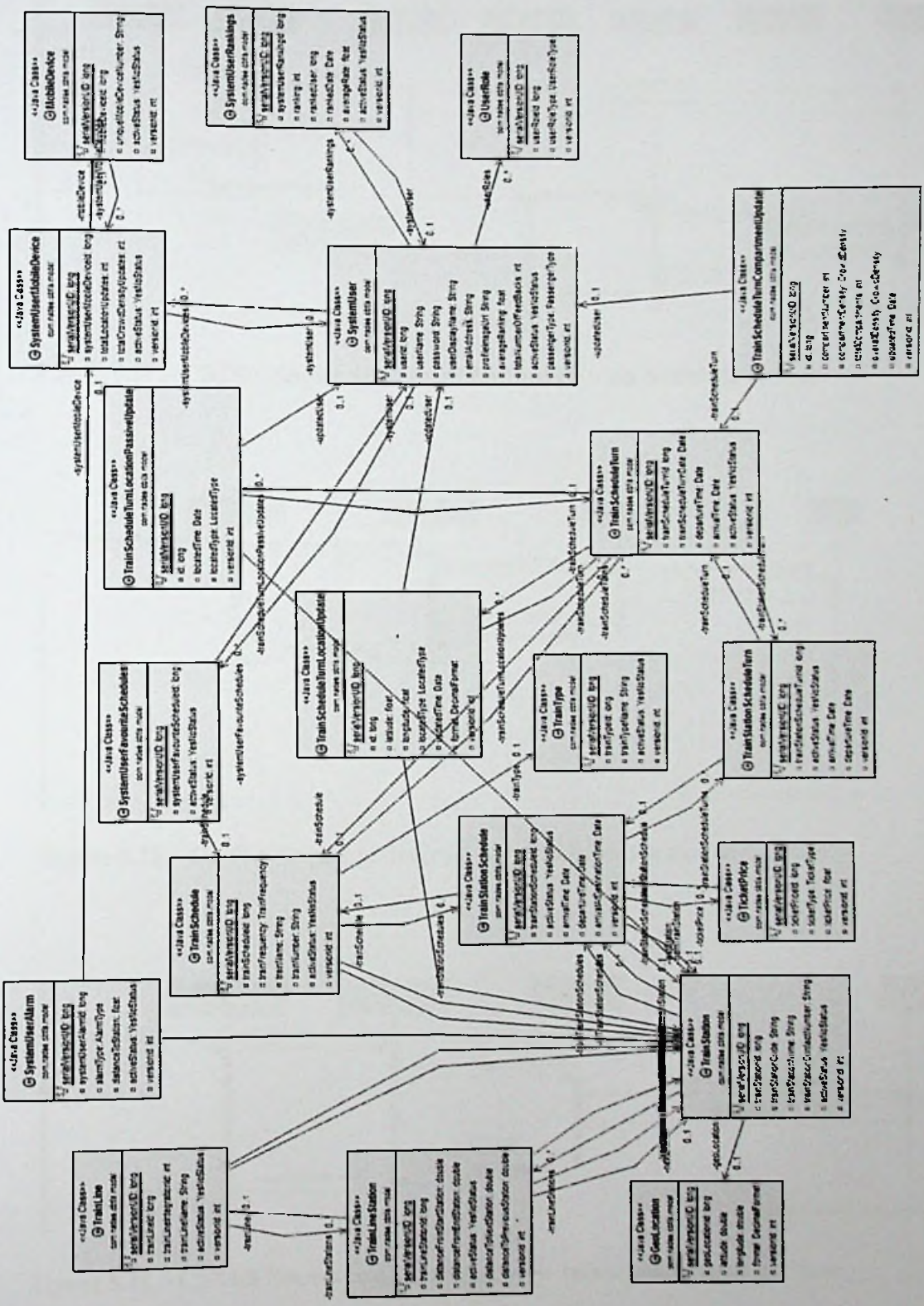


Figure 5.16 – CBTLS Class diagram

Sequence diagrams of Community Based Train Locating System

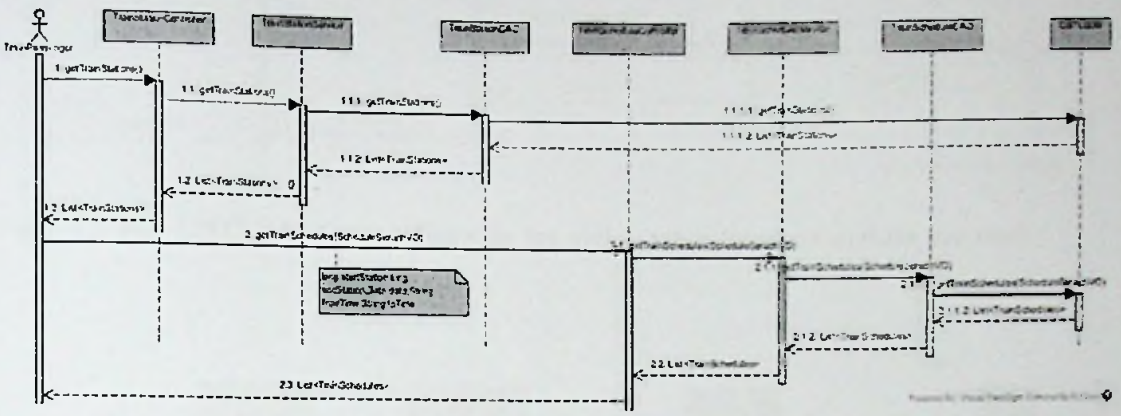


Figure 5.17 – CBTLS Sequence diagram for Search train Schedule use case

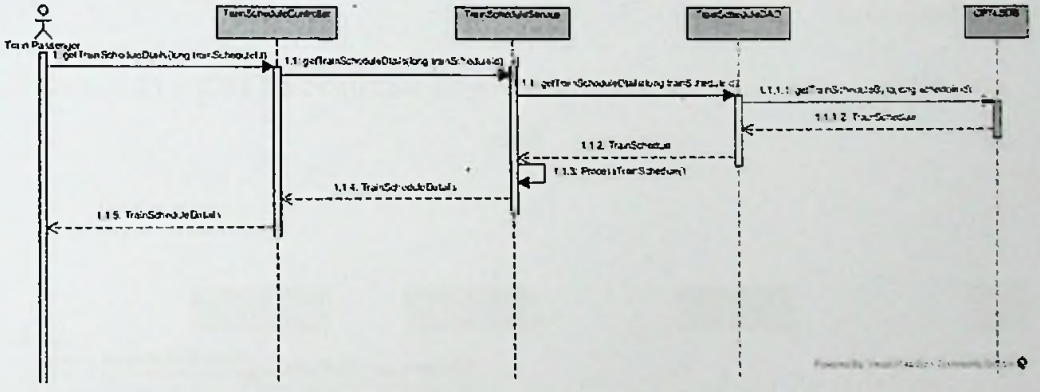


Figure 5.18 – CBTLS Sequence diagram for view schedule details use case

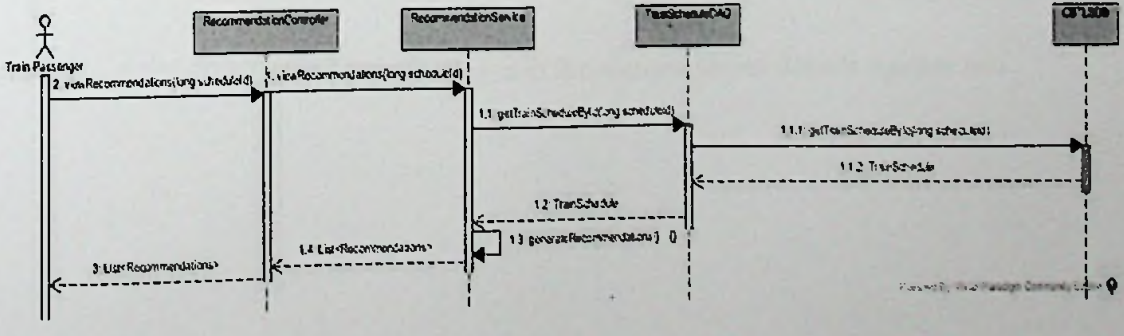


Figure 5.19 – CBTLS Sequence diagram for view recommendations use case

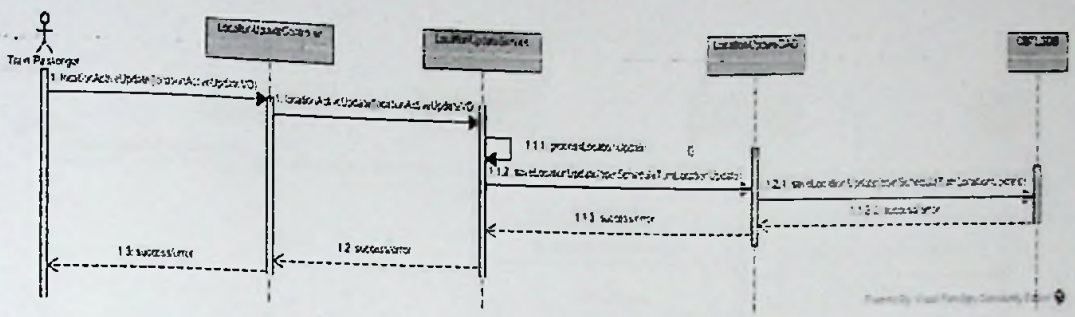


Figure 5.20 – CBTLS Sequence diagram for active train location update use case

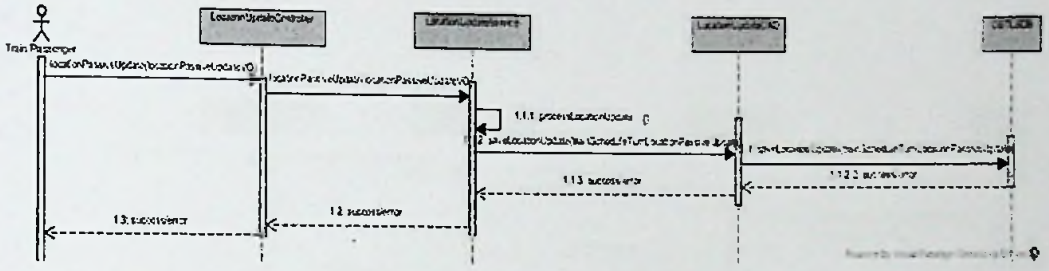


Figure 5.21 – CBTLS Sequence diagram for passive train location update use case

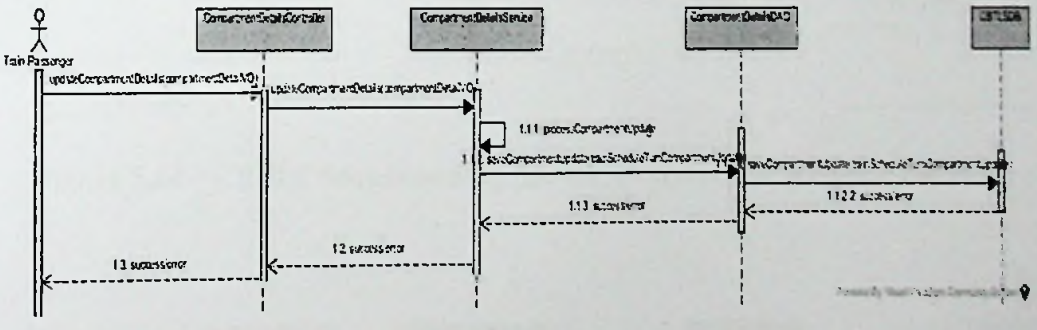


Figure 5.22 – CBTLS Sequence diagram for compartment details update use case

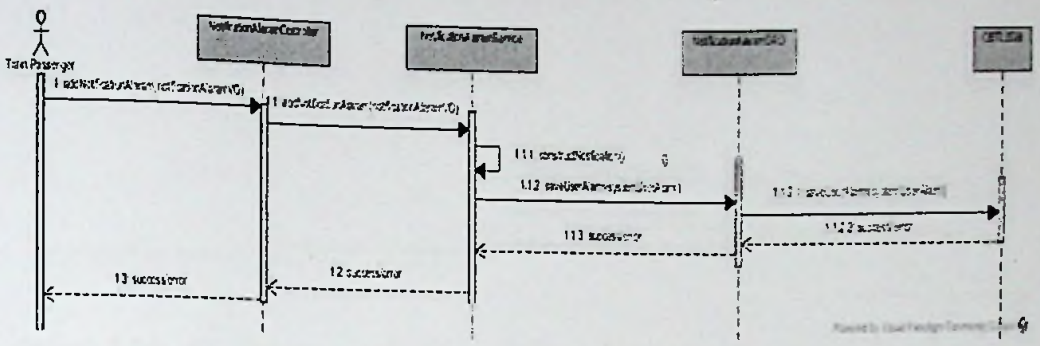


Figure 5.23 – CBTLS Sequence diagram for set alarm clock use case

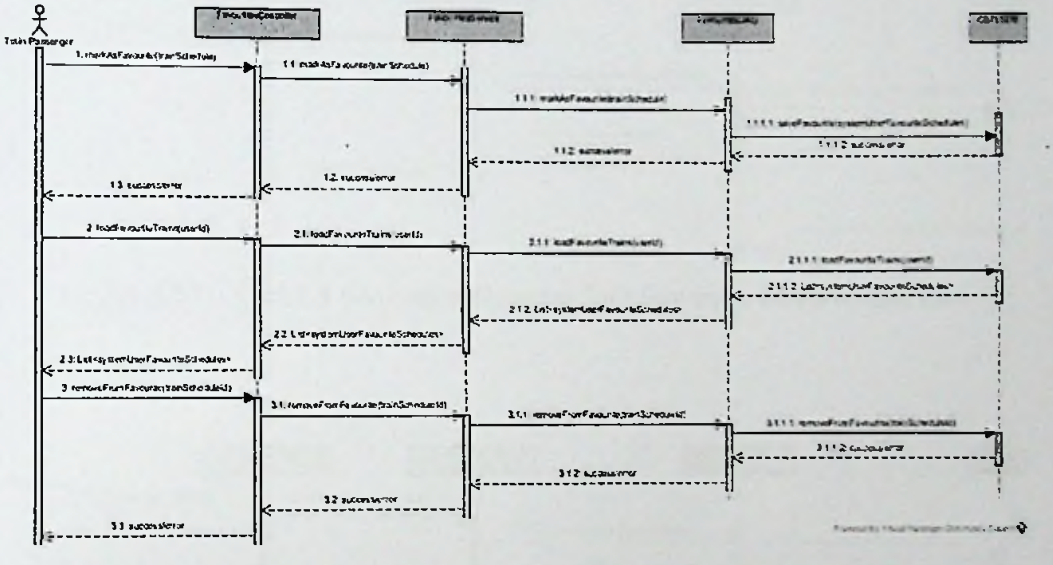


Figure 5.24 – CBTLS Sequence diagram for favorite train schedule use case

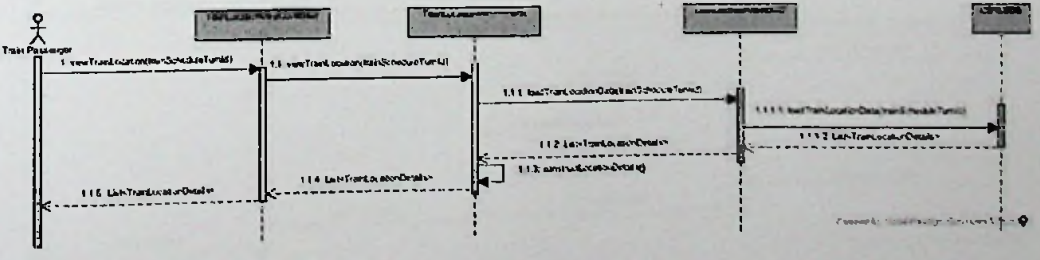


Figure 5.25 – CBTLS Sequence diagram for view train location use case

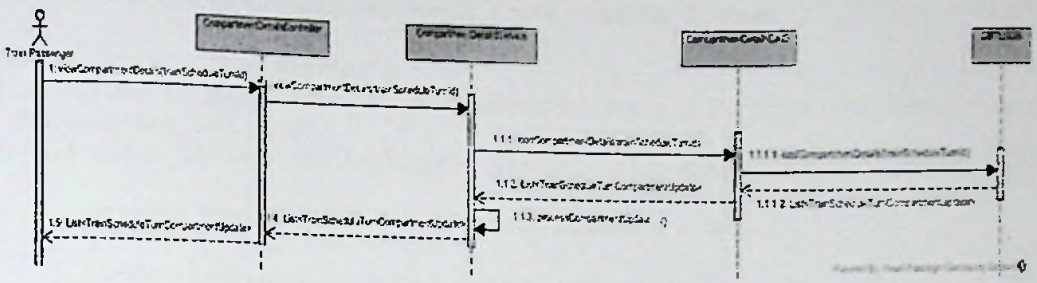


Figure 5.26 – CBTLS Sequence diagram for view compartment details use case

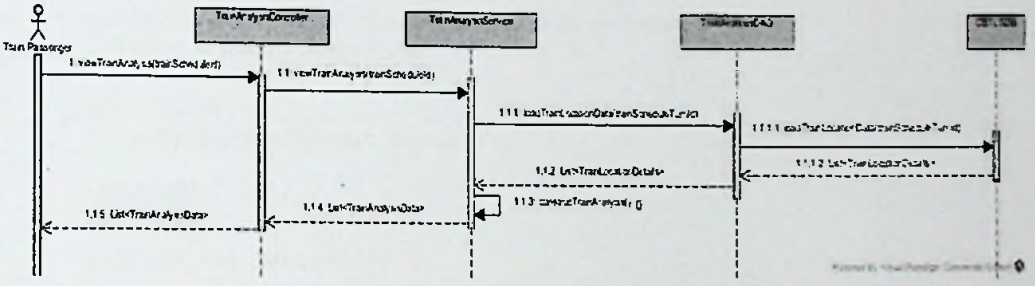


Figure 5.27 – CBTLS Sequence diagram for view train analysis use case

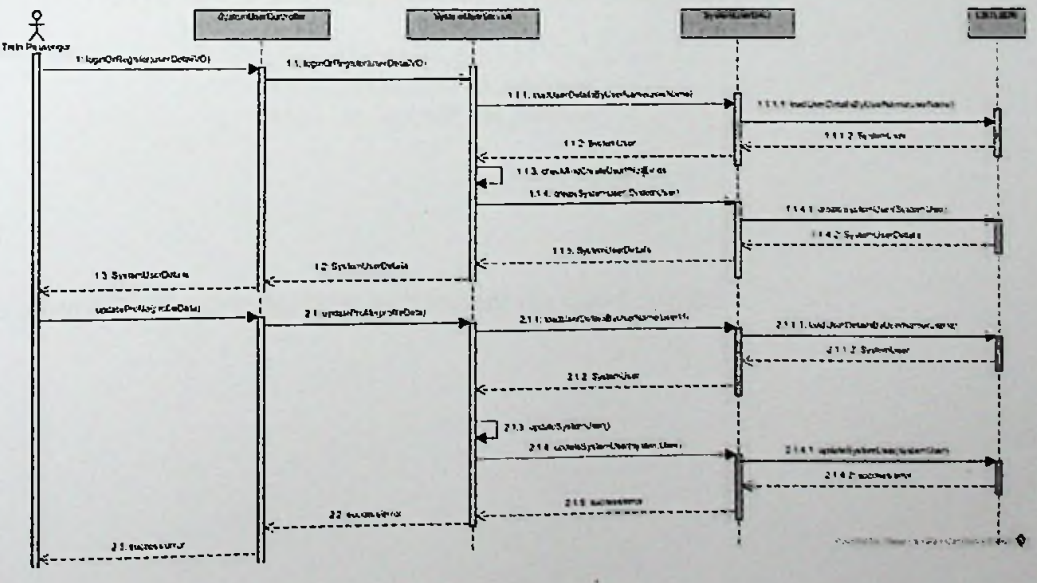


Figure 5.28 – CBTLS Sequence diagram for user sign in/profile update use case

Domain Classes of Community Based Train Locating System

```
@Entity(name = "geoLocation")
@Table(name = "geo_location", uniqueConstraints = { @UniqueConstraint(columnNames = {
"geo_location_id" }) })
public class GeoLocation implements Serializable, Comparable<GeoLocation> {

	private static final long serialVersionUID = 1L;
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	@Column(name = "geo_location_id", nullable = false)
	private long geoLocationId;

	@Column(name = "latitude", nullable = false)
	private double latitude;

	@Column(name = "longitude", nullable = false)
	private double longitude;

	@Transient
	private DecimalFormat format;

	@Version
	@Column(name = "version_id")
	private int versionId;

	public GeoLocation() {
		super();
		this.format = new DecimalFormat("##.######");
	}

	public GeoLocation(boolean defaultConstructor) {
		this();
		if(!(defaultConstructor)){
			this.latitude=00.000000;
			this.longitude=00.000000;
		}
	}

}
```

Figure 6.7 –The domain class to represent the Geo Location

```
@Entity(name="mobileDevice")
@Table(name = "mobile_device", uniqueConstraints =@UniqueConstraint(columnNames =
"unique_mobile_device_number"))
public class MobileDevice implements Serializable {

      private static final long serialVersionUID = 1L;

      @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "mobile_device_id", nullable = false)
      private long mobileDeviceId;

      @Column(name = "unique_mobile_device_number")
      private String uniqueMobileDeviceNumber;

      @Enumerated(EnumType.STRING)
    @Column(name="active_status")
      private YesNoStatus activeStatus;

      @OneToMany(mappedBy = "mobileDevice", fetch = FetchType.EAGER)
    @Cascade(org.hibernate.annotations.CascadeType.SAVE_UPDATE)
      private List<SystemUserMobileDevice> systemUserMobileDevices;

      @Version
    @Column(name = "version_id")
      private int versionId;
```

Figure 6.8 –The domain class to represent Mobile Device

```
@Entity(name = "systemUser")
@Table(name = "system_user", uniqueConstraints = @UniqueConstraint(columnNames = "user_name"))
public class SystemUser implements UserDetails,Serializable {
        private static final long serialVersionUID = 1L;

        @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "user_id", nullable = false)
        private long userId;

        @Column(name = "user_name", nullable = false)
        private String userName;

        @Column(name = "password", nullable = false)
        private String password;

        @Column(name = "user_display_name")
        private String userDisplayName;

        @Column(name = "email_address")
        private String emailAddress;

        @Column(name = "profile_image_url")
        private String profileImageUrl;

        @Column(name = "ranking")
        private float averageRanking;

        @Column(name = "total_number_of_feed_backs")
        private int totalNumberOfFeedBacks;

        @OneToMany(mappedBy = "systemUser", fetch = FetchType.EAGER)
    @Cascade(org.hibernate.annotations.CascadeType.ALL)
        @Fetch(value = FetchMode.SUBSELECT)
        private List<SystemUserRankings> systemUserRankings;

        @OneToMany(mappedBy = "systemUser", fetch = FetchType.EAGER)
    @Cascade(org.hibernate.annotations.CascadeType.ALL)
        @Fetch(value = FetchMode.SUBSELECT)
        private List<SystemUserMobileDevice> systemUserMobileDevices;

        @OneToMany(mappedBy = "systemUser", fetch = FetchType.EAGER)
    @Cascade(org.hibernate.annotations.CascadeType.ALL)
        @Fetch(value = FetchMode.SUBSELECT)
        private List<SystemUserFavouriteSchedules> systemUserFavouriteSchedules;

        @ManyToMany(fetch = FetchType.EAGER)
        @JoinTable(name = "system_user_roles", joinColumns = @JoinColumn(name = "user_id"),
inverseJoinColumns = @JoinColumn(name = "user_role_id"))
        @Cascade(org.hibernate.annotations.CascadeType.SAVE_UPDATE)
        @Fetch(value = FetchMode.SUBSELECT)
        private List<UserRole> userRoles;

        @Enumerated(EnumType.STRING)
        @Column(name = "yes_no_status")
        private YesNoStatus activeStatus;

        @Enumerated(EnumType.STRING)
        @Column(name = "passenger_type")
        private PassengerType passengerType;

        @Version
        @Column(name = "version_id")
        private int versionId;
```

Figure 6.9 –The domain class to represent System User

```
@Entity(name="systemUserAlarm")
@Table(name = "system_user_alarm")
public class SystemUserAlarm implements Serializable {
	private static final long serialVersionUID = 1L;

	@Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "system_user_alarm_id", nullable = false)
	private long systemUserAlarmId;

	@ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "system_user_mobile_device_id", referencedColumnName =
"system_user_mobile_device_id")
    @Cascade(org.hibernate.annotations.CascadeType.MERGE)
	private SystemUserMobileDevice systemUserMobileDevice;

	@ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "train_station_id", referencedColumnName = "train_station_id")
    @Cascade(org.hibernate.annotations.CascadeType.MERGE)
	private TrainStation destinationStation;

	@Enumerated(EnumType.STRING)
    @Column(name="alarm_type")
	private AlarmType alarmType;

	@Column(name="distance_to_station")
	private float distanceToStation;

	@Enumerated(EnumType.STRING)
    @Column(name="active_status")
	private YesNoStatus activeStatus;

	@Version
    @Column(name = "version_id")
	private int versionId;
```

Figure 6.10–The domain class to represent System User Alarm

```
@Entity(name="systemUserFavouriteSchedules")
@Table(name = "system_user_favourite_schedule")
public class SystemUserFavouriteSchedules implements Serializable {
	private static final long serialVersionUID = 1L;

	@Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "system_user_favourite_schedule_id", nullable = false)
	private long systemUserFavouriteScheduleId;

	@ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "train_schedule_id", referencedColumnName =
"train_schedule_id")
    @Cascade(org.hibernate.annotations.CascadeType.MERGE)
	private TrainSchedule trainSchedule;

	@ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "system_user_id", referencedColumnName = "user_id")
    @Cascade(org.hibernate.annotations.CascadeType.MERGE)
	private SystemUser systemUser;

	@Enumerated(EnumType.STRING)
    @Column(name="active_status")
	private YesNoStatus activeStatus;

	@Version
    @Column(name = "version_id")
	private int versionId;
```

Figure 6.11–The domain class to represent System User Favorite Schedules

```
@Entity(name="systemUserRankings")
@Table(name = "system_user_rankings")
public class SystemUserRankings implements Serializable {
      private static final long serialVersionUID = 1L;

      @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "system_user_ranking_id", nullable = false)
      private long systemUserRankingId;

      @Column(name = "ranking", nullable = false)
      private int ranking;

      @Column(name = "ranked_user", nullable = true)
      private long rankedUser;

      @Temporal(TemporalType.TIMESTAMP)
      @Column(name = "ranked_date", nullable = true)
      private Date rankedDate;

      @Column(name = "average_rate", nullable = false)
      private float averageRate;

      @Enumerated(EnumType.STRING)
    @Column(name="active_status")
      private YesNoStatus activeStatus;

      @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "system_user_id", referencedColumnName = "user_id")
    @Cascade(org.hibernate.annotations.CascadeType.MERGE)
      private SystemUser systemUser;

      @Version
    @Column(name = "version_id")
      private int versionId;
```

Figure 6.12–The domain class to represent System User Rankings

```
@Entity(name = "ticketPrice")
@Table(name = "ticket_price", uniqueConstraints = {@UniqueConstraint(columnNames =
{"ticket_price_id"})})
public class TicketPrice implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  @Column(name = "ticket_price_id", nullable = false)
    private long ticketPriceId;

    @ManyToOne(fetch=FetchType.EAGER,optional=false)
    @JoinColumn(name="train_station_schedule",nullable=false)
    @Cascade(org.hibernate.annotations.CascadeType.MERGE)
    private TrainStationSchedule trainStationSchedule;

    @Enumerated(EnumType.STRING)
    @Column(name = "user_role_type")
    private TicketType ticketType;

    @Column(name = "ticket_price")
    private float ticketPrice;

    @Version
    @Column(name = "version_id")
    private int versionId;
```

Figure 6.13–The domain class to represent Ticket Price

```
@Entity(name="trainLine")
@Table(name = "train_line", uniqueConstraints =@UniqueConstraint(columnNames =
"train_line_id"))
public class TrainLine implements Serializable {
       private static final long serialVersionUID = 1L;

       @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "train_line_id", nullable = false)
       private long trainLineId;

       @Column(name = "train_line_integration_id")
       private int trainLineIntegrationId;

       @Column(name = "train_line_name")
       private String trainLineName;

       @Enumerated(EnumType.STRING)
    @Column(name="active_status")
       private YesNoStatus activeStatus;

       @OneToMany(mappedBy = "trainLine", fetch = FetchType.LAZY)
    @Cascade(org.hibernate.annotations.CascadeType.SAVE_UPDATE)
       private List<TrainLineStation> trainLineStations;

       @ManyToOne(fetch=FetchType.LAZY)
       @Cascade(org.hibernate.annotations.CascadeType.MERGE)
       @JoinColumn(name="start_station_id")
       private TrainStation startStation;

       @JsonIgnore
       @ManyToOne(fetch=FetchType.LAZY)
       @JoinColumn(name="end_station_id")
       @Cascade(org.hibernate.annotations.CascadeType.MERGE)
       private TrainStation endStation;

       @Version
    @Column(name = "version_id")
       private int versionId;
```

Figure 6.14–The domain class to represent Train Line

```
@Entity(name="trainLineStation")
@Table(name = "train_line_station", uniqueConstraints =@UniqueConstraint(columnNames
= "train_line_station_id"))
public class TrainLineStation implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  @Column(name = "train_line_station_id", nullable = false)
    private long trainLineStationId;

    @Column(name = "distance_from_start_station")
    private double distanceFromStartStation;

    @Column(name = "distance_from_end_station")
    private double distanceFromEndStation;

    @Enumerated(EnumType.STRING)
  @Column(name="active_status")
    private YesNoStatus activeStatus;

    @JsonIgnore
    @ManyToOne(fetch=FetchType.LAZY,optional=false)
    @JoinColumn(name="train_line_id",nullable=false)
    @Cascade(org.hibernate.annotations.CascadeType.MERGE)
    private TrainLine trainLine;

    @JsonIgnore
    @ManyToOne(fetch=FetchType.EAGER,optional=false)
    @JoinColumn(name="train_station_id",nullable=false)
    @Cascade(org.hibernate.annotations.CascadeType.MERGE)
    private TrainStation trainStation;

    @ManyToOne(fetch=FetchType.LAZY)
    @JoinColumn(name="next_station_id")
    @Cascade(org.hibernate.annotations.CascadeType.MERGE)
    private TrainStation nextStation;

    @Column(name = "distance_to_next_station")
    private double distanceToNextStation;

    @ManyToOne(fetch=FetchType.LAZY)
    @Cascade(org.hibernate.annotations.CascadeType.MERGE)
    @JoinColumn(name="previous_station_id")
    private TrainStation previousStation;

    @Column(name = "distance_to_previous_station")
    private double distanceToPreviousStation;


    @Version
  @Column(name = "version_id")
    private int versionId;
```

Figure 6.15–The domain class to represent Train Line Station

```
//record the train schedules - master data
@Entity(name="trainSchedule")
@Table(name = "train_schedule", uniqueConstraints =@UniqueConstraint(columnNames =
"train_schedule_id"))
public class TrainSchedule implements Serializable {
      private static final long serialVersionUID = 1L;

      @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "train_schedule_id", nullable = false)
      private long trainScheduleId;

      @Enumerated(EnumType.STRING)
    @Column(name="train_frequency")
      private TrainFrequency trainFrequency;

      @Column(name="train_name")
      private String trainName;

      @Column(name="train_number")
      private String trainNumber;

      @ManyToOne(fetch=FetchType.EAGER,optional=false)
      @JoinColumn(name="start_station_id",nullable=false)
      private TrainStation startStation;

      @ManyToOne(fetch=FetchType.EAGER,optional=false)
      @JoinColumn(name="end_station_id",nullable=false)
      private TrainStation endStation;

      @ManyToOne(fetch=FetchType.EAGER,optional=false)
      @JoinColumn(name="train_type_id",nullable=false)
      @Cascade(org.hibernate.annotations.CascadeType.MERGE)
      private TrainType trainType;

      @OneToMany(mappedBy = "trainSchedule", fetch = FetchType.LAZY)
    @Cascade(org.hibernate.annotations.CascadeType.SAVE_UPDATE)
      private List<TrainStationSchedule> trainStationSchedules;

      @OneToMany(mappedBy = "trainSchedule", fetch = FetchType.LAZY)
    @Cascade(org.hibernate.annotations.CascadeType.SAVE_UPDATE)
      private List<TrainScheduleTurn> trainScheduleTurns;

      @Enumerated(EnumType.STRING)
    @Column(name="active_status")
      private YesNoStatus activeStatus;

      @Version
    @Column(name = "version_id")
      private int versionId;
```

Figure 6.16–The domain class to represent Train Schedule

```
@Entity(name="trainScheduleTurn")
@Table(name = "train_schedule_turn", uniqueConstraints =@UniqueConstraint(columnNames
= "train_schedule_turn_id"))
public class TrainScheduleTurn implements Serializable {
      private static final long serialVersionUID = 1L;

      @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "train_schedule_turn_id", nullable = false)
      private long trainScheduleTurnId;

      @ManyToOne(fetch=FetchType.EAGER,optional=false)
      @JoinColumn(name="train_schedule_id",nullable=false)
      private TrainSchedule trainSchedule;

      @Column(name="train_schedule_turn_date")
      @Temporal(TemporalType.TIMESTAMP)
      private Date trainScheduleTurnDate;

      @Column(name="departure_time")
      @Temporal(TemporalType.TIMESTAMP)
      private Date departureTime;

      @Column(name="arrival_time")
      @Temporal(TemporalType.TIMESTAMP)
      private Date arrivalTime;

      @OneToMany(mappedBy = "trainScheduleTurn", fetch = FetchType.LAZY)
    @Cascade(org.hibernate.annotations.CascadeType.SAVE_UPDATE)
      private List<TrainStationScheduleTurn> trainStationScheduleTurn;

      @OneToMany(mappedBy = "trainScheduleTurn", fetch = FetchType.LAZY)
    @Cascade(org.hibernate.annotations.CascadeType.SAVE_UPDATE)
      private List<TrainScheduleTurnLocationUpdate>
trainScheduleTurnLocationUpdates;

      @OneToMany(mappedBy = "trainScheduleTurn", fetch = FetchType.LAZY)
    @Cascade(org.hibernate.annotations.CascadeType.SAVE_UPDATE)
      private List<TrainScheduleTurnLocationPassiveUpdate>
trainScheduleTurnLocationPassiveUpdates;

      @Enumerated(EnumType.STRING)
    @Column(name="active_status")
      private YesNoStatus activeStatus;

      @Version
    @Column(name = "version_id")
      private int versionId;
```

Figure 6.17–The domain class to represent Train Schedule Turn

```
@Entity(name = "trainScheduleTurnCompartmentUpdate")
@Table(name = "train_schedule_turn_compartment_update", uniqueConstraints = {
public class TrainScheduleTurnCompartmentUpdate implements Serializable{

        private static final long serialVersionUID = 1L;
        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        @Column(name = "id", nullable = false)
        private long id;

        @Column(name = "compartment_number", nullable = false)
        private int compartmentNumber;

        @Enumerated(EnumType.STRING)
        @Column(name = "compartment_density")
        private CrowdDensity compartmentDensity;

        @Column(name = "total_compartments", nullable = false)
        private int totalCompartments;

        @Enumerated(EnumType.STRING)
        @Column(name = "overall_density")
        private CrowdDensity overallDensity;

        @ManyToOne(fetch=FetchType.EAGER,optional=false)
        @JoinColumn(name="train_schedule_turn_id",nullable=false)
        private TrainScheduleTurn trainScheduleTurn;

        @ManyToOne(fetch=FetchType.EAGER,optional=false)
        @JoinColumn(name="user_id",nullable=false)
        private SystemUser updatedUser;

        @Column(name="updated_time")
        @Temporal(TemporalType.TIMESTAMP)
        private Date updatedTime;


        @Version
        @Column(name = "version_id")
        private int versionId;
```

Figure 6.18–The domain class to represent Compartment Update

```
@Entity(name = "trainScheduleTurnLocationPassiveUpdate")
@Table(name = "train_schedule_turn_location_passive_update", uniqueConstraints = {
@UniqueConstraint(columnNames = { "id" }) })
public class TrainScheduleTurnLocationPassiveUpdate implements Serializable{

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id", nullable = false)
    private long id;

    @ManyToOne(fetch=FetchType.EAGER,optional=false)
    @JoinColumn(name="train_schedule_turn_id",nullable=false)
    private TrainScheduleTurn trainScheduleTurn;

    @ManyToOne(fetch=FetchType.EAGER,optional=false)
    @JoinColumn(name="last_station_id",nullable=false)
    private TrainStation lastStation;

    @ManyToOne(fetch=FetchType.EAGER,optional=false)
    @JoinColumn(name="user_id",nullable=false)
    private SystemUser updatedUser;

    @Column(name="located_time")
    @Temporal(TemporalType.TIMESTAMP)
    private Date locatedTime;

    @Enumerated(EnumType.STRING)
    @Column(name = "located_type")
    private LocatedType locatedType;

    @Version
    @Column(name = "version_id")
    private int versionId;
```

Figure 6.19–The domain class to represent Train Location Passive Update

```
@Entity(name = "trainScheduleTurnLocationUpdate")
@Table(name = "train_schedule_turn_location_update", uniqueConstraints = {
public class TrainScheduleTurnLocationUpdate implements Serializable,
Comparable<TrainScheduleTurnLocationUpdate> {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id", nullable = false)
    private long id;

    @Column(name = "latitude", nullable = false)
    private float latitude;

    @Column(name = "longitude", nullable = false)
    private float longitude;

    @ManyToOne(fetch=FetchType.EAGER,optional=false)
    @JoinColumn(name="train_schedule_turn_id",nullable=false)
    private TrainScheduleTurn trainScheduleTurn;

    @ManyToOne(fetch=FetchType.EAGER,optional=false)
    @JoinColumn(name="last_station_id",nullable=false)
    private TrainStation lastStation;

    @Enumerated(EnumType.STRING)
    @Column(name = "located_type")
    private LocatedType locatedType;

    @ManyToOne(fetch=FetchType.EAGER,optional=false)
    @JoinColumn(name="user_id",nullable=false)
    private SystemUser updatedUser;

    @Column(name="updated_time")
    @Temporal(TemporalType.TIMESTAMP)
    private Date updatedTime;

    @Transient
    private DecimalFormat format;

    @Version
    @Column(name = "version_id")
    private int versionId;
```

Figure 6.20–The domain class to represent Train Location active Update

```
@Entity(name="trainStation")
@Table(name = "train_station", uniqueConstraints =@UniqueConstraint(columnNames =
{"train_station_id","train_station_code"}))
public class TrainStation implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  @Column(name = "train_station_id", nullable = false)
    private long trainStationId;


    @OneToOne(fetch=FetchType.EAGER,optional=false)
    @JoinColumn(name="geo_location_id",nullable=false)
    @Cascade(org.hibernate.annotations.CascadeType.SAVE_UPDATE)
    private GeoLocation geoLocation;

    @Column(name = "train_station_code", nullable = false, length=25)
    private String trainStationCode;

    @Column(name = "train_station_name", length=255)
    private String trainStationName;

    @Column(name = "train_station_contact_number", length=10)
    private String trainStationContactNumber;

    @Enumerated(EnumType.STRING)
  @Column(name="active_status")
    private YesNoStatus activeStatus;

    @OneToMany(mappedBy = "trainStation")
  @Cascade(org.hibernate.annotations.CascadeType.SAVE_UPDATE)
    private List<TrainLineStation> trainLineStations;

    @OneToMany(mappedBy = "fromTrainStation")
  @Cascade(org.hibernate.annotations.CascadeType.SAVE_UPDATE)
    private List<TrainStationSchedule> fromTrainStationSchedules;

    @OneToMany(mappedBy = "toTrainStation")
  @Cascade(org.hibernate.annotations.CascadeType.SAVE_UPDATE)
    private List<TrainStationSchedule> toTrainStationSchedules;

    @Version
  @Column(name = "version_id")
    private int versionId;
```

Figure 6.21–The domain class to represent Train Station

```
@Entity(name="trainStationSchedule")
@Table(name = "train_station_schedule", uniqueConstraints
=@UniqueConstraint(columnNames = {"train_station_schedule_id"}))
public class TrainStationSchedule implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
@GeneratedValue(strategy = GenerationType.AUTO)
@Column(name = "train_station_schedule_id", nullable = false)
    private long trainStationScheduleId;

    @ManyToOne(fetch=FetchType.EAGER,optional=false)
    @JoinColumn(name="train_schedule_id",nullable=false)
    @Cascade(org.hibernate.annotations.CascadeType.MERGE)
    private TrainSchedule trainSchedule;

    @ManyToOne(fetch=FetchType.EAGER,optional=false)
    @JoinColumn(name="from_train_station_id",nullable=false)
    private TrainStation fromTrainStation;

    @ManyToOne(fetch=FetchType.EAGER,optional=false)
    @JoinColumn(name="to_train_station_id",nullable=false)
    private TrainStation toTrainStation;

    @Enumerated(EnumType.STRING)
@Column(name="active_status")
    private YesNoStatus activeStatus;

    @Column(name="arrival_time")
    @Temporal(TemporalType.TIMESTAMP)
    private Date arrivalTime;

    @Column(name="departure_time")
    @Temporal(TemporalType.TIMESTAMP)
    private Date departureTime;

    @Column(name="arrival_at_destination_time")
    @Temporal(TemporalType.TIMESTAMP)
    private Date arrivalAtDestinationTime;

    @OneToMany(mappedBy = "trainStationSchedule", fetch = FetchType.LAZY)
@Cascade(org.hibernate.annotations.CascadeType.SAVE_UPDATE)
    private List<TicketPrice> ticketPrice;

    @OneToMany(mappedBy = "trainStationSchedule", fetch = FetchType.LAZY)
@Cascade(org.hibernate.annotations.CascadeType.SAVE_UPDATE)
    private List<TrainStationScheduleTurn> trainStationScheduleTurns;


    @Version
@Column(name = "version_id")
    private int versionId;
```

Figure 6.22–The domain class to represent Train Station Schedule

```
//record the train schedule details daily per station
@Entity(name="trainStationScheduleTurn")
@Table(name = "train_station_schedule_turn")
=@UniqueConstraint(columnNames = {"train_station_schedule_turn_id"}))
public class TrainStationScheduleTurn implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  @Column(name = "train_station_schedule_turn_id", nullable = false)
    private long trainStationScheduleTurnId;

    @ManyToOne(fetch=FetchType.EAGER,optional=false)
    @JoinColumn(name="train_schedule_turn_id",nullable=false)
    private TrainScheduleTurn trainScheduleTurn;

    @ManyToOne(fetch=FetchType.EAGER,optional=false)
    @JoinColumn(name="train_station_schedule_id",nullable=false)
    private TrainStationSchedule trainStationSchedule;

    @Enumerated(EnumType.STRING)
  @Column(name="active_status")
    private YesNoStatus activeStatus;

    @Column(name="arrival_time")
    @Temporal(TemporalType.TIMESTAMP)
    private Date arrivalTime;

    @Column(name="departure_time")
    @Temporal(TemporalType.TIMESTAMP)
    private Date departureTime;


    @Version
  @Column(name = "version_id")
    private int versionId;
```

Figure 6.23–The domain class to represent Train Station Schedule Turn

```
@Entity(name="trainType")
@Table(name = "train_type", uniqueConstraints =@UniqueConstraint(columnNames =
"train_type_id"))
public class TrainType implements Serializable {

       private static final long serialVersionUID = 1L;

       @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "train_type_id", nullable = false)
       private long trainTypeId;

       @Column(name = "train_type_name")
       private String trainTypeName;

       @Enumerated(EnumType.STRING)
    @Column(name="active_status")
       private YesNoStatus activeStatus;


       @Version
    @Column(name = "version_id")
       private int versionId;
```

Figure 6.24–The domain class to represent Train Type

```
@Entity(name = "userRole")
@Table(name = "user_role", uniqueConstraints = {@UniqueConstraint(columnNames =
{"user_role_id"})})
public class UserRole implements GrantedAuthority, Serializable {

       private static final long serialVersionUID = 1L;
       @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "user_role_id", nullable = false)
       private long userRoleId;

       @Enumerated(EnumType.STRING)
       @Column(name = "user_role_type")
       private UserRoleType userRoleType;

       @Version
       @Column(name = "version_id")
       private int versionId;
```

Figure 6.25–The domain class to represent User Role

User interfaces of CBTLS web application

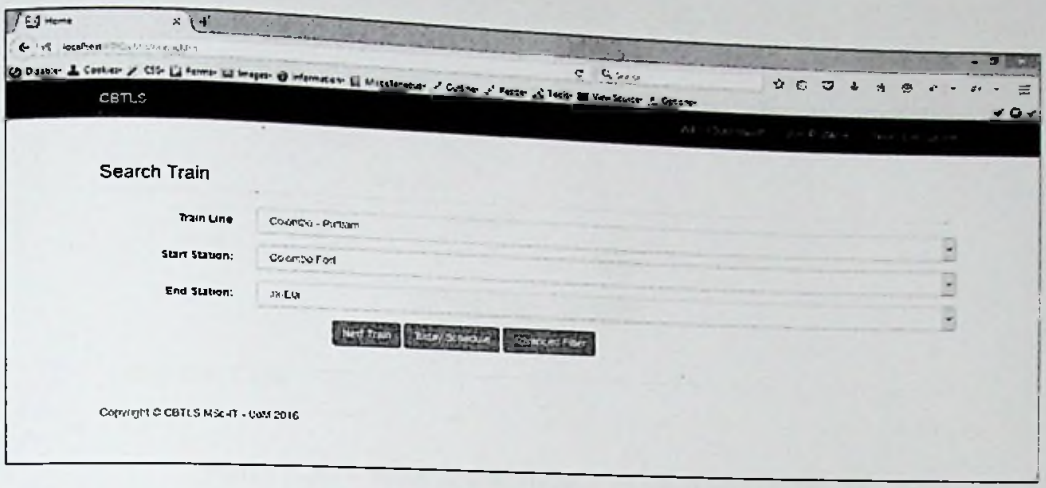


Figure 6.31 – Web application – Search Train basic UI

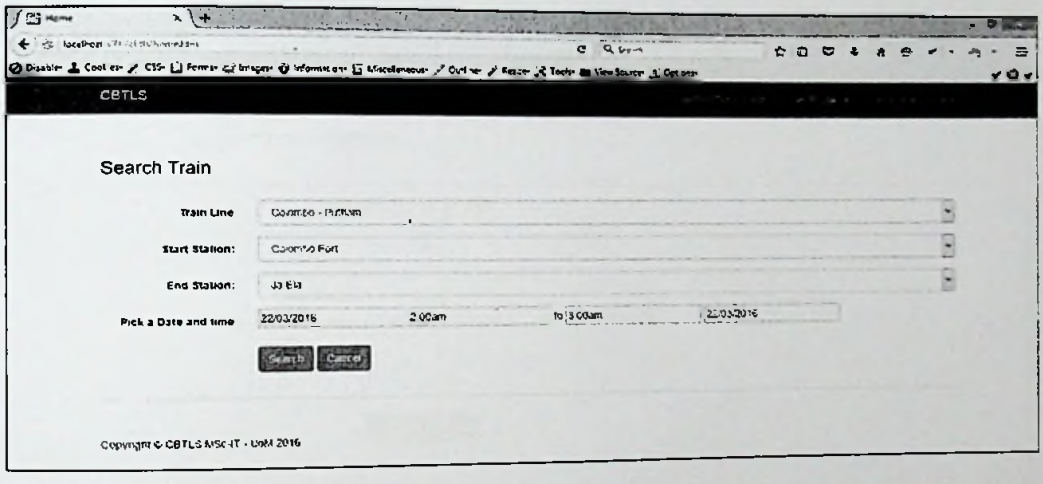


Figure 6.32 – Web application – Search Train advanced UI

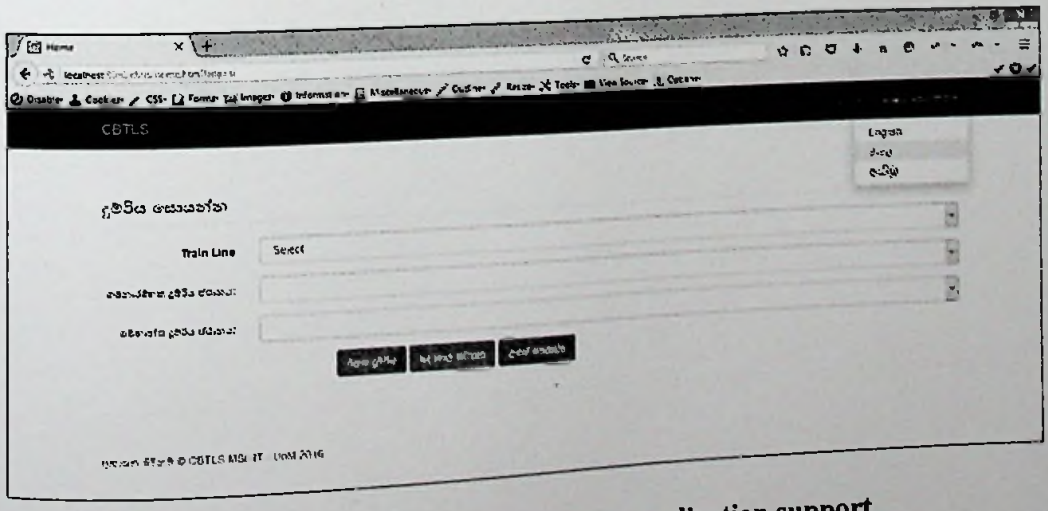


Figure 6.33 – Web application – Localization support

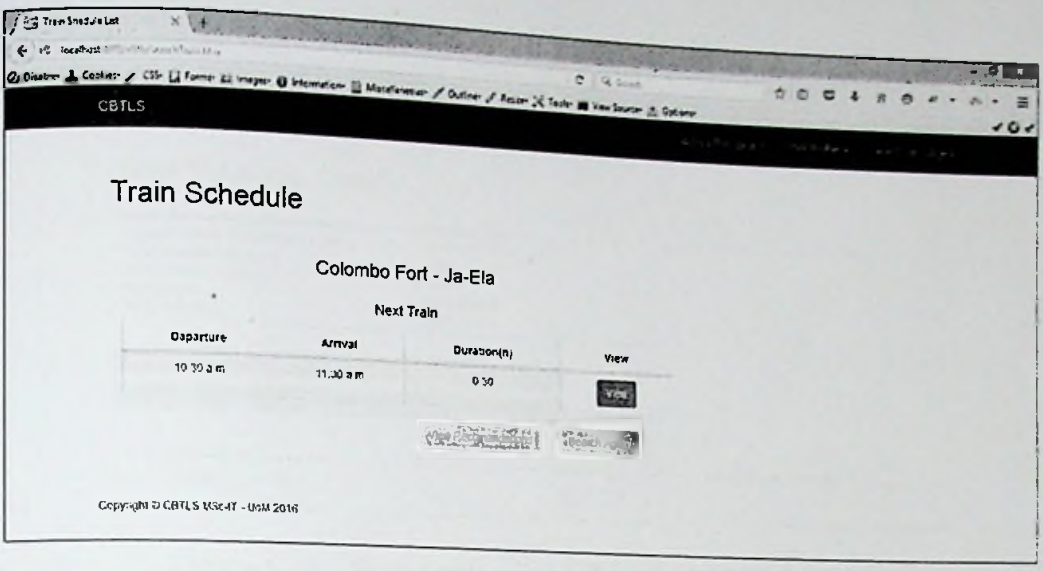


Figure 6.34 – Web application – Train schedule list UI

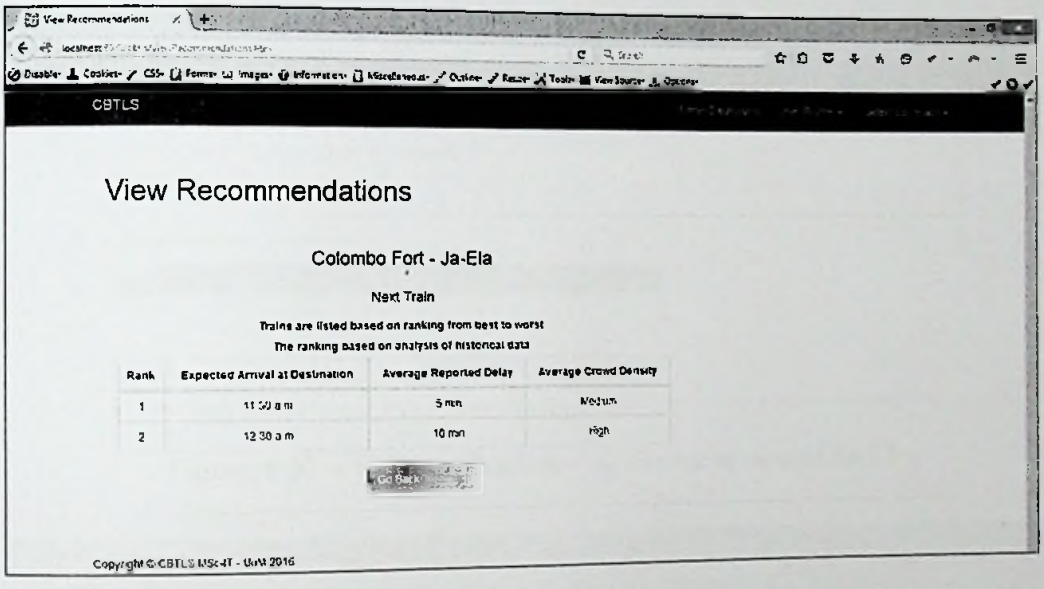


Figure 6.35 – Web application – View recommendations UI

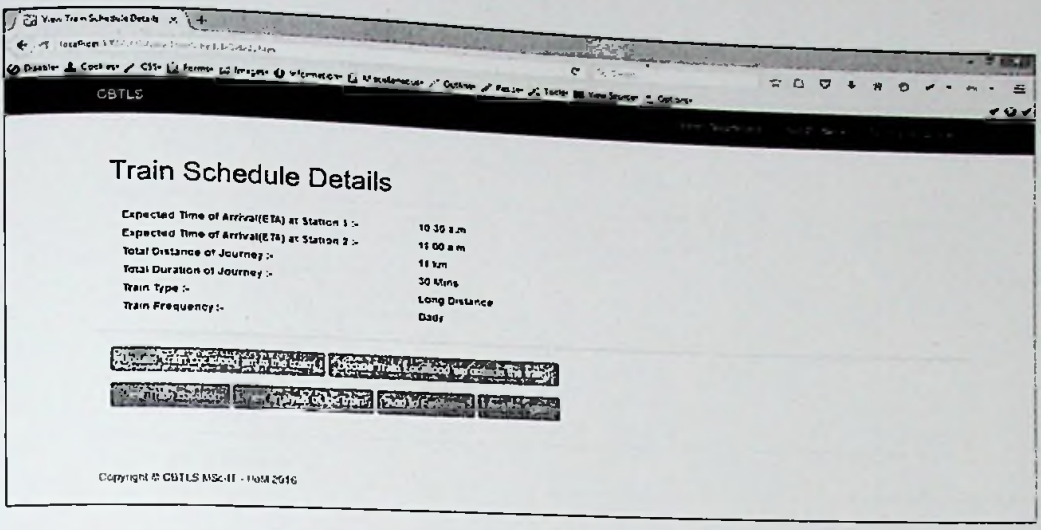


Figure 6.36 – Web application – Train schedule details UI

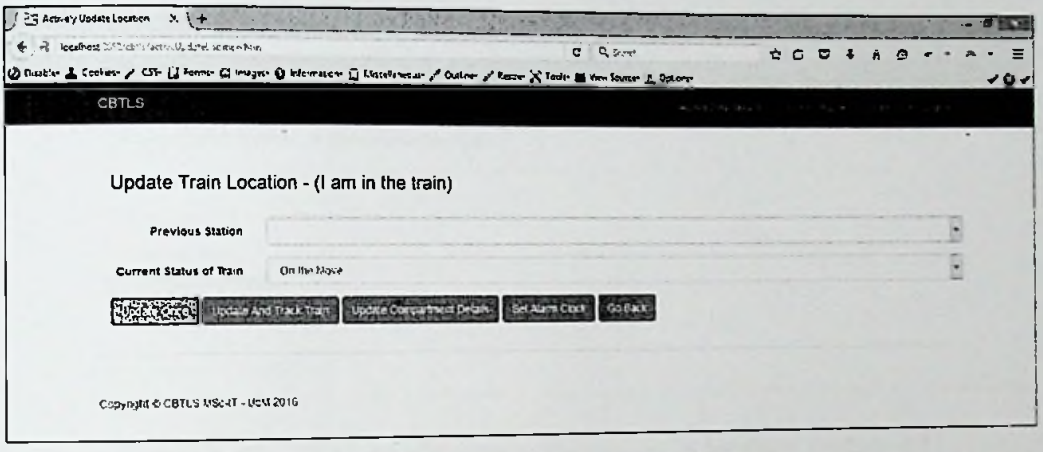


Figure 6.37 – Web application – Active location update UI

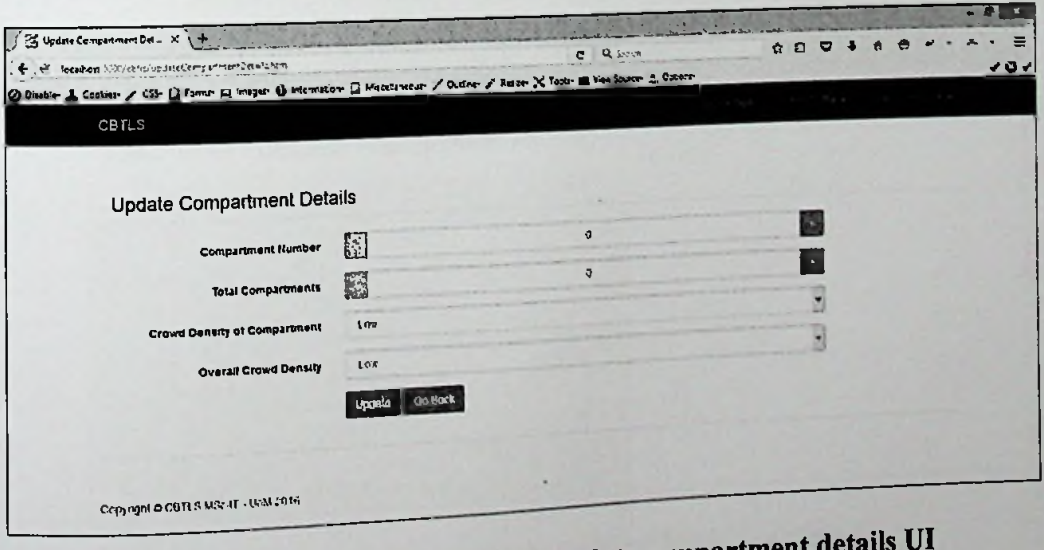


Figure 6.38 – Web application – Update compartment details UI

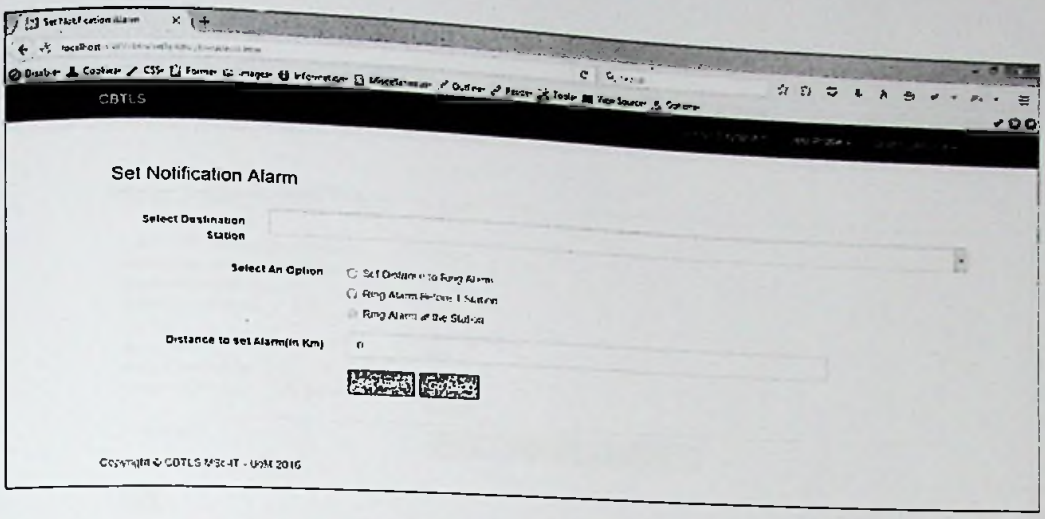


Figure 6.39 – Web application – Set alarm clock UI

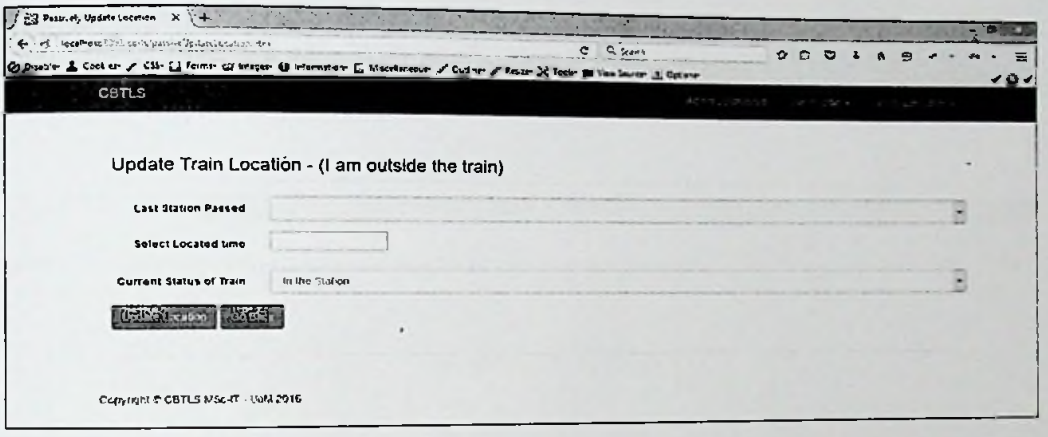


Figure 6.40 – Web application – Passive train location update UI

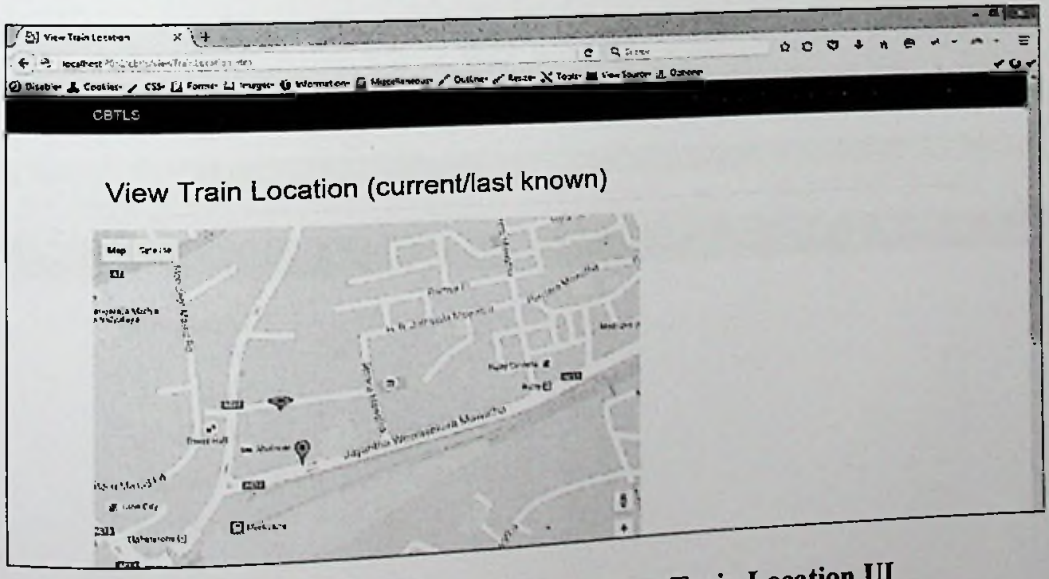


Figure 6.41 – Web application – View Train Location UI

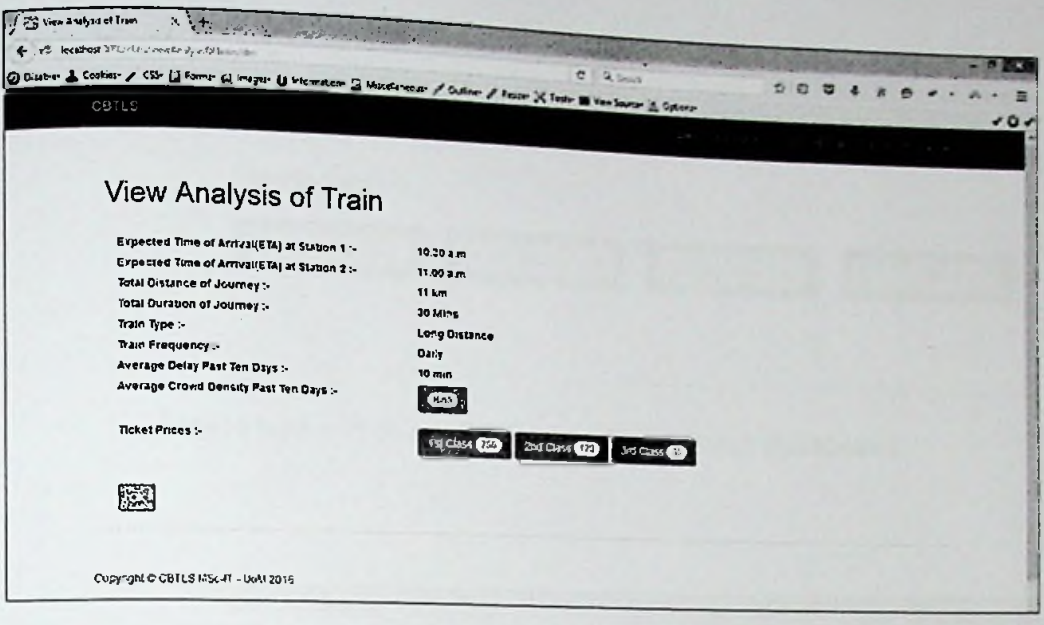


Figure 6.42 – Web application – View analysis of Train UI

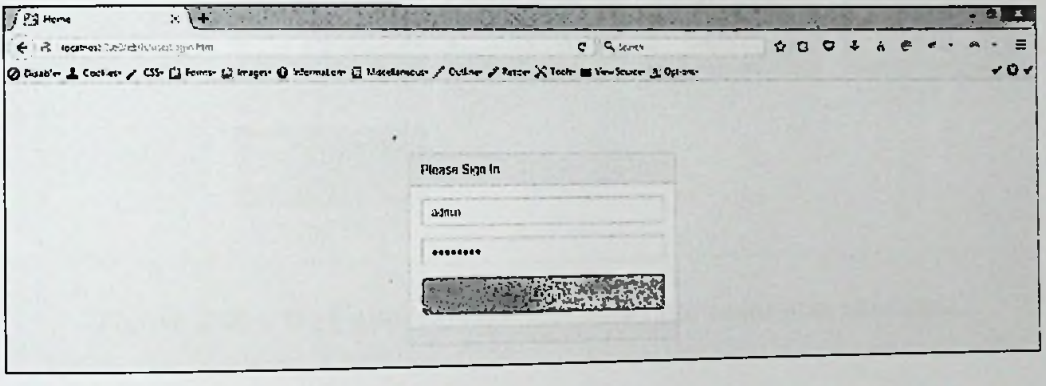


Figure 6.43 – Web application – Login UI

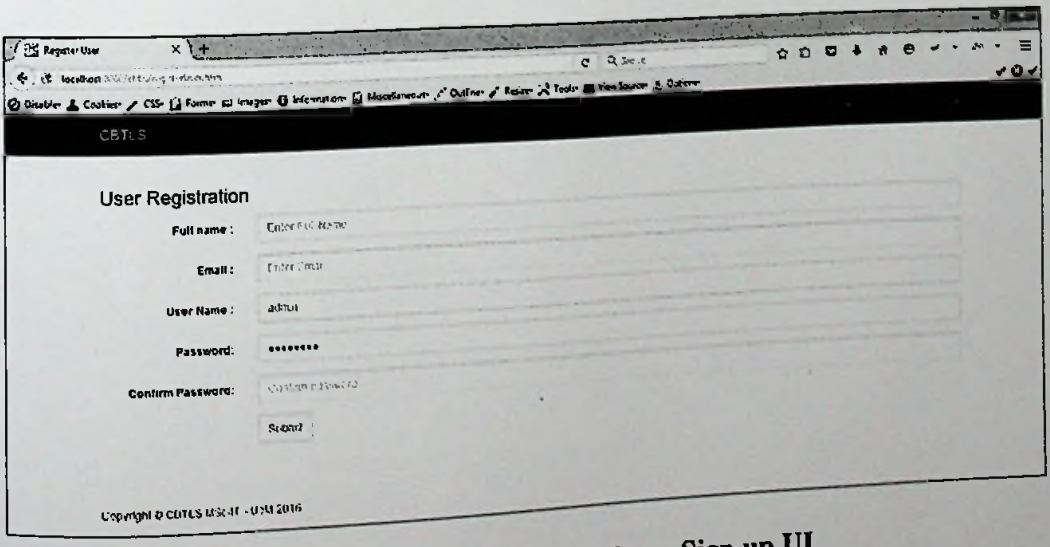


Figure 6.44 – Web application – Sign up UI

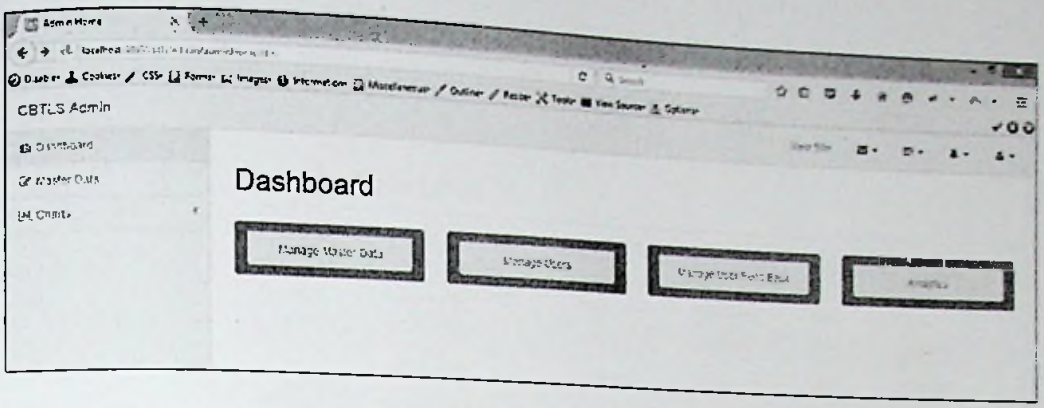


Figure 6.45 – Web application – Administrator Dashboard

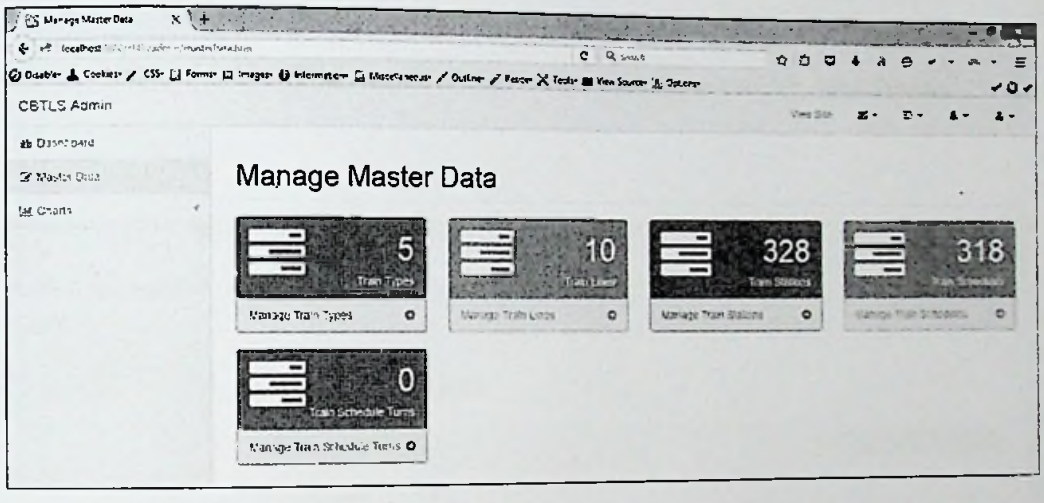


Figure 6.46 – Web application – Administrator manage master data

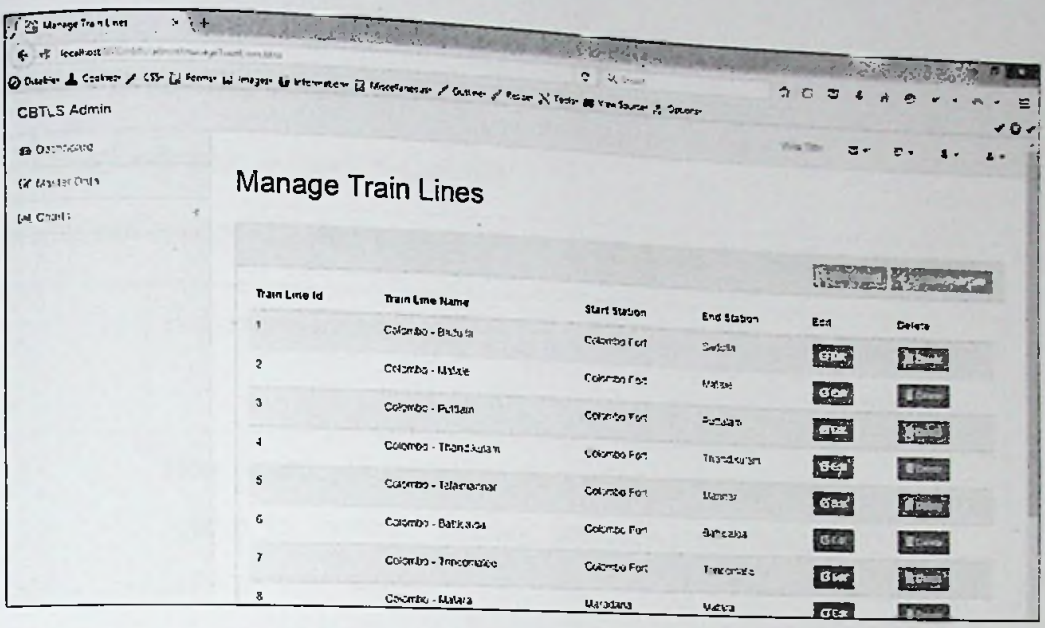


Figure 6.47 – Web application – Administrator master data details

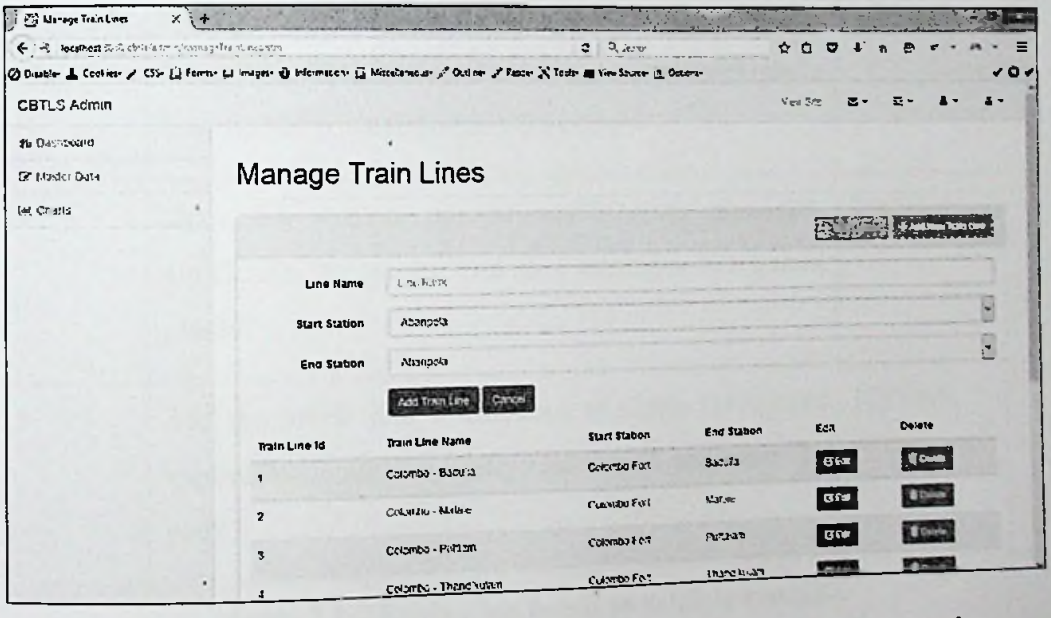


Figure 6.48 – Web application – Administrator master data modification

System Evaluation Forms

CBTLS Evaluation in terms of Usability

Predefined options- 1 - Very poor, 2 – Poor, 3 – Average, 4 - Good, 5 - Excellent

	Evaluation criteria	Selectable options
1	How would you categorize the navigation through the system in terms of entering, proceeding and finally leaving the system?	1/2/3/4/5
2	Could you get familiar with system functionality by yourself at the very first time	1/2/3/4/5
3	Based on the time it took for you to get execute a function available in system and get a successful result, how would you categorize the system?	1/2/3/4/5
4	Does the system provide confirmation messages, notification messages, and alert messages in required places?	1/2/3/4/5
5	Are the labels and instructions available throughout application could be easily understood and clear for you?	1/2/3/4/5

Table 7.1 – Evaluation forms to validate Usability

CBTLS Evaluation in terms of system functionality

Predefined options- 1 - Very poor, 2 – Poor, 3 – Average, 4 - Good, 5 - Excellent

	Evaluation criteria	Selectable options
1	Does the system provide adequate functionalities as a train schedule viewing system based on your experience?	1/2/3/4/5
2	Rate the system in terms of giving clear expected output for your actions	1/2/3/4/5
3	How would you rate the system for its actual behavior versus your expectation	1/2/3/4/5
4	When comparing with existing systems for the same purpose, how would you categorize this system	1/2/3/4/5
5	Are you satisfied with the functionality offered by this system?	1/2/3/4/5

Table 7.2 – Evaluation forms to validate System functionality

CBTLS Evaluation in terms of Overall Impression.

Predefined options- 1 - Very poor, 2 – Poor, 3 – Average, 4 - Good, 5 - Excellent

	Evaluation criteria	Selectable options
1	Does system provide the final result within your expected time scope with expected quality? Please categorize the system based on this	1/2/3/4/5
2	Did you encounter any inconvenience while accessing the system functionalities, how would you categorize system based on this feature	1/2/3/4/5
3	How would you categorize the system in terms of familiarity (the effort required to proceed through system as a returning user)?	1/2/3/4/5
4	How would you categorize the system based on its look and feel for a regular usage?	1/2/3/4/5
5	Please rate the system based on your overall satisfaction with this system	1/2/3/4/5

Table 7.3 – Evaluation forms to validate Overall Impression