Annexure-I

Database Management System

OuRCTC

(RAIWAY CATERING AND TOURISM COORPORATION)

Submitted in partial fulfillment of the requirements for the award of the degree of

master of Computer Applications

To

Guru Gobind Singh Indraprastha University, Delhi

Guide:
DEEPAK SHARMA SIR

Submitted By:
Pooja Gautam
Enrollment no- 02811404420

DEPARTMENT OF TRAINING AND TECHNICAL EDUCATION

BPIBS

BHAI PARMANAND INSTITUTE OF BUSINESS STUDIES

Annexure- II

Certificate

I ”Pooja Gautam”,certify that the Minar Project Report (MCA) entitled “OuRCTC” is done by me and it is an authentic work carried out by me. The matter embodied in this project work has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

Signature of the student:

Date:

Certified that the project report (MCA) entitled________________” done by the above student is completed under my guidance.

Signature of the Guide:
Date:
Name of the guide:
Designation:

Acknowledgement

We have made this report file on the project “**OuRCTC**”. We have tried our best to elucidate all the relevant detail to the topic to be included in the report, while in the beginning we have tried to give a general view about this topic. Our efforts have ended on a successful note. I express my sincere gratitude to **MR.DEEPAK SHARMA SIR**, for giving us this opportunity to develop a project

Annexure-lll

FORMAT FORTABLE OF CONTENTS

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 ACTORS

1.2 SOME SAMPLE QUERIES

CHAPTER 2: SYSTEM DESIGN

CHAPTER 3: PHYSICAL DESIGN

CHAPTER 4: IMPLEMENTATION AND RESULTS

TEAM MEMBERS
REQUIREMENTS

ABSTRACT

The Indian Railways (IR) carries about 5.5 lakhs passengers in reserved accommodation every day. The Computerised Passenger Reservation System (PRS) facilitates the booking and cancellation of tickets from any of the 4000 terminals (i.e. PRS booking window all over the countries). These tickets can be booked or cancelled for journeys commencing in any part of India and ending in any other part, with travel time as long as 72 hours and distance up to several thousand kilometres.

In the given project we will be developing a SQLite Database which will help users to find train details, enquire about trains running between given two stations, book tickets and know the exact rates of their tickets to the desired destination.

With the help of online booking people can book their tickets online through internet, sitting in their home by a single click of mouse.

The main objective of the project is management of the database of Railway System. This is done by creating database of the trains between various stations, user database, booking database and many more. The database is then connected to main program using interconnection of the program with the database using NodeJS.

To access this Railway Ticket Booking System Project , users have to register by giving their entire details such as their name, full address details, sex, age, date of birth, nationality, mobile number, email id. After successful registration, users will be provided with their login id and password. The Ticket Management System has applicants and administrators.

Ticket Booking Offices are located in various parts of the state and each office is looked after by administrators. Each administrator has a unique identity, name, address, start date of work at an office in particular location.

Chapter-1
Introduction

PROBLEM STATEMENT

Indian Railways (IR) is India's national railway system operated by the Ministry of Railways. It manages the fourth-largest railway network in the world by size, with 121,407 kilometres (75,439 mi) of total track over a 67,368-kilometre (41,861 mi) route. IR runs more than 20,000 passenger trains daily, on both long-distance and suburban routes, from 7,349 stations across India. The trains have a five-digit numbering system. In the freight segment, IR runs more than 9,200 trains daily.

Pseudo Indian Railway Catering and Tourism Corporation is a subsidiary of the Indian Railways that handles the catering, tourism and online ticketing operations of the Indian railways, with around 5,50,000 to 6,00,000 bookings everyday is the world's second busiest. It's tagline is "Lifeline of the nation".

It is known for changing the face of railway ticketing in India. It pioneered internet-based rail ticket booking through its website, as well as from the mobile phones via WiFi, GPRS.In addition to e-tickets, Indian Railways Catering and Tourism Corporation also offers I-tickets that are basically like regular tickets.

In the given project we will be developing a SQL Database which will help users to find train details, enquire about trains running between given two stations, book tickets and know the exact rates of their tickets to the desired destination.

With the help of online booking people can book their tickets online through internet, sitting in their home by a single click of mouse.

The main objective of the project is management of the database of Railway System. This is done by creating database of the trains between various stations,user database,booking database and many more. The database is then connected to main program using interconnection of the program with the database using Django

- **Actors**

People who interact with database:

- Admin
- User

- **SOME SAMPLE QUERIES**

Different actors have different access to the database:

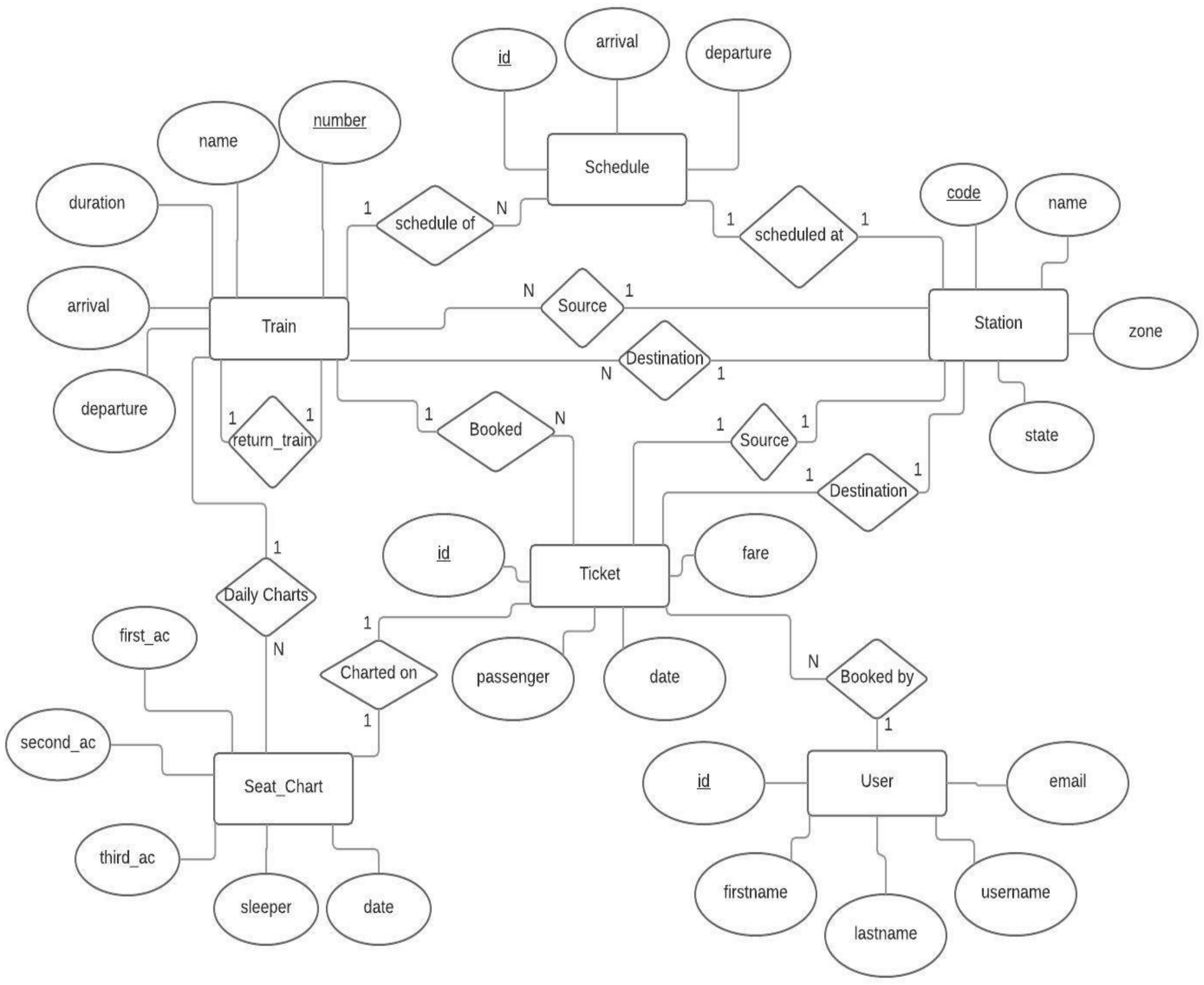
- A Admin can
- Check and edit the list of stations in a country
- Obtain and edit/add trains between stations.
- View and also edit the tickets of any user.
    - Change/view the schedule for a train
    - Generate chart for a specific train for a given journey
    - A User can
    - Get list of ticket that he/she has booked
    - Book multiple tickets.
    - Cancel the ticket he/she has booked

CHAPTER 2 SYSTEM DESIGN

2.1 LOGICAL DESIGN (CONCEPTUAL DESIGN)

2.2 SCHEMA DIAGRAM

Users:



username	Email	first_name	last_name	Password

Station:

code	State	name	zone	Address

Train:

number	Source	name	zone	arrival	Departure
return_train	Dest	duration_h	duration_m	type	Distance

Schedule:

train	Station	arrival	day	departure

Seat_Chart:

train	date	second_ac	third_ac	sleeper	first_ac

Ticket:

passenger	train	type	chart	user	source	dest	source_schedule	dest_schedule	date

2.3 ADVANCED LOGICAL DESIGN

2.3.1 NORMALIZATION TECHNIQUES

Every candidate key in a table determines all other attributes and no non-key attributes determine any attributes in the tables. Hence all tables are already in 3rd NF.

4th NF is not required since there are no repeated values in any of the tables.

2.3.1 GLOBAL SCHEMA

Users	
Attribute name	**Attribute size (type in bytes)**
username	char(20)
email	char(20)
first_name	char(10)
last_name	char(10)
password	char(20)

Train

Attribute name	**Attribute size (type in bytes)**
arrival	char(8)
source	char(10)

name	char(30)
zone	char(10)
number	char(15)

Stations	
Attribute name	**Attribute size (type in bytes)**
state	char(20)
code	char(10)
name	char(30)
zone	char(10)
address	char(50)

Schedule	
Attribute name	**Attribute size (type in bytes)**
arrival	char(8)
day	Int
train	char(15)
station	char(10)
id	Int
departure	char(8)

Ticket	
Attribute name	**Attribute size (type in bytes)**
passenger	char(20)
train	char(15)

type	char(2)
chart	Int
user	char(20)
source	char(10)
dest	char(10)
source_schedule	Int
dest_schedule	Int
date	Date
fare	Int

Seat_Chart	
Attribute name	**Attribute size (type in bytes)**
train	char(15)
first_ac	Int
second_ac	Int
third_ac	Int
sleeper	Int
date	Date

• QUERIES

- **<u>Queries to be executed:</u>**

- **User login.**

  SELECT * FROM Users WHERE username = input_name
AND password = password;

- **Search for trains.** SELECT * FROM trainDetails

- **Book Tickets.**

  INSERT INTO Ticket

```
(input_name,input_train,input_seat_type,input_chart,user.u
sername,input_source,in
put_dest,input_sourceSchedule,input_destSchedule,
input_date,fare)
```

- **View previous bookings** SELECT * FROM Ticket
  WHERE user = cookie_user;

- **Cancel Bookings**

```
DELETE FROM Ticket
WHERE user = cookie_user AND id = input_id;
```

- **VIEWS**

```
CREATE view trainDetails AS
SELECT t.name, t.number, s1.name, s2.name,
sc1.departure, sc2.arrival, sch.seats FROM Train t
JOIN Station s1
on
s1.code=input
_source JOIN
Station s2 on
s2.code=input
_dest
JOIN Schedule sc1 on sc1.station=s1.code
AND sc1.train=t.number JOIN Schedule sc2
on sc2.station=s2.code AND
sc2.train=t.number JOIN Seat_Chart sch on
t.number=sch.train AND
sch.date=input_date WHERE sc1.departure
< sc2.arrival
```

- **TRIGGERS**

- **For booking tickets:**

```
DROP
TRIGGER IF
EXISTS
book_ticket;
DELIMITE
R //
```

```
CREATE TRIGGER book_ticket
AFTER INSERT ON Ticket
FOR EACH ROW
BEGIN
update seat_chart
set seats=seats-1 WHERE train=NEW.train AND date=NEW.date
```

- **For cancelling tickets:**

```
DROP
TRIGGER IF EXISTS
cancel_ticket;
DELIMITER
//
CREATE TRIGGER cancel_ticket
AFTER DELETE ON Ticket FOR
EACH ROW BEGIN
update seat_chart
set seats=seats+1
WHERE
train=OLD.train AND date=OLD.date
```

Chapter 3 Physical Design

ASSUMPTIONS

Number of tuples in each relation

User	1000
Train	2000
Station	10000
Schedule	20000
Seat_Chart	60000
Ticket	80000

- ## STORAGE REQUIREMENTS: DISK PARAMETERS

Avg Seek Time
rotational delay(latency time)
block transfer time block
pointer size block size

Following are the assumptions which are considered for storage requirements:

- Fixed length records are considered for all relations.
- The delimiter for each field is length of the field
- Total number of records in respective relations (provided in below table).
- Block size is 1024 bytes.
- Record doesn't span over multiple blocks (this can be achieved by taking floor function during calculating number of records per block to restrict single record doesn't span over blocks).
- Block pointer(Bp) size is 4 bytes
- Average Seek Time(S) is 20 ms irrespective of any site.
- Average Disk rotation time (Latency) Time (L) is 10 ms irrespective of any site.
- Block transfer rate (Tr) is 0.5 ms irrespective of any site.
- **Blocking factor= ceil(Block size / Record size in bytes) • # no of blocks = ceil(# of records/ Blocking factor)**

Relation	**# of records**	**Record size in bytes**	**Blocking factor**	**# no. of blocks**
User	1000	80	12	84
Train	2000	123	8	250
Station	10000	120	8	1250
Schedule	20000	49	20	1000
Seat_Chart	60000	40	25	2400

Ticket	80000	100	10	8000

Total number of blocks used = 84+250+1250+1000+2400+8000
= 12,984 blocks

- **ACCESS METHODS:**

Considering the assumption we can calculate easily the size of single record (tuple) of every relation with the help of Schema. The above table gives the number of records in each relation, size of each record, blocking factor for a particular block of that relation and number of blocks required to store entire relation. Having records on secondary storage, if you want to access them faster, then you need indexing. If a database is frequently queried and it is too large then it is supposed to have index to increase performance. There are various indexes used in databases. Here, we consider the following indexing scheme: Primary Index, Clustered Index and Secondary index. Based on the query, we decide what type of indexing file.

Relation	**Indexing type**	**Indexing attribute(s)**	**Is a key?**
User	Primary	Username	Yes
Train	Primary	Number	Yes
Stations	Primary	Code	Yes
Schedule	Primary	Id	Yes
Seat_Chart	Primary	{ Train, Date }	Yes
Ticket	Primary	{ User, Train, Date }	Yes

The following table explains what is the disk block access time to extract particular record for all the relations.

Relation	# of records	# no of data blocks	Index size per record	# of index records per block	# no of index blocks	# no of block access with indexing	# no of block access with indexing
User	1000	84	24	42	24	84	6
Train	2000	250	19	53	38	250	6
Stations	10000	1250	14	73	137	1250	8
Schedule	20000	1000	8	128	157	1000	8
Seat_Cha rt	60000	2400	27	37	1622	2400	12
Ticket	80000	8000	47	21	3810	8000	13

# of index records per block= Block size / Index size per record
# no of index blocks =ceil(# of records / # of index records per block) # no of block access without indexing = # no of data blocks number of block accesses with indexing = ceil [log(# no of index blocks)] + 1 Indexing the data file definitely reduces the number of block accesses needed to find particular record from the data file. The complete statistics is shown in above table.

• TIMINGS

Disk access time = Average seek time + latency time + block transfer time
= 20 + 10 + 0.5
= 30.5 ms

Therefore, to access one random block and transfer it, the time is 30.5ms. If the blocks are consecutive seek time and latency time are not included. Also, there can be overhead delay and queuing delay.

CHAPTER 4
IMPLEMENTATION AND RESULTS

Login Form

- **Signup Form**
- **Home Page**
- **Select from Map**
- **Display Trains**
- **Display Available Seats**
- **Display Connecting Trains**
- **Booked Ticket History**

# Pseudo-RCTC

## Introduction

The Indian Railways (IR) carries about 5.5 lakhs passengers in reserved accommodation every day. The Computerised Passenger Reservation System (PRS) facilitates the booking and cancellation of tickets from any of the 4000 terminals (i.e. PRS booking window all over the countries). These tickets can be booked or cancelled for journeys commencing in any part of India and ending in any other part, with travel time as long as 72 hours and distance up to several thousand kilometres.

In the given project we will be developing a SQL Database which will help users to find train details, enquire about trains running between given two stations, book tickets and know the exact rates of their tickets to the desired destination.

## Implementation Details

- Users have a unique serial number (SN) used as primary key, name, email, phone, and adminship(decides if the user is an admin or not).

- Each station has a unique serial number(PK) and station name.

- All routes across the country are present here, each is identified by a serial number and the station which it passes through. One route contains more than one station.

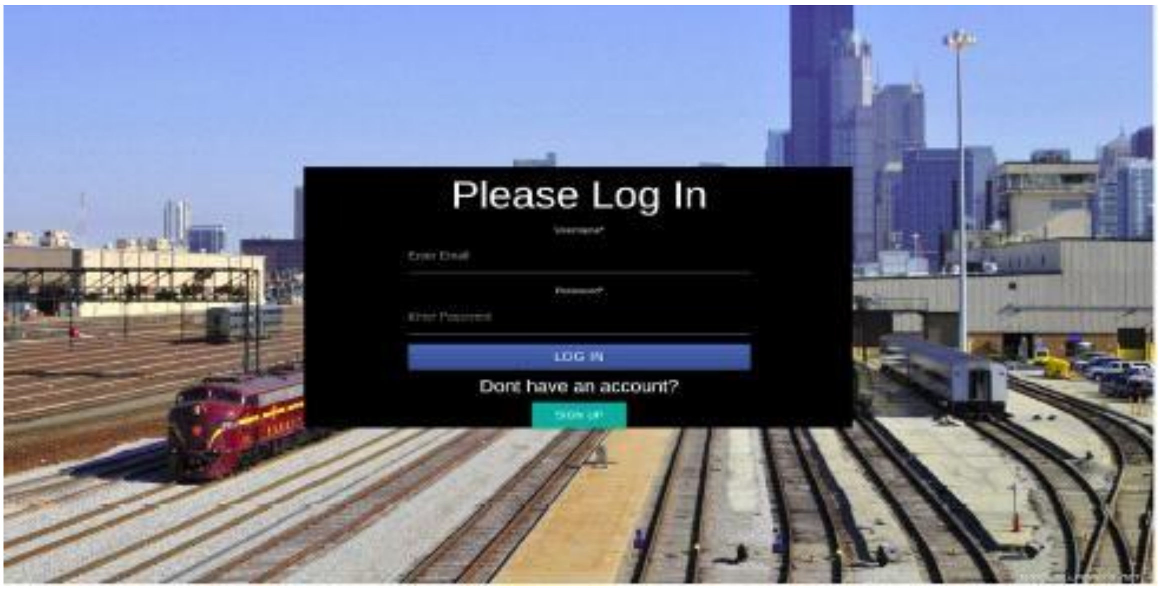
- Train entries have a unique train number(PK), name, source, destination, route, number of seats available, start time and end time. One train runs only on one route and has a single sou+rce and destination stations.

- Booking details of each ticket contain a unique serial number, user who booked the ticket, train for which the ticket is booked, user’s departure and arrival station. One user may have more than one tickets also many tickets can be booked for the same train. Each ticket can have only one departure and one arrival stations.

- Each region has a unique serial number(PK), name and the stations which fall under that region. One region may have more than one station.

## Project Images

#### 1. Login Form
<div align="center">
  <img
src="https://raw.githubusercontent.com/lPOOJA/OuRCTC/master/im1.png" />
</div>


Please Log In

Username*

Enter Email

Password*

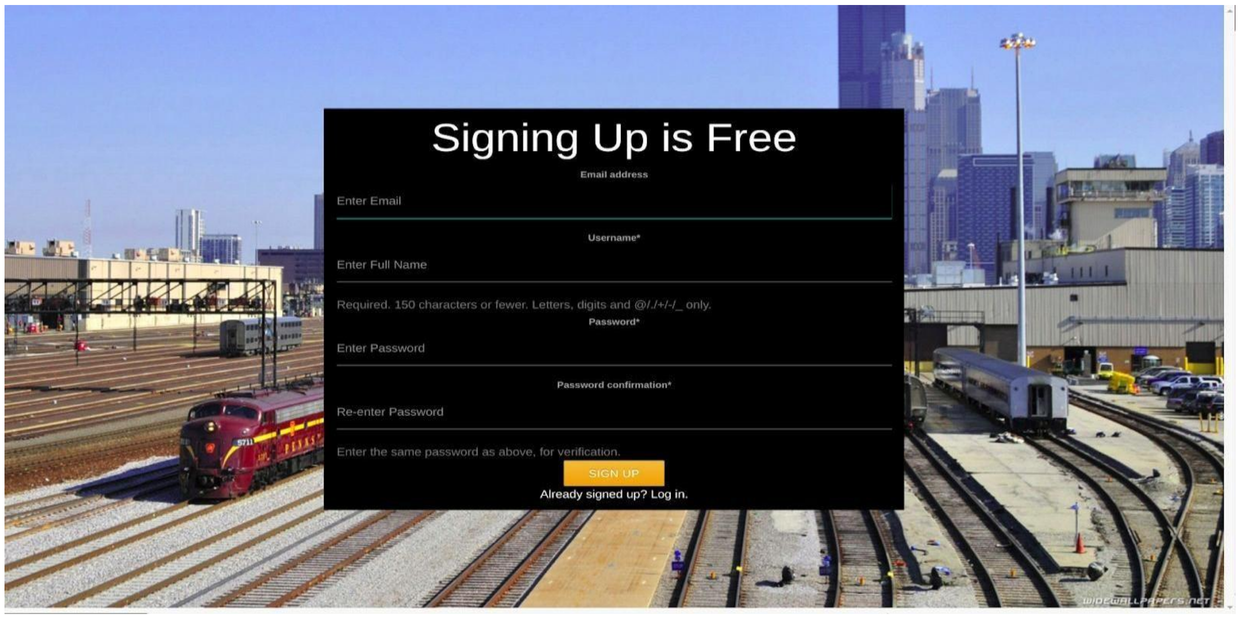
Enter Password

LOG IN

Dont have an account?

SIGN UP

#### 2. SignUp Form
<div align="center">
  <img
src="https://raw.githubusercontent.com/POOJAlP/OuRCTC/master/im2.png" />
</div>



Signing Up is Free

Email address

Enter Email

Username*

Enter Full Name

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password*

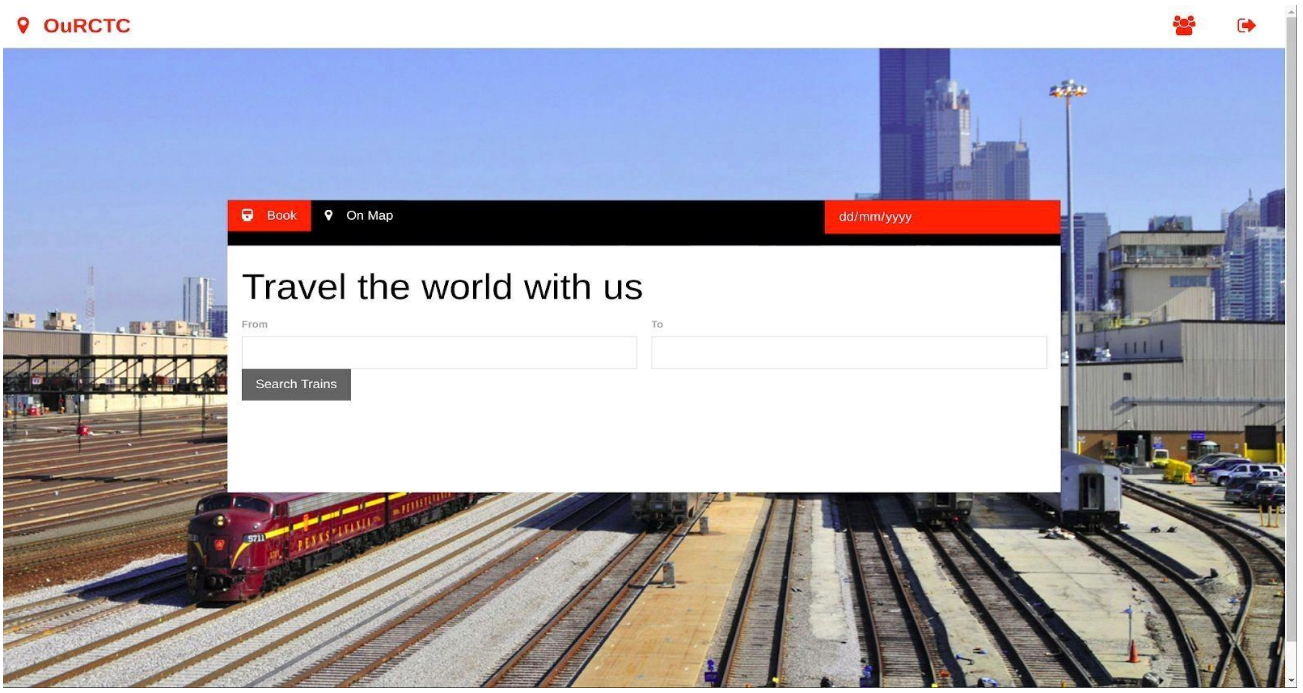
Enter Password

Password confirmation*

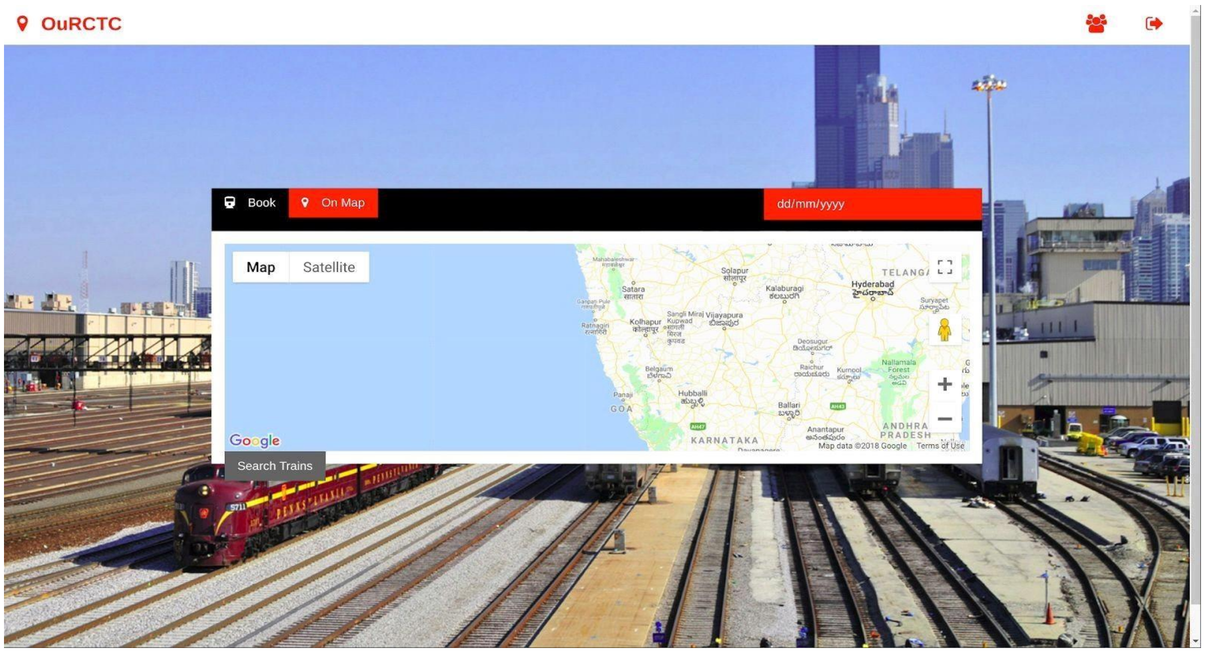
Re-enter Password

Enter the same password as above, for verification.

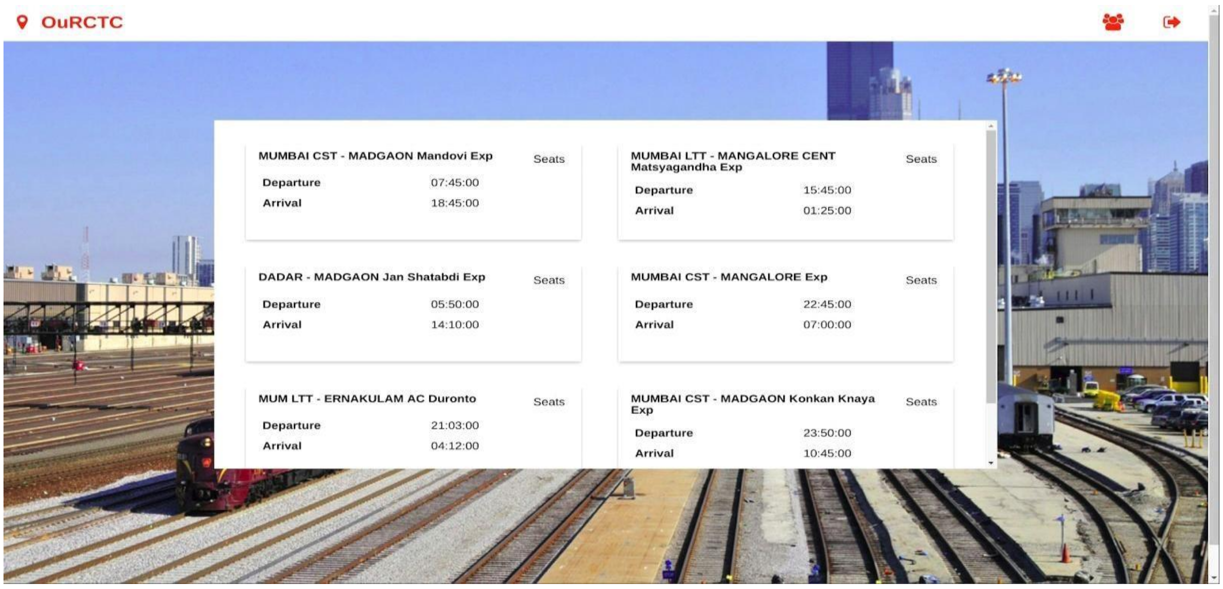
#### 3. Home Page
<div align="center">
  <img
src="https://raw.githubusercontent.com/Pooja515lP/OuRCTC/master/im3.png"
/>
</div>



#### 4. Select From Map
<div align="center">
  <img src="https://raw.githubusercontent.com/pooja/OuRCTC/master/im4.png" />
</div>



#### 5. Display Trains
<div align="center">
  <img
src="https://raw.githubusercontent.com/poojalP/OuRCTC/master/im5.png" />
</div>



MUMBAI CST - MADGAON Mandovi Exp — Seats
Departure 07:45:00
Arrival 18:45:00

MUMBAI LTT - MANGALORE CENT Matsyagandha Exp — Seats
Departure 15:45:00
Arrival 01:25:00

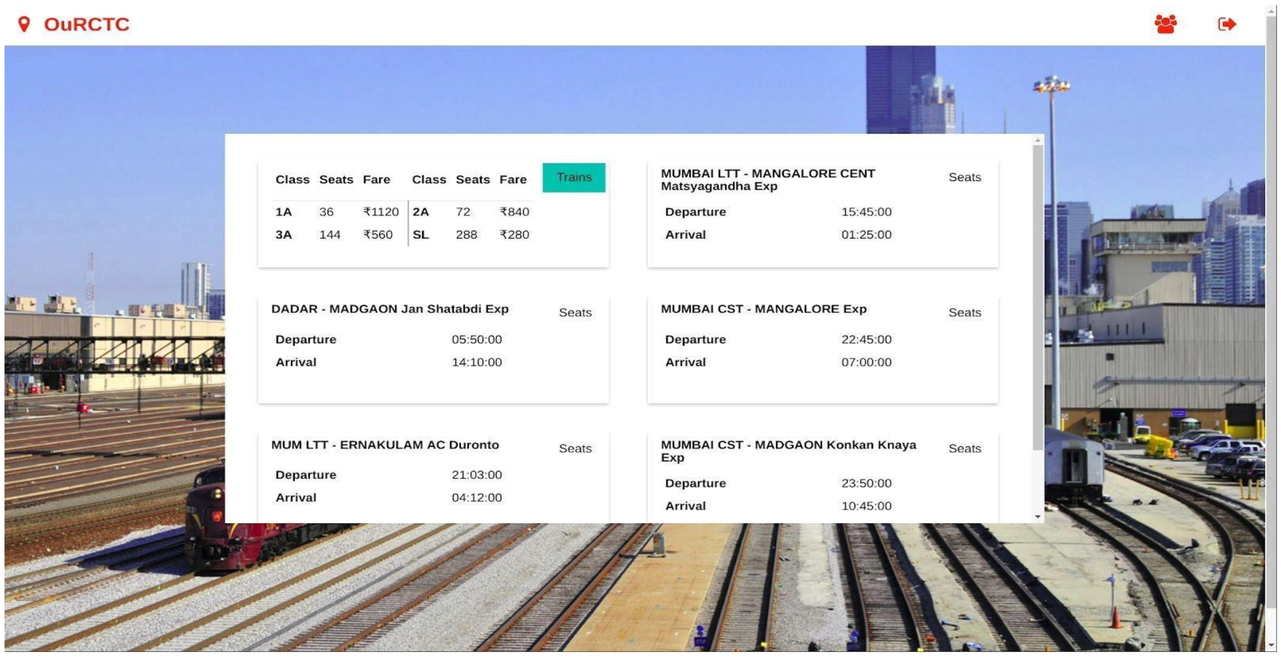
DADAR - MADGAON Jan Shatabdi Exp — Seats
Departure 05:50:00
Arrival 14:10:00

MUMBAI CST - MANGALORE Exp — Seats
Departure 22:45:00
Arrival 07:00:00

MUM LTT - ERNAKULAM AC Duronto — Seats
Departure 21:03:00
Arrival 04:12:00

MUMBAI CST - MADGAON Konkan Knaya Exp — Seats
Departure 23:50:00
Arrival 10:45:00

#### 6. Display Available Seats
<div align="center">
  <img
src="https://raw.githubusercontent.com/Pooja/OuRCTC/master/im6.png" />
</div>



Class	Seats	Fare	Class	Seats	Fare
1A	36	₹1120	2A	72	₹840
3A	144	₹560	SL	288	₹280

MUMBAI LTT - MANGALORE CENT Matsyagandha Exp — Seats
Departure 15:45:00
Arrival 01:25:00

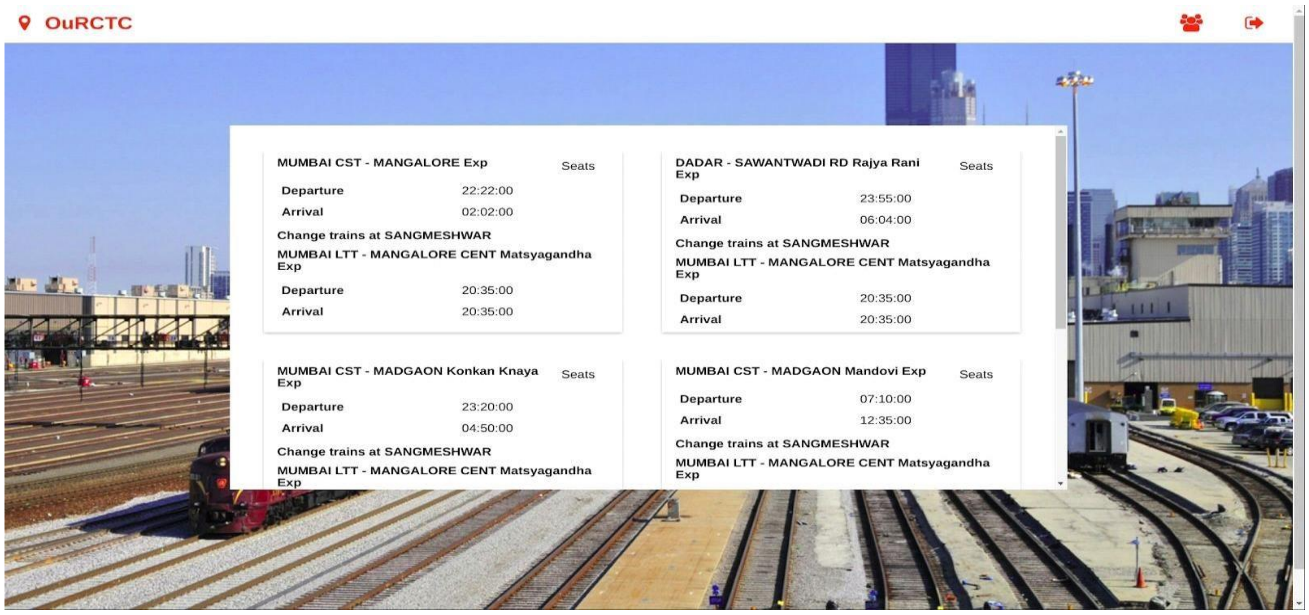
DADAR - MADGAON Jan Shatabdi Exp — Seats
Departure 05:50:00
Arrival 14:10:00

MUMBAI CST - MANGALORE Exp — Seats
Departure 22:45:00
Arrival 07:00:00

MUM LTT - ERNAKULAM AC Duronto — Seats
Departure 21:03:00
Arrival 04:12:00

MUMBAI CST - MADGAON Konkan Knaya Exp — Seats
Departure 23:50:00
Arrival 10:45:00

#### 7. Display Connecting Trains
<div align="center">
  <img src="https://raw.githubusercontent.com/Pooja/OuRCTC/master/im7.png" />
</div>



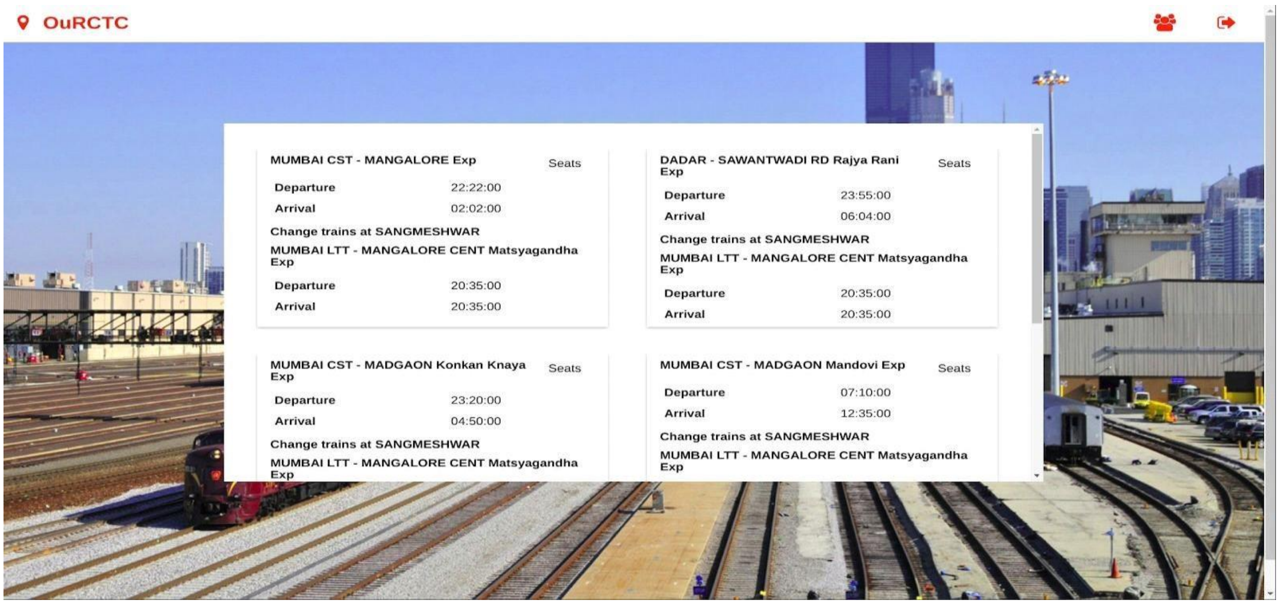
MUMBAI CST - MANGALORE Exp — Seats
Departure 22:22:00
Arrival 02:02:00
Change trains at SANGMESHWAR
MUMBAI LTT - MANGALORE CENT Matsyagandha Exp
Departure 20:35:00
Arrival 20:35:00

DADAR - SAWANTWADI RD Rajya Rani Exp — Seats
Departure 23:55:00
Arrival 06:04:00
Change trains at SANGMESHWAR
MUMBAI LTT - MANGALORE CENT Matsyagandha Exp
Departure 20:35:00
Arrival 20:35:00

MUMBAI CST - MADGAON Konkan Knaya Exp — Seats
Departure 23:20:00
Arrival 04:50:00
Change trains at SANGMESHWAR
MUMBAI LTT - MANGALORE CENT Matsyagandha Exp

MUMBAI CST - MADGAON Mandovi Exp — Seats
Departure 07:10:00
Arrival 12:35:00
Change trains at SANGMESHWAR
MUMBAI LTT - MANGALORE CENT Matsyagandha Exp

#### 8. Booked Ticket History
<div align="center">
  <img
src="https://raw.githubusercontent.com/Pooja/OuRCTC/master/im8.png" />
</div>



MUMBAI CST - MANGALORE Exp   Seats

Departure	22:22:00
Arrival	02:02:00

Change trains at SANGMESHWAR

MUMBAI LTT - MANGALORE CENT Matsyagandha Exp

Departure	20:35:00
Arrival	20:35:00

DADAR - SAWANTWADI RD Rajya Rani Exp   Seats

Departure	23:55:00
Arrival	06:04:00

Change trains at SANGMESHWAR

MUMBAI LTT - MANGALORE CENT Matsyagandha Exp

Departure	20:35:00
Arrival	20:35:00

MUMBAI CST - MADGAON Konkan Knaya Exp   Seats

Departure	23:20:00
Arrival	04:50:00

Change trains at SANGMESHWAR

MUMBAI LTT - MANGALORE CENT Matsyagandha Exp

MUMBAI CST - MADGAON Mandovi Exp   Seats

Departure	07:10:00
Arrival	12:35:00

Change trains at SANGMESHWAR

MUMBAI LTT - MANGALORE CENT Matsyagandha Exp

ACCOUNTS admin:

```
from django.contrib import admin

# Register your models here.
```

APPS:

```
from django.apps import AppConfig


class AccountsConfig(AppConfig):
    name = 'accounts'
```

FORMS:

```
from __future__ import unicode_literals
from django.contrib.auth.forms import AuthenticationForm from
django import forms
from crispy_forms.helper import FormHelper from crispy_forms.layout import
Layout, Div, Submit, HTML, Button, Row, Field from crispy_forms.bootstrap
import AppendedText, PrependedText, FormActions from authtools import forms
```

```
as authtoolsforms from django.contrib.auth import forms as authforms from
django.urls import reverse


class LoginForm(AuthenticationForm):
    remember_me = forms.BooleanField(required=False, initial=False)

    def __init__(self, *args, **kwargs):
super(LoginForm, self).__init__(*args, **kwargs)
self.helper = FormHelper()
        # self.fields["username"].widget.input_type = "email"  # ugly hack

        self.helper.layout = Layout(
            Field('username', placeholder="Enter Email", autofocus=""),
            Field('password', placeholder="Enter Password"),
Submit('sign_in', 'Log in',
                   css_class="btn btn-lg btn-primary btn-block"),
            )


class SignupForm(authtoolsforms.UserCreationForm):

    def __init__(self, *args, **kwargs):
        super(SignupForm, self).__init__(*args, **kwargs)
self.helper = FormHelper()
        self.fields["email"].widget.input_type = "email"  # ugly hack

        self.helper.layout = Layout(
            Field('email', placeholder="Enter Email", autofocus=""),
            Field('username', placeholder="Enter Full Name"),
            Field('password1', placeholder="Enter Password"),
            Field('password2', placeholder="Re-enter Password"),
            Submit('sign_up', 'Sign up', css_class="btn-warning"),
            )


class PasswordChangeForm(authforms.PasswordChangeForm):

    def __init__(self, *args, **kwargs):
        super(PasswordChangeForm, self).__init__(*args, **kwargs)
self.helper = FormHelper()

        self.helper.layout = Layout(
            Field('old_password', placeholder="Enter old password",
autofocus=""),
            Field('new_password1', placeholder="Enter new password"),
            Field('new_password2', placeholder="Enter new password (again)"),
            Submit('pass_change', 'Change Password', css_class="btn-warning"),
            )
```

MODELS:

```
from django.db import models

# Create your models here.
```

TESTS:

```
from django.test import TestCase

# Create your tests here.
```

VIEWS:

```
from __future__ import unicode_literals
from django.shortcuts import render, get_object_or_404, redirect,reverse
from django.urls import reverse_lazy from django.views import generic
from django.contrib.auth import get_user_model from django.contrib
import auth from django.contrib import messages from authtools import
views as authviews from braces import views as bracesviews from
django.conf import settings from . import forms

User = get_user_model()


class LoginView(bracesviews.AnonymousRequiredMixin, authviews.LoginView):
    template_name = "accounts/login.html"
form_class = forms.LoginForm


    def form_valid(self, form):
        r = super(LoginView, self).form_valid(form)
remember_me = form.cleaned_data.get('remember_me')        if
remember_me is True:
            ONE_MONTH = 30*24*60*60
            expiry = getattr(settings, "KEEP_LOGGED_DURATION", ONE_MONTH)
self.request.session.set_expiry(expiry)

        return redirect('book:home')


class LogoutView(authviews.LogoutView):
    url = reverse_lazy('book:home')


class SignUpView(bracesviews.AnonymousRequiredMixin,
bracesviews.FormValidMessageMixin,
generic.CreateView):     form_class = forms.SignupForm
    model = User
    template_name = 'accounts/signup.html'
success_url = reverse_lazy('book:home')
```

```
    form_valid_message = "You're signed up!"

    def form_valid(self, form):
        r = super(SignUpView, self).form_valid(form)
username = form.cleaned_data["username"]        password
= form.cleaned_data["password1"]
        user = auth.authenticate(username=username, password=password)
auth.login(self.request, user)
        return r


class PasswordChangeView(authviews.PasswordChangeView):
form_class = forms.PasswordChangeForm     template_name =
'accounts/password-change.html'     success_url =
reverse_lazy('book:home')

    def form_valid(self, form):
        form.save()
        messages.success(self.request,
                         "Your password was changed, "
                         "hence you have been logged out. Please relogin")
return super(PasswordChangeView, self).form_valid(form)
```

BOOK:

Admin

```
from django.contrib import admin from
.models import *

# Register your models here.
admin.site.register(Station) admin.site.register(Train)
admin.site.register(Schedule)
admin.site.register(Seat_Chart)
admin.site.register(Ticket          )
```

Apps:

```
from django.apps import AppConfig


class BookConfig(AppConfig):
    name = 'book'
```

Models:

```
from django.db import models
from django.contrib.auth import get_user_model

User= get_user_model()


# Create your models here.
```

```
class Station(models.Model):
    state = models.CharField("State",max_length=20, null=True)     code
= models.CharField("Code",max_length=10,primary_key=True)
name = models.CharField("Name",max_length=30)     zone =
models.CharField("Zone",max_length=10, null=True)     address =
models.CharField("Address",max_length=50, null=True)

    def __str__(self):
return self.name

class Train(models.Model):     arrival =
models.CharField("Arrival",max_length=8, null=True)
    source = models.ForeignKey(Station, on_delete=models.SET(None), related_name="train_source")
name = models.CharField("Name", max_length=30)     zone = models.CharField("Zone",
max_length=10, null=True)     number = models.CharField("Number",
max_length=15,primary_key=True)     departure = models.CharField("Departure",max_length=8,
null=True)     return_train = models.CharField("Return Train",max_length=15, null=True)
    dest = models.ForeignKey(Station, on_delete=models.SET(None), related_name="train_dest")
duration_h = models.IntegerField("Duration Hours", null=True)     duration_m =
models.IntegerField("Duration Minutes", null=True)     type =
models.CharField("Type",max_length=5, null=True)     distance =
models.IntegerField("Distance", null=True)

    def __str__(self):
return self.name


class Schedule(models.Model):     arrival =
models.CharField("Arrival", max_length=8, null=True)     day =
models.IntegerField("Day", null=True)
    train = models.ForeignKey(Train, on_delete=models.CASCADE, related_name="train_schedule")
station = models.ForeignKey(Station, on_delete=models.CASCADE,
related_name="station_schedule")
    id = models.IntegerField("id",primary_key=True)
    departure = models.CharField("Departure", max_length=8, null=True)

    def __str__(self):
        return str(self.train) +" at "+str(self.station)


class Seat_Chart(models.Model):
    train = models.ForeignKey(Train, on_delete=models.CASCADE, related_name="train_chart")
first_ac = models.IntegerField("1st AC")     second_ac = models.IntegerField("2nd AC")
third_ac = models.IntegerField("3rd AC")     sleeper = models.IntegerField("Sleeper")     date =
models.DateField("Date")

    def get1A(self):        return self.first_ac -
self.chart_tickets.all().filter(type="1A").count()

    def get2A(self):
        return self.second_ac - self.chart_tickets.all().filter(type="2A").count()
    def get3A(self):
        return self.third_ac - self.chart_tickets.all().filter(type="3A").count()

    def getSL(self):
        return self.sleeper - self.chart_tickets.all().filter(type="SL").count()

    def __str__(self):
        return str(self.train) +" on "+str(self.date)

class Ticket(models.Model):     passenger =
models.CharField("Name",max_length=20)
    train = models.ForeignKey(Train, on_delete=models.CASCADE, related_name="train_tickets")
type = models.CharField("Type",max_length=2)
    chart = models.ForeignKey(Seat_Chart, on_delete=models.CASCADE,
related_name="chart_tickets")
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="tickets")
source = models.ForeignKey(Station, on_delete=models.CASCADE,
related_name="source_tickets")
    dest = models.ForeignKey(Station, on_delete=models.CASCADE, related_name="dest_tickets")
source_schedule = models.ForeignKey(Schedule, on_delete=models.CASCADE,
related_name="source_schedule_tickets")
    dest_schedule = models.ForeignKey(Schedule, on_delete=models.CASCADE,
related_name="dest_schedule_tickets")     date = models.DateField("Date")
fare = models.IntegerField("Fare")

    def __str__(self):
        return str(self.passenger) +" on "+str(self.date)+" in "+str(self.train)

    def calculateFare(self):
        factor=1
if(self.type=="1A"):
            factor=20
elif(self.type=="2A"):
            factor=15        elif
(self.type == "3A"):
            factor = 10
else:
            factor = 5
        self.fare = (self.dest_schedule.pk - self.source_schedule.pk)*factor
```

Tests:

```
from django.test import TestCase

# Create your tests here.
```

Urls:

```
from django.urls import path

from . import views

app_name = 'book'
```

```
urlpatterns = [
   path('',views.homeView,name="home"),
path('search',views.searchView,name="search"),     path('complexSearch/<source>-
<dest>/<date>',views.complexSearchView,name="complexSearch"),
path('mapSearch>',views.MapSearchView.as_view(),name="mapSearch"),
path('book/<chart>/<sourceSchedule>-
<destSchedule>/<type>/<date>',views.bookView,name="book"),
   path('book/<chart1>-<chart2>/<sourceSchedule>-<commonSchedule1>/<commonSchedule2>-
<destSchedule>/<type>/<date>',views.complexBookView,name="complexBook"),
path('confirm/<chart>/<sourceSchedule>-
<destSchedule>/<type>/<date>',views.confirmTicketView,name="confirm"),
path('confirm/<chart1>-<chart2>/<sourceSchedule>-
<commonSchedule1>/<commonSchedule2><destSchedule>/<type>/<date>',views.complexConfirmT
icketView,name="complexConfirm"),     path('profile', views.profileView, name="profile"),
   path('cancel/<pk>', views.cancelTicket.as_view(), name="cancel"),


]
```

Views:

```
from django.shortcuts import render,redirect from
django.urls import reverse_lazy
from django.views.generic.edit import CreateView,UpdateView,DeleteView
from django.contrib.auth.decorators import login_required
from django.contrib.auth.mixins import LoginRequiredMixin
from rest_framework import authentication, permissions
from rest_framework.views import APIView from
rest_framework.response import Response from .models
import * from dateutil import parser import json, time

# Create your views here.

@login_required(login_url="/login") def
homeView(request):
   stations=Station.objects.all()
context={
       "stations" : stations,
       "st" : stations[0:4]
   }
   return render(request,'book/home.html',context)

@login_required(login_url="/login")
def searchView(request):
   source = Station.objects.get(pk=request.POST['source'])
dest = Station.objects.get(pk=request.POST['dest'])     date
= request.POST['journey_date']     sourceTrains = []     for
s in source.station_schedule.all():
       sourceTrains.append(s.train)
destTrains = []     for s in
dest.station_schedule.all():
       destTrains.append(s.train)
   allTrains=list(set(sourceTrains) & set(destTrains))
```

```
    trains=[]
sourceSchedules=[]
destSchedules=[]
scheduleCharts=[]
fares=[]     for t in
allTrains:
        departing_station = t.train_schedule.get(station=source)
arriving_station = t.train_schedule.get(station=dest)         if
departing_station.pk < arriving_station.pk:
            scheduleCharts.append(Seat_Chart.objects.get(date=parser.parse(date),train=t))
trains.append(t)
            sourceSchedules.append(departing_station)
destSchedules.append(arriving_station)             fare={}
            fare["1A"]=(arriving_station.pk - departing_station.pk)*20
fare["2A"]=(arriving_station.pk - departing_station.pk)*15
fare["3A"]=(arriving_station.pk - departing_station.pk)*10
fare["SL"]=(arriving_station.pk - departing_station.pk)*5             fares.append((fare))

    schedules=zip(trains,sourceSchedules,destSchedules,scheduleCharts,fares)
data={
        "source": source,
        "dest": dest,
        "schedules":schedules,
        "date": date,
    }
    return render(request,'book/trainSearch.html',data)

@login_required(login_url="/login") def
complexSearchView(request,source,dest,date):
    source = Station.objects.get(pk=source)
dest = Station.objects.get(pk=dest)
sourceTrains = []     for s in
source.station_schedule.all():
        sourceTrains.append(s.train)
destTrains = []     for s in
dest.station_schedule.all():
        destTrains.append(s.train)
    # allTrains=list(set(sourceTrains) & set(destTrains))

    trains1 = [] trains2
    = []
    sourceSchedules =
    []
    commonSchedule
    s1 = []
    commonSchedule
    s2 = []
    destSchedules = []
    scheduleCharts1 =
    []
    scheduleCharts2 =
    []
    fares=[]
    for ts in sourceTrains:
sourceSchedule=source.station_schedule.get(train=ts)
source_stations = []          for a in
ts.train_schedule.filter(id__gte=sourceSchedule.id):
            source_stations.append(a.station)          for td in
destTrains:               destSchedule =
dest.station_schedule.get(train=td)               dest_stations = []
for a in td.train_schedule.filter(id__lte=destSchedule.id):
dest_stations.append(a.station)               common =
list(set(source_stations) & set(dest_stations))               for c in
common:
                tempDestSchedule=c.station_schedule.get(train=ts)
tempSourceSchedule=c.station_schedule.get(train=td)
if(tempDestSchedule.arrival < tempSourceSchedule.departure):
trains1.append(ts)
                    trains2.append(td)
                    commonSchedules1.append(tempDestSchedule)
commonSchedules2.append(tempSourceSchedule)
sourceSchedules.append(sourceSchedule)                    destSchedules.append(destSchedule)
                    c1=Seat_Chart.objects.get(date=parser.parse(date),train=ts)
c2=Seat_Chart.objects.get(date=parser.parse(date),train=td)
scheduleCharts1.append(c1)                    scheduleCharts2.append(c2)
# fares.append(None)                    fare = {}
                    fare["1A"] = (destSchedule.pk - tempSourceSchedule.pk + tempDestSchedule.pk -
sourceSchedule.pk) * 20
                    fare["2A"] = (destSchedule.pk - tempSourceSchedule.pk + tempDestSchedule.pk -
sourceSchedule.pk) * 15
                    fare["3A"] = (destSchedule.pk - tempSourceSchedule.pk + tempDestSchedule.pk -
sourceSchedule.pk) * 10
                    fare["SL"] = (destSchedule.pk - tempSourceSchedule.pk + tempDestSchedule.pk -
sourceSchedule.pk) * 5
                    fares.append((fare))
                    break


schedules=zip(trains1,trains2,sourceSchedules,commonSchedules1,commonSchedules2,destSchedule
s,scheduleCharts1,scheduleCharts2,fares)     data={
        "source": source,
        "dest": dest,
        "schedules":schedules,
        "date": date,
    }
    return render(request,'book/connectingTrainSearch.html',data)


class MapSearchView(LoginRequiredMixin, APIView):
login_url =  '/login'
    authentication_classes = (authentication.SessionAuthentication,)
permission_classes = (permissions.IsAuthenticated,)
```

```
    def get(self, request, format=None):
date=request.GET["date"]
sources = request.GET["source"]
dests = request.GET["dest"]         for s
in sources:
            source=Station.objects.filter(name__iexact=sources)
print(s)             if source.count():
                break
for d in dests:
            dest=Station.objects.filter(name__iexact=dests)
if dest.count():
                break

        data={
"date":date,
            "source":source[0].code,
            "dest": dest[0].code,
        }

        return Response(data)

@login_required(login_url="/login") def
bookView(request,chart,sourceSchedule,destSchedule,type,date):
chart = Seat_Chart.objects.get(pk=chart)
    train = chart.train
    sourceSchedule=Schedule.objects.get(pk=sourceSchedule)
destSchedule=Schedule.objects.get(pk=destSchedule)
source = sourceSchedule.station     dest =
destSchedule.station
    data = {
"train": train,
        "chart": chart,
        "sourceSchedule": sourceSchedule,
        "destSchedule": destSchedule,
        "source":source,
        "dest":dest,
        "type":type,
        "date":date,
    }
    return render(request,'book/booking.html',data)

@login_required(login_url="/login") def
complexBookView(request,chart1,chart2,sourceSchedule,commonSchedule1,commonSchedule2,dest
Schedule,type,date):
    chart1 = Seat_Chart.objects.get(pk=chart1) chart2
    = Seat_Chart.objects.get(pk=chart2) train1 =
    chart1.train     train2 = chart2.train
    sourceSchedule=Schedule.objects.get(pk=sourceSchedule)
commonSchedule1=Schedule.objects.get(pk=commonSchedule1)
commonSchedule2=Schedule.objects.get(pk=commonSchedule2)
destSchedule=Schedule.objects.get(pk=destSchedule)     source =
sourceSchedule.station     dest = destSchedule.station
```

```
    data = {
"train1": train1,
        "train2": train2,
        "chart1": chart1,
        "chart2": chart2,
        "sourceSchedule": sourceSchedule,
        "commonSchedule1": commonSchedule1,
        "commonSchedule2": commonSchedule2,
        "destSchedule": destSchedule,
        "source":source,
        "dest":dest,
        "type":type,
        "date":date,
    }
    return render(request,'book/complexBooking.html',data)

@login_required(login_url="/login") def
confirmTicketView(request,chart,sourceSchedule,destSchedule,type,date):
chart=Seat_Chart.objects.get(pk=chart)
    train=chart.train
    sourceSchedule=Schedule.objects.get(pk=sourceSchedule)
destSchedule=Schedule.objects.get(pk=destSchedule)
source = sourceSchedule.station     dest =
destSchedule.station     user= request.user
seats=int(request.POST["seats"])     for i in range(seats):
        name=request.POST.get("name"+str(i))
b=Ticket()
        b.passenger=name
        b.train=train
        b.type=type
        b.chart=chart
        b.user=user
        b.source=source
        b.dest=dest
        b.source_schedule=sourceSchedule
        b.dest_schedule=destSchedule
        b.date = date
        b.calculateFare()
        b.save()


    data = {
"train": train,
        "sourceSchedule": sourceSchedule,
        "destSchedule": destSchedule,
        "source":source,
        "dest":dest,
        "type":type
    }
    return render(request, 'book/home.html')
```

```
@login_required(login_url="/login") def
complexConfirmTicketView(request,chart1,chart2,sourceSchedule,commonSchedule1,commonSched
ule2,destSchedule,type,date):     chart1 = Seat_Chart.objects.get(pk=chart1)     chart2 =
Seat_Chart.objects.get(pk=chart2)
    train1 = chart1.train
train2 = chart2.train
    sourceSchedule = Schedule.objects.get(pk=sourceSchedule)
commonSchedule1 = Schedule.objects.get(pk=commonSchedule1)
commonSchedule2 = Schedule.objects.get(pk=commonSchedule2)
    destSchedule = Schedule.objects.get(pk=destSchedule)
source = sourceSchedule.station     dest =
destSchedule.station     user= request.user
seats=int(request.POST["seats"])     for i in range(seats):
        name=request.POST.get("name"+str(i))
b=Ticket()
        b.passenger=name
        b.train=train1
        b.type=type
        b.chart=chart1
        b.user=user
        b.source=source
        b.dest=dest
        b.source_schedule=sourceSchedule
        b.dest_schedule=commonSchedule1
        b.date = date
        b.calculateFare()
        b.save()

        b = Ticket()
        b.passenger = name
        b.train = train2
        b.type = type
        b.chart = chart2
        b.user = user
        b.source = source
        b.dest = dest
        b.source_schedule = commonSchedule2
        b.dest_schedule = destSchedule
        b.date = date
        b.calculateFare()
        b.save()


    data = {
        "train1": train1,
        "train2": train2,
        "chart1": chart1,
        "chart2": chart2,
        "sourceSchedule": sourceSchedule,
        "commonSchedule1": commonSchedule1,
        "commonSchedule2": commonSchedule2,
        "destSchedule": destSchedule,
        "source":source,
        "dest":dest,
        "type":type,
        "date":date,
    }
    return render(request, 'book/home.html')

@login_required(login_url="/login")
def profileView(request):
user=request.user
booked=user.tickets.all()     data={
        "booked":booked,
    }
    return render(request, 'book/profile.html',data)

class cancelTicket(LoginRequiredMixin, DeleteView):
login_url = '/login'     model = Ticket
    success_url = reverse_lazy('book:profile')

    def dispatch(self, request, *args, **kwargs):         pk=kwargs['pk']         if
request.user != Ticket.objects.get(pk=pk).user:             return redirect('book:home')
if request.method.lower() in self.http_method_names:             handler = getattr(self,
request.method.lower(), self.http_method_not_allowed)         else:
            handler = self.http_method_not_allowed
return handler(request, *args, **kwargs)
```

Indian Railways Data

This repository has Indian Railways data that [Sanjay](https://twitter.com/sanjaybhangar) and [Sajjad](https://twitter.com/geohacker) have been gathering for a few months. Read [more here](http://sajjad.in/2016/08/gathering-indian-railways-data/).

There are three JSON files:

### Stations
[GeoJSON](http://geojson.org/) FeatureCollection, each Feature is a Station and looks like:

```json
{
    "geometry": {
        "type": "Point",
        "coordinates": [75.4516454, 27.2520587]
    },
    "type": "Feature",
    "properties": {
        "state": "Rajasthan",
        "code": "BDHL",
        "name": "Badhal",
        "zone": "NWR",
        "address": "Kishangarh Renwal, Rajasthan"
    }
}
```

### Trains
GeoJSON FeatureCollection, each Feature is a Train and looks like:

```json
{
    "geometry": {
        "type": "LineString",
        "coordinates": [
            [72.89173899999999, 19.070320000000002],
            [78.2266994458, 26.0352337224],
            [78.18700399999999, 26.145594],
            [78.18229199999999, 26.216483]
        ]
    },
    "type": "Feature",
    "properties": {
        "third_ac": true,
        "arrival": "15:35:00",
        "from_station_code": "LTT",
        "name": "Mumbai LTT - Gwalior (Weekly) Special",
        "zone": "CR",
        "chair_car": true,
        "first_class": true,
        "duration_m": 45,
        "sleeper": true,
        "from_station_name": "LOKMANYA TILAK TERM",
        "number": "01101",
        "departure": "15:50:00",
        "return_train": "01102",
        "to_station_code": "GWL",
        "second_ac": true,
        "classes": "",
        "to_station_name": "GWALIOR JN",
        "duration_h": 23,
        "type": "Exp",
        "first_ac": true,
        "distance": 1216
    }
}
```

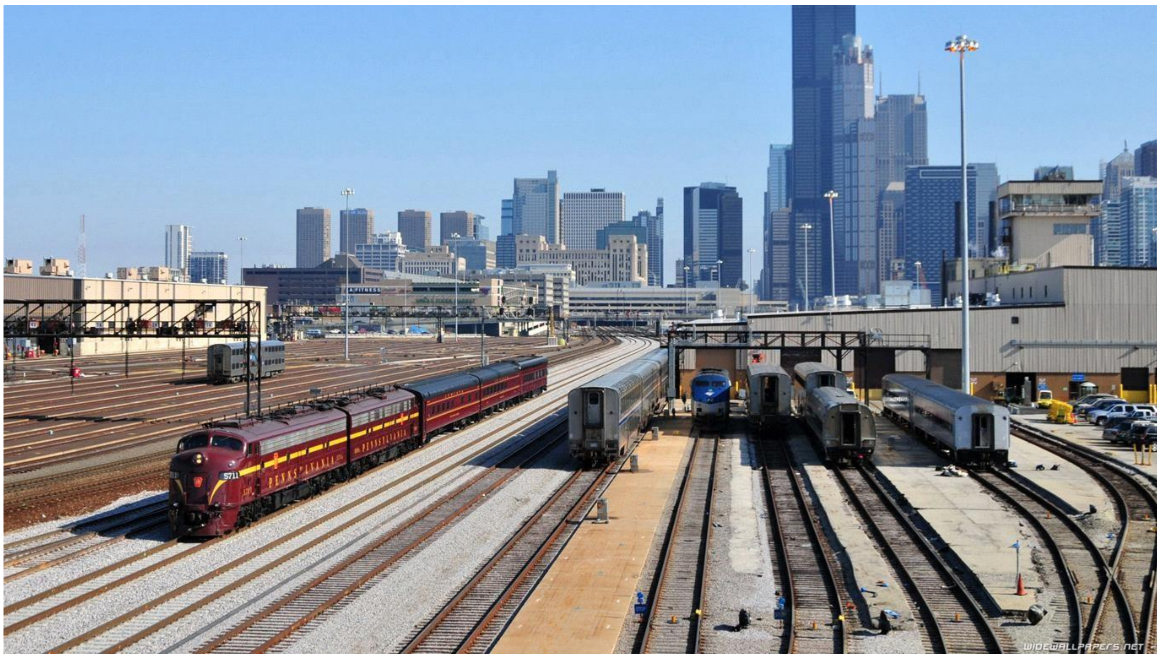
### Schedules
An array of objects. Each object is a schedule which defines a Train stop at a Station.

```json
{
    "arrival": "None",
    "day": 1,
    "train_name": "Falaknuma Lingampalli MMTS",
    "station_name": "KACHEGUDA FALAKNUMA",
    "station_code": "FM",
    "id": 302214,
    "train_number": "47154",
    "departure": "07:55:00"
}
```

#### License

```
# json_data = open('static/railways/stations.json','r')
# stations = json.loads(json_data.read()) # for d in
stations["features"]:
#     data = d["properties"]
#     station = Station()
#     station.state = data["state"]
#     station.code = data["code"]
#     station.name = data["name"]
#     station.zone = data["zone"]
#     station.address = data["address"]
#     station.save()

# json_data = open('static/railways/trains.json', 'r')
# trains = json.loads(json_data.read()) # for d in
trains["features"]:
#     data = d["properties"]
#     train = Train()
#     train.arrival = data["arrival"]
#     train.source = Station.objects.get(pk=data["from_station_code"])
#     train.name = data["name"]
#     train.number = data["number"]
#     train.departure = data["departure"]
#     train.return_train = data["return_train"]
#     train.dest = Station.objects.get(pk=data["to_station_code"])
#     train.duration_m = data["duration_m"]
#     train.duration_h = data["duration_h"]
#     train.type = data["type"]
#     train.distance = data["distance"]
#     print(train)
#     train.save()
#
# json_data = open('static/railways/schedules.json', 'r')
# schedules = json.loads(json_data.read()) # for data
in schedules:
#     schedule = Schedule()
#     schedule.arrival = data["arrival"]
#     schedule.day = data["day"]
#     schedule.station = Station.objects.get(pk=data["station_code"])
#     schedule.train = Train.objects.get(pk=data["train_number"])
#     schedule.departure = data["departure"]
#     schedule.id = data["id"]
#     print(schedule.id)
#     schedule.save()
```



Manage:

```
#!/usr/bin/env python
import os import sys

if __name__ == '__main__':
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'PseudoRCTC.settings')
try:
        from django.core.management import execute_from_command_line
except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)
```

## Team Members

* [pooja Gautam](https://github.com/pooja515), 16IT138

Requirements

Django>=2.1.5 django-authtools==1.6.0 django-braces==1.13.0 django-crispy-forms==1.7.2 pytz==2020.4

Conclusion

Indian Railway Catering and Tourism Corporation (IRCTC) is a subsidiary of the Indian Railways that handles the catering, tourism and online ticketing operations of the Indian railways, with around 5,50,000 to 6,00,000 bookings everyday is the world's second busiest. It's tagline is "Lifeline of the nation". It is known for changing the face of railway ticketing in India. Databases are used to support internal operations of organizations and to underpin online interactions with customers and suppliers. Databases are used to hold administrative information and more specialized data, such as engineering data or economic models. Examples include computerized library systems, flight reservation systems, computerized parts inventory systems, and many content management systems that store websites as collections of webpages in a database. We have tried to implement a part of IRCTC and it has helped us to understand how Database is managed

References

- https://en.wikipedia.org/wiki/Indian_Railway_Catering_and_Tourism_Corporation
- https://www.irctc.co.in/nget/train-search
- https://data.gov.in/keywords/indian-railways
- http://api.erail.in/
- https://railwayapi.com/
- https://indianrailapi.com/

- https://www.programmableweb.com/api/indian-railways

HYPERLINK "https://data.gov.in/resources/indian-railways-time-table-trains-availablereservation-03082015/api" HYPERLINK "https://data.gov.in/resources/indian-railways-timetable-trains-available-reservation-03082015/api"HYPERLINK "https://data.gov.in/resources/indian-railways-time-table-trains-available-reservation03082015/api" HYPERLINK "https://data.gov.in/resources/indian-railways-time-table-trainsavailable-reservation-03082015/api" HYPERLINK "https://data.gov.in/resources/indianrailways-time-table-trains-available-reservation-03082015/api"

APPENDIX Description of tool

Application Tools

- Database: MySQL
- MySQL has proved to be the database for web based applications, because of its performance and scalability, reliability, availability. From the perspective of a database administrator, it's perfectly reliable and easily maintainable.
- Backend Framework : Django
- Django encourages clean, practical way of designing highly customizable applications.
- It is a very reliable, efficient, architecturally sound and secure when building web apps.
- Frontend Tools : HTML, CSS, JavaScript

- HTML stays the markup language for creating web pages and web applications.
- CSS is the stylesheet language for styling the documents.
- JavaScript is the front end scripting language.

Development Tools

- PyCharm Professional 2020.2
- PyCharm provides smart code completion, code inspections, on-the-fly error highlighting and quick-fixes with automated code refactorings and rich navigation capabilities.
- Github
- Helps developers to collaborate over the code easily and for version controlling the source code.