CHAPTER 1

INTRODUCTION

Overview:

"RAILWAY BOOKING SYSTEM" is a simple web application used to book train tickets, it has a simple and extremely user friendly UI, where users can book tickets, view their existing tickets, view all train information, etc. Once a ticket is booked users will also receive an email with their ticket attached to it. Users can also cancel any ticket at any time.

Problem Statement:

The main aim of this web app is to simplify the train booking experience. Without any complicated procedures.

Database Management System:

A database management system (DBMS) is system software for creating and managing databases. The DBMS provides users and programmers with a systematic way to create, retrieve, update and manage data. The DBMS essentially serves as an interface between the database and end users application programs, ensuring that data is consistently organized and remains easily accessible.

The DBMS manages three important things: the data, the database engine that allows data to be accessed, locked and modified, and the database schema, which defines the database's logical structure. These three foundational elements help to provide concurrency, security, data integrity and uniform administration procedures. Typical database administration tasks supported by the DBMS include change management, performance monitoring/tuning and backup and recovery. Many database management systems are also responsible for automated rollbacks, restarts and recovery as well as the logging and auditing of activity.

SQL:

SQL is a standard language for storing, manipulating and retrieving data in databases. Originally based upon relational algebra and tuple relational calculus, SQL consists of a data definition language, data manipulation language, and data control language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control.

SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987. Since then, the standard has been revised to include a larger set of features. Despite the existence of such standards, most SQL code is not completely portable among different database systems without adjustments.

HTML:

HTML is a markup language used for structuring and presenting content on the web and the fifth and current major version of the HTML standard.

HTML5 includes detailed processing models to encourage more interoperable implementations, it extends, improves and rationalizes the markup available for documents, and introduces markup and application programming interfaces (APIs) for complex web applications.

CSS:

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a language such as HTML, CSS is a cornerstone technology of the World Wide Web alongside HTML and JavaScript.

Flask:

To connect the database to the front-end we use Flask, a Python web framework which also acts as back-end in this project.

Flask is a micro web framework written in Python. It is classified as a micro-framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

CHAPTER 2

REQUIREMENTS SPECIFICATIONS

A computerized way of handling information about property and users details is efficient, organized and time saving, compared to a manual way of doing so. This is done through a database driven web application whose requirements are mentioned in this section.

Software Requirements:

- Database Requirements
  - MySQL Server
  - MySQL Shell
  - MySQL Workbench
- HTML and CSS
- Flask
- mysql-connector-python
- Operating System – Windows / Ubuntu
- Python
- Server Deployment – Local Server

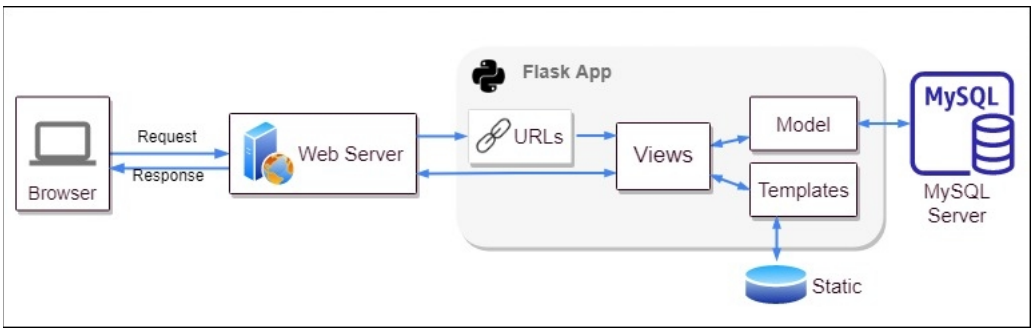
Hardware Requirements:

- RAM – 2GB or above
- Processor – Pentium IV or above
- Hard Disk – 3GB or more

CHAPTER 3

DETAILED DESIGN

System Design:



Three-tier Client / Server database architecture is commonly used architecture for web applications. Intermediate layer called Application server or Web Server stores the web connectivity software and the business logic (constraints) part of application used to access the right amount of data from the database server. This layer acts like medium for sending partially processed data between the database server and the client. Database architecture focuses on the design, development, implementation and maintenance of computer programs that store and organize information for businesses, agencies and institutions.

Entity Relationship Diagram:

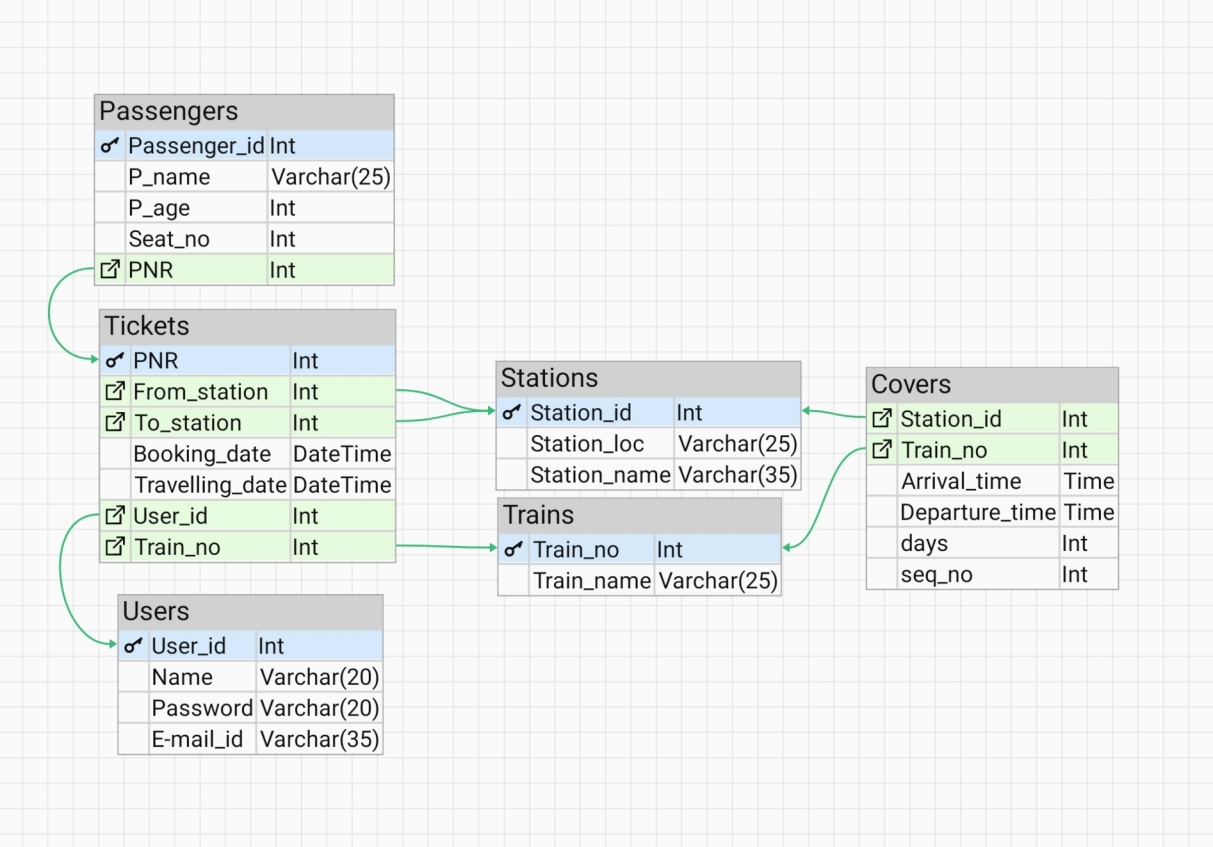
An entity–relationship model is usually the result of systematic analysis to define and describe what is important to processes in an area of a business.

An E-R model does not define the business processes; it only presents a business data schema in graphical form. It is usually drawn in a graphical form as boxes (entities) that are connected by lines (relationships) which express the associations and dependencies between entities.

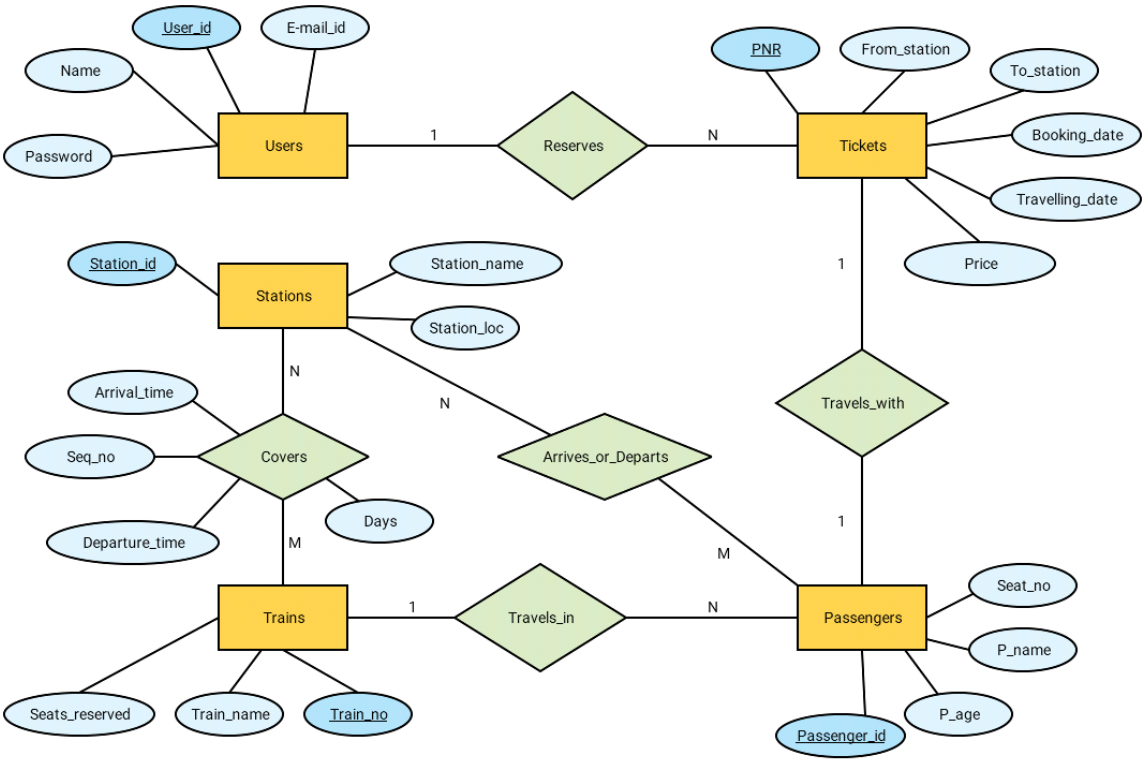
An ER model is typically implemented as a database. In a simple relational database implementation, each row of a table represents one instance of an entity type, and each field in a table represents an attribute type. In a relational database a relationship between entities is implemented by storing the primary key of one entity as a pointer or “foreign key” in the table of another entity.

There is a tradition for ER/data models to be built at two or three levels of abstraction. Note that the conceptual-logical-physical hierarchy below is used in other kinds of specification, and is different from the three schema approach to software engineering.

The below figure is the enhanced ER diagram of railway booking system:



The below figure is the ER diagram of railway booking system:



Relational Schema:

The term "schema" refers to the organization of data as a blueprint of how the database is constructed. The formal definition of a database schema is a set of formulas called integrity constraints imposed on a database. A relational schema shows references among fields in the database. When a primary key is referenced in another table in the database, it is called a foreign key. This is denoted by an arrow with the head pointing at the referenced key attribute. A schema diagram helps organize values in the database. The following diagram shows the schema diagram for the database.

USERS:

user_id	Name	Password	E-mail_id

TRAINS:

Train_no	Train_name

STATIONS:

Station_id	Station_location	Station_name

COVERS:

Station_id	Train_no	Arrival_time	Departure_time	Days	Seq_no

TICKETS:

PNR	From	To	Booking_date	Travel_date	User_id	Train_no

PASSENGERS:

Passenger_id	Passenger_name	Passenger_age	Seat_no	PNR

Description of Tables:

The database consists of 6 tables:

1. USERS: It contains all user's data
   - user_id: Unique ID for all users
   - name: Name of the user (login purpose)
   - password: Password of the user (login purpose)
   - e-mail: Unique e-mail id of user
2. TRAINS: It contains all train's data
   - train_no: Unique ID for all trains
   - train_name: Name of the train
3. STATIONS: It contains all station's data
   - station_id: Unique ID for all stations
   - station_location: Name of the location of station
   - station_name: Name of the station

4. COVERS: It contains data of stations covered by all trains
   - station_id: ID of the station
   - train_no: ID of the train
   - arrival_time: Time at which train arrives to the station
   - departure_time: Time at which train departs from the station
   - days: Number of days taken to reach current station
   - seq_no: Order in which the train covers the station
5. TICKETS: Contains all the data related to a ticket
   - PNR: Unique ID for all tickets
   - from: The origin station
   - to: The destination station
   - booking_date: Date of booking the ticket
   - travel_date: Date of traveling
   - user_id: ID of the user who booked
   - train_no: ID of the train
6. PASSENGERS: Contains all the passenger's details
   - passenger_id: Unique ID for all passengers
   - passenger_name: Name of the passenger
   - passenger_age: Age of the passenger
   - seat_no: Seat number of the passenger
   - PNR: ID of the ticket

CHAPTER 4

IMPLEMENTATION

Module and their Roles:

Login:

The user will be able to login with his username and password to login into his account. Once he's logged in he'll be viewing the home page, which contains links to book ticket, view tickets and view all train information.

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    session["user_id"] = ""
    session["user_name"] = ""
    if request.method == 'POST':
        if db.does_user_exist(request.form):
            session["user_id"] = db.get_user_id_by_user_name(
                request.form.get('username'))
            session["user_name"] = request.form.get('username').title()
            return redirect('/home')
        return render_template("index.html", status=False)
    return render_template("index.html", status=True)
```

Sign-up:

The user can register his/her account by providing an username, password and E-mail ID, once registered he can head back to login page to login into his account.

```
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        response = db.add_user(request.form)
        return redirect('/login', code=302) if response else render_template("signup.html",
status=response)
    return render_template("signup.html", status=True)
```

Home:

The user when login into his/her account, he'll be viewing the home page which contains links to book ticket, view tickets, view all train information and navigation which also contains all these links along with sign-out option.

```
@app.route("/home")
def home():
   return render_template("home.html") if session.get("user_id") else redirect('/login',
code=302)
```

All Trains Info:

Here user will be able to view all train details, like, train number, train name, all the stations it covers, the arrival and departure time at each station.

```
@app.route('/home/getalltrainsinfo')
def get_all_trains_info():
   return render_template("trainsinfo.html", trainsinfo=db.get_train_details()) if
session.get("user_id") else redirect("/login")
```

View Tickets:

When user enters this page, he/she can see all the tickets he/she has booked so far, it contains all the information about the travel. The user can also cancel the ticket if wants to.

```
@app.route("/home/viewtickets")
def viewtickets():
   return render_template("tickets.html", tickets=db.get_tickets(session.get("user_id"))) if
session.get("user_id") else redirect('/login')
```

Book Ticket:

When user enters this page, he/she can book a new ticket, it contains a form, with source, destination and travel date as fields. Then by clicking submit it will take to next step.

```
@app.route("/home/booktickets", methods=["GET", "POST"])
def booktickets():
   if session.get("user_id"):
      if request.method == 'POST':
         db.passengerDetails['source'] = request.form.get('source')
         db.passengerDetails['destination'] = request.form.get(
            'destination')
```

```
            db.passengerDetails['travel_date'] = request.form.get(
                'travel_date')
            return redirect("/home/trains")
        today = date.today()
        min = "%d-%.2d-%.2d" % (today.year, today.month, today.day)
        max = "%d-%.2d-%.2d" % (today.year + 1, today.month, today.day)
        return render_template("booktickets.html", stations=db.get_stations(), min_date=min,
max_date=max)
    return redirect('/login')
```

Available Trains:

When user enters this page, he/she can view all the trains for given source and destination, with it's arrival, departure and reaching time, he/she can select any one train among all the available ones.

```
@app.route("/home/trains")
def trains():
    if session.get("user_id"):
        return render_template("trains.html", trains=db.get_trains(db.passengerDetails))
    return redirect('/login')
```

Add Passengers:

When user is done selecting the train, user can add passengers, with maximum of 6 passengers per ticket and at least 1 passenger. It will be asking passenger's name and age.

```
@app.route('/home/addpassengers', methods=['GET', 'POST'])
def addpassengers():
    if session.get("user_id"):
        if request.method == 'POST':
            name = request.form.get('p_name')
            age = int(request.form.get('p_age'))
            db.passengerDetails['passengers'].append({
                'p_name': name,
                'p_age': age
            })
        return render_template("addpassengers.html", passengers=zip(range(1,
len(db.passengerDetails.get('passengers')) + 1), db.passengerDetails['passengers']),
noOfpassg=len(db.passengerDetails.get('passengers')))
    return redirect("/login")
```

Payment:

After adding the passengers, user can make the payment either in form of Debit/Credit card or UPI. In Debit/Credit card payment, user will be fill prompted to enter card details and in UPI user will be asked to enter VPA (Virtual Payment Address) or user can also scan the QR code to make the payment.

```
@app.route('/home/payment', methods=['GET', 'POST'])
def payment():
    if session.get("user_id"):
        if request.method == 'POST':
            return redirect('/home/reserveticket')
        return render_template("payment.html",
amount=db.get_totalprice(db.passengerDetails))
    return redirect("/login")
```

CHAPTER 5

TESTING

Software Testing:

Testing is the process used to help identify correctness, completeness, security and quality of developed software. This includes executing a program with the intent of finding errors. It is important to distinguish between faults and failures. Software testing can provide objective, independent information about the quality of software and risk of its failure to users or sponsors. It can be conducted as soon as executable software (even if partially complete) exists.

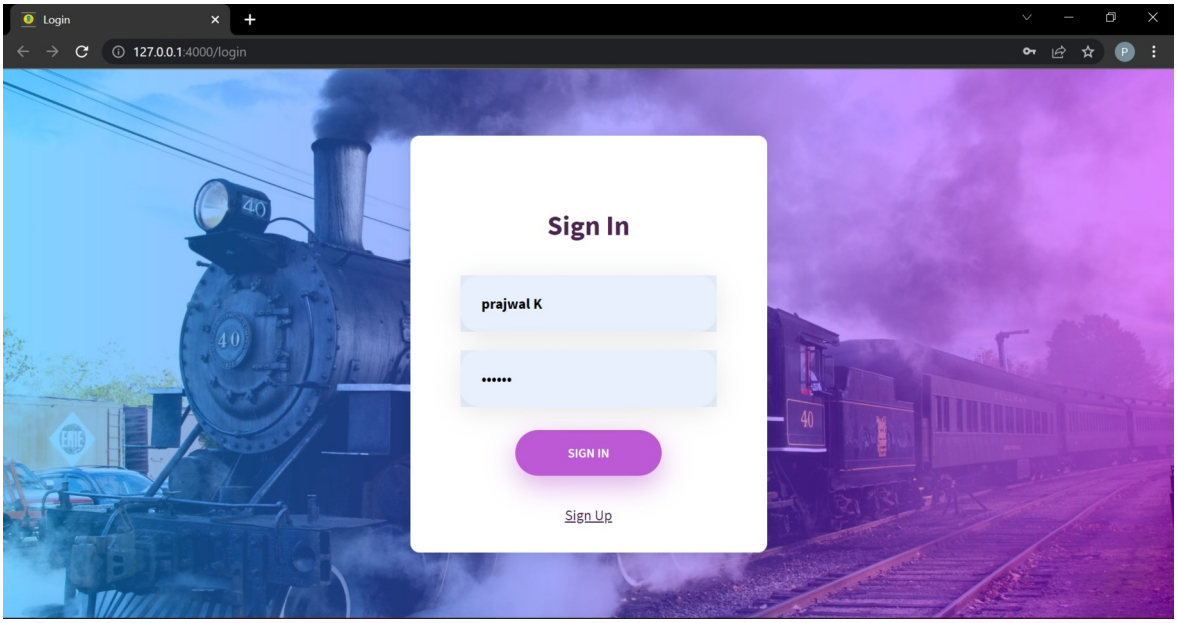
Module Testing and Integration:

Module testing is a process of testing the individual subprograms, subroutines, classes, or procedures in a program. Instead of testing whole software program at once, module testing recommend testing the smaller building blocks of the program. It is largely white box oriented. The objective of doing Module testing is not to demonstrate proper functioning of the module but to demonstrate the presence of an error in the module. Module testing allows implementing of parallelism into the testing process by giving the opportunity to test multiple modules simultaneously. The final integrated system too has been tested for various test cases such as duplicate entries and type mismatch.

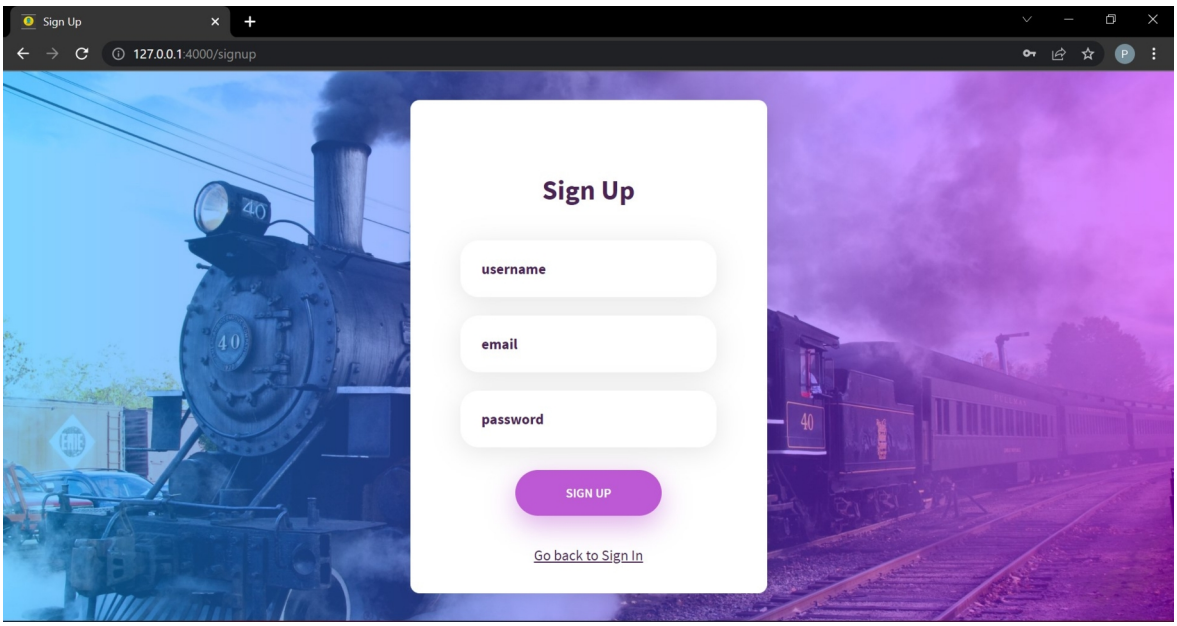
CHAPTER 6

SNAPSHOTS

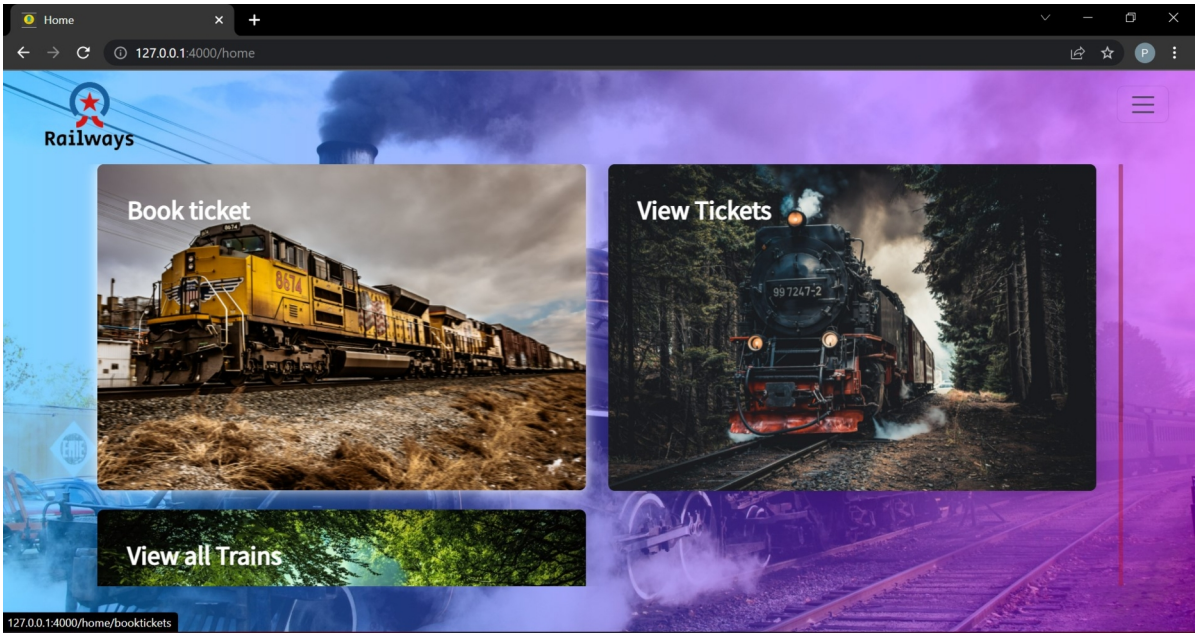
Sign-in:



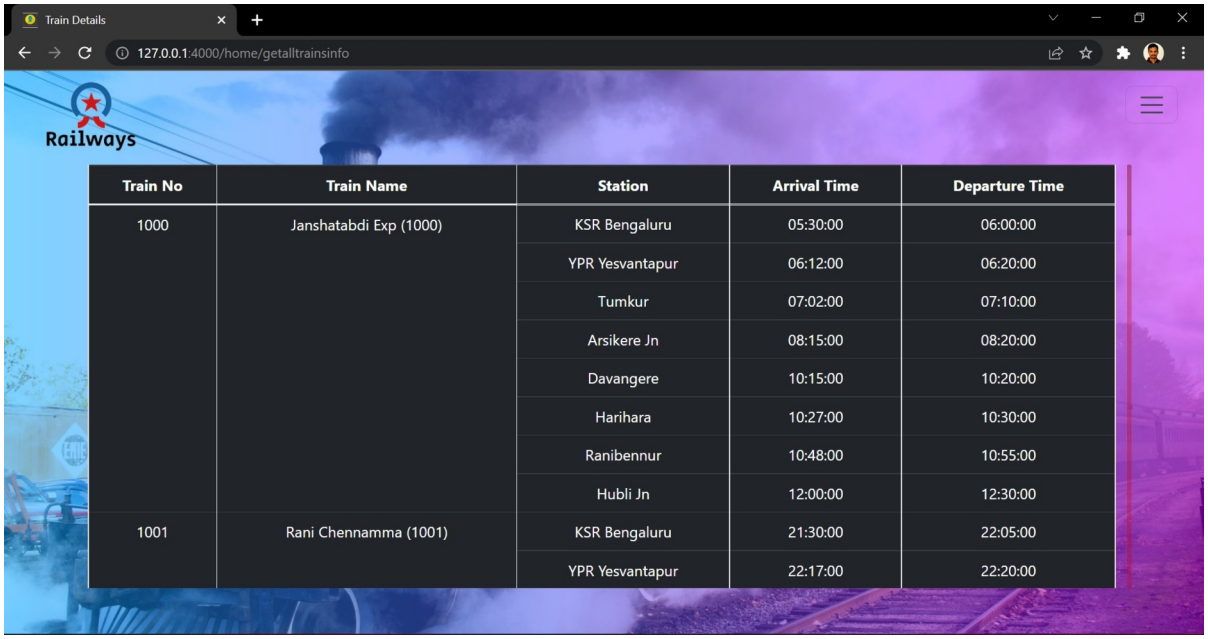
Sign-up:



Home:

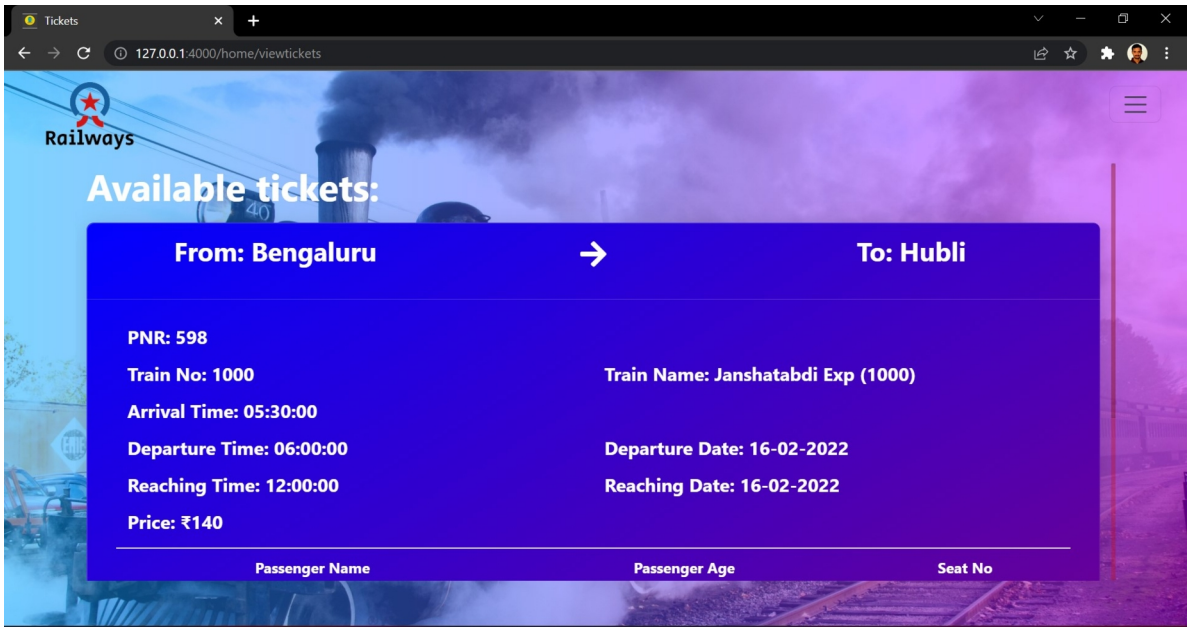


All Train Info:

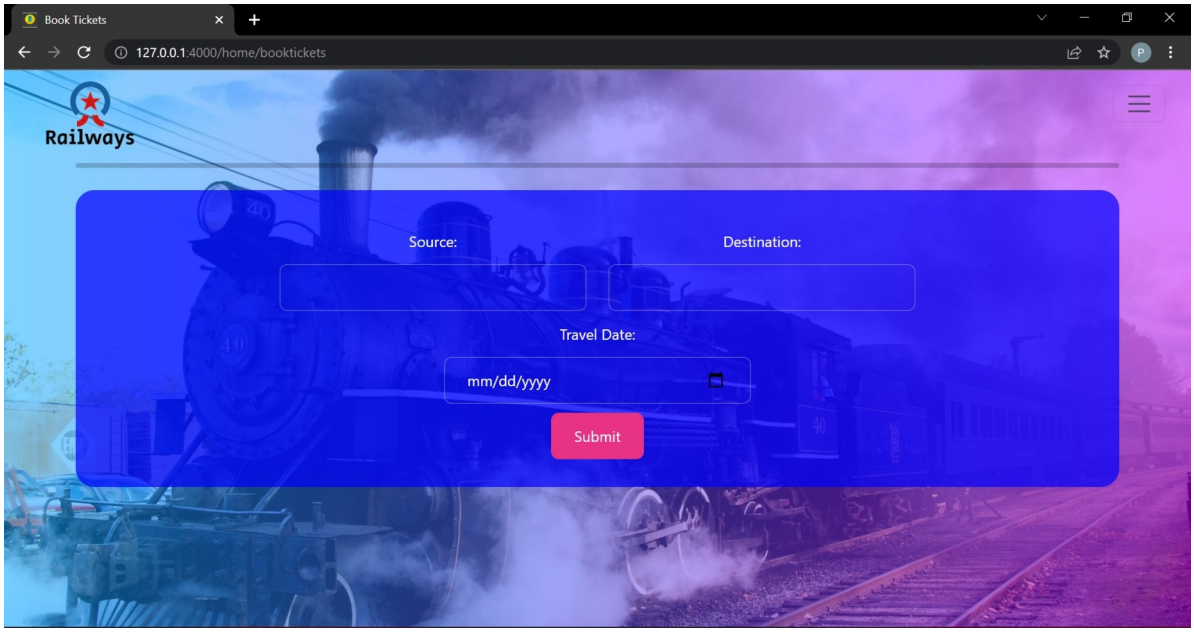


Train No	Train Name	Station	Arrival Time	Departure Time
1000	Janshatabdi Exp (1000)	KSR Bengaluru	05:30:00	06:00:00
		YPR Yesvantapur	06:12:00	06:20:00
		Tumkur	07:02:00	07:10:00
		Arsikere Jn	08:15:00	08:20:00
		Davangere	10:15:00	10:20:00
		Harihara	10:27:00	10:30:00
		Ranibennur	10:48:00	10:55:00
		Hubli Jn	12:00:00	12:30:00
1001	Rani Chennamma (1001)	KSR Bengaluru	21:30:00	22:05:00
		YPR Yesvantapur	22:17:00	22:20:00

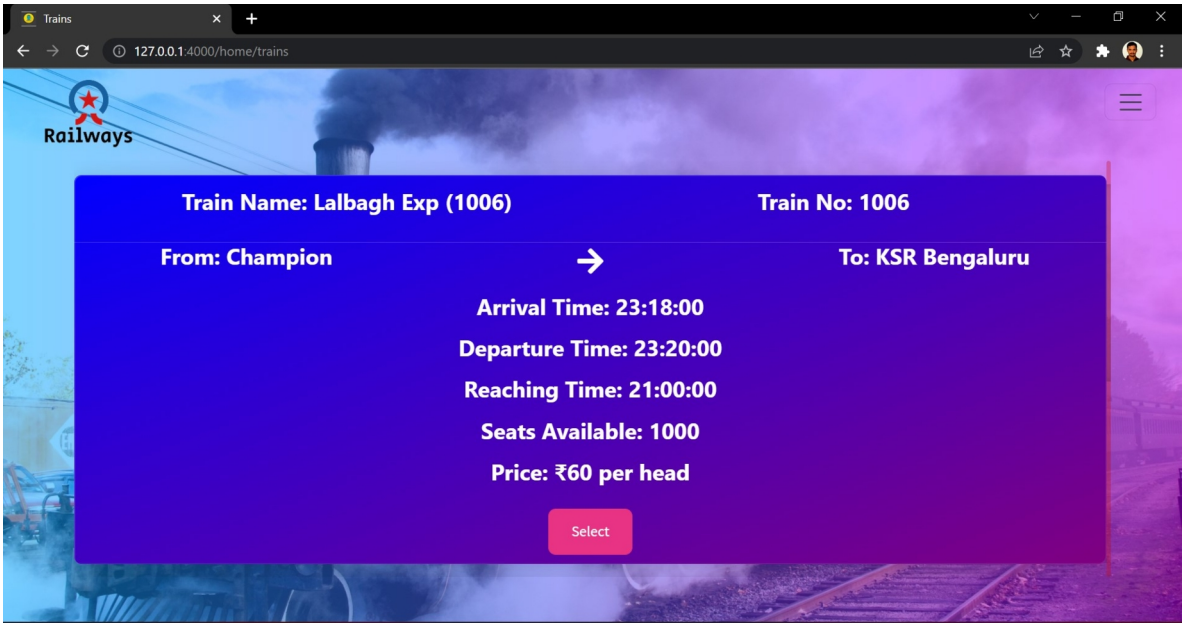
View Tickets:



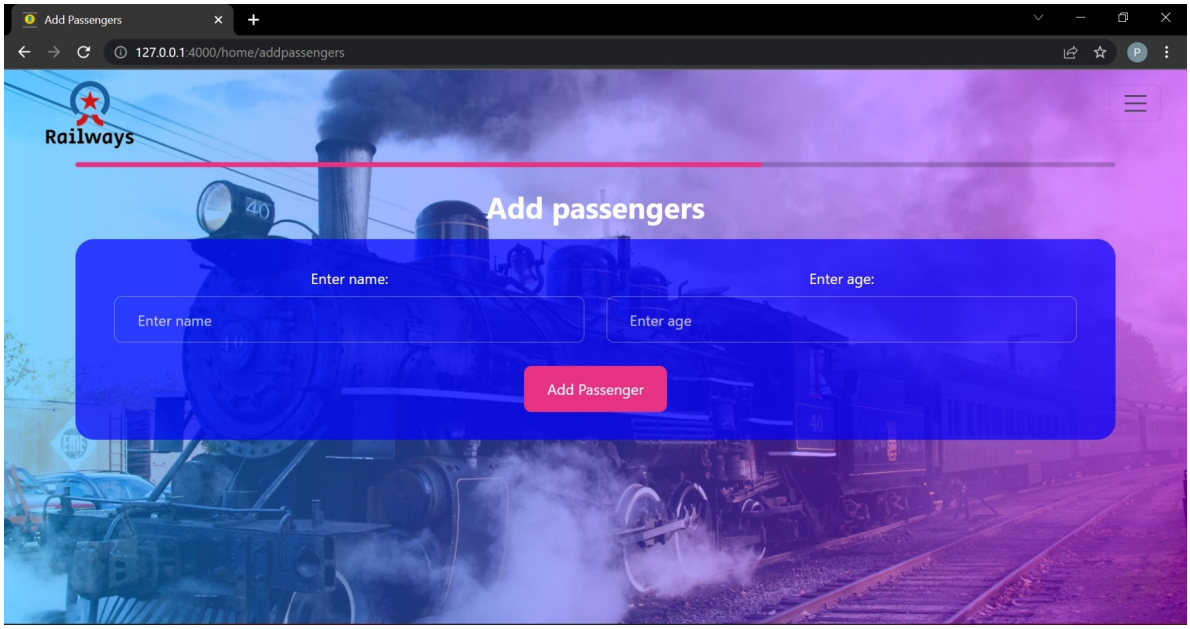
Book Ticket:



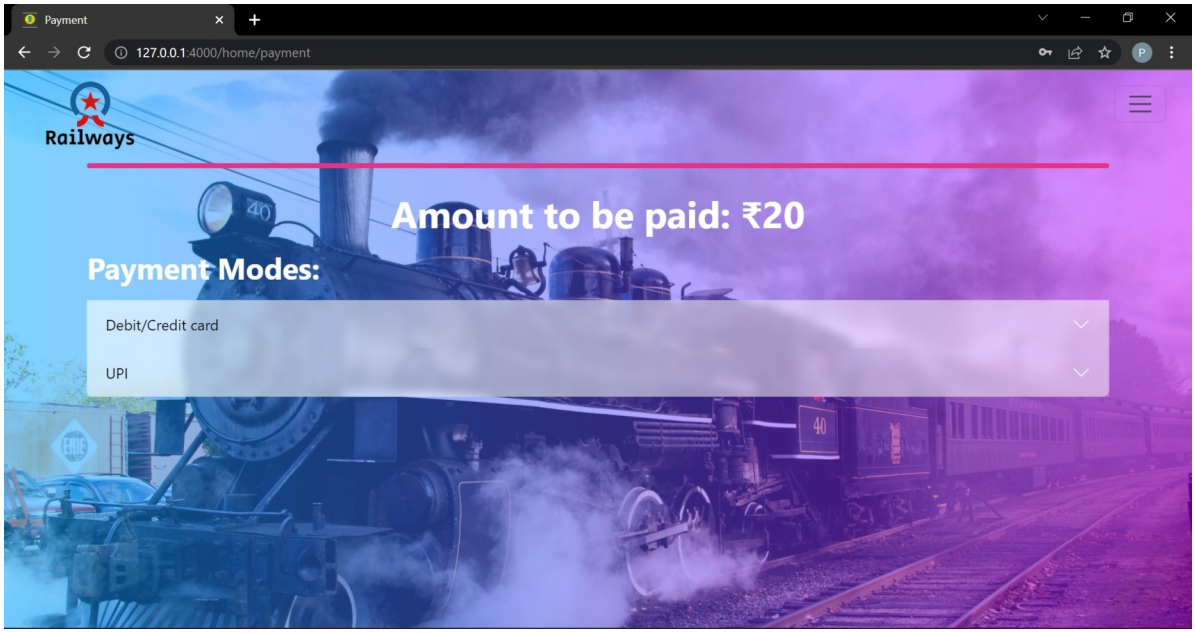
Available Trains:



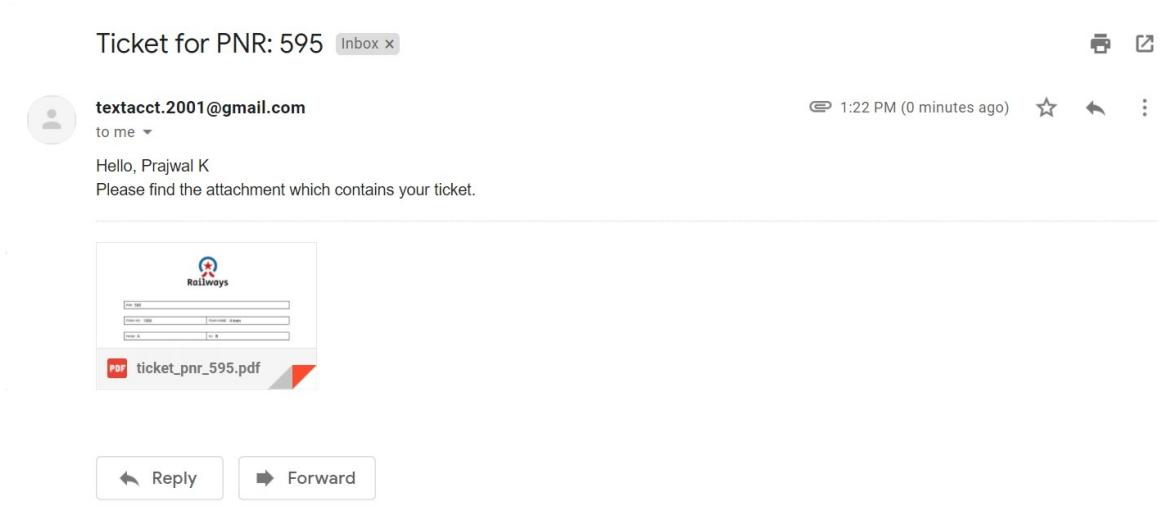
Add Passengers:



Payment:



Mail:



Ticket PDF:

Railways

PNR: 598	
TRAIN NO: 1000	TRAIN NAME: Janshatabdi Exp
FROM: Bengaluru	TO: Hubli
ARRIVAL TIME: 05:30:00	
DEPARTURE TIME: 06:00:00	DEPARTURE DATE: 16-02-2022
REACHING TIME: 12:00:00	REACHING DATE: 16-02-2022
PRICE: ₹140	
BOOKING DATE: 07-02-2022	

PASSENGER DETAILS:

NAME	AGE	SEAT NO
Prajwal Kulkarni	21	1
Prajwal G	20	2

CHAPTER 7

CONCLUSION

The railway booking system provides a easy and clean website to manage booking, viewing and canceling tickets. It allows simplified ways to control database. And also the ability to view the data that is being added and modified. It is also very secure as the password will be encrypted.

This project was developed using HTML5, CSS, Flask and MySQL.

The goals achieved by this project are:

- Centralized database
- Easier adding, modifying and deleting of data.
- Efficient management of tickets.
- Ability to securely manage data in one place.

CHAPTER 8

REFERENCES

1. Ramakrishnan, R., & Gehrke, J. (2011). Database Management systems. Boston: McGraw-Hill.
2. Database systems concepts. Estados Unidos: McGraw-Hill Companies, Inc.
3. Flask Web Development: Developing Web Applications with Python 2nd Edition
4. https://www.w3schools.com
5. https://www.geeksforgeeks.com