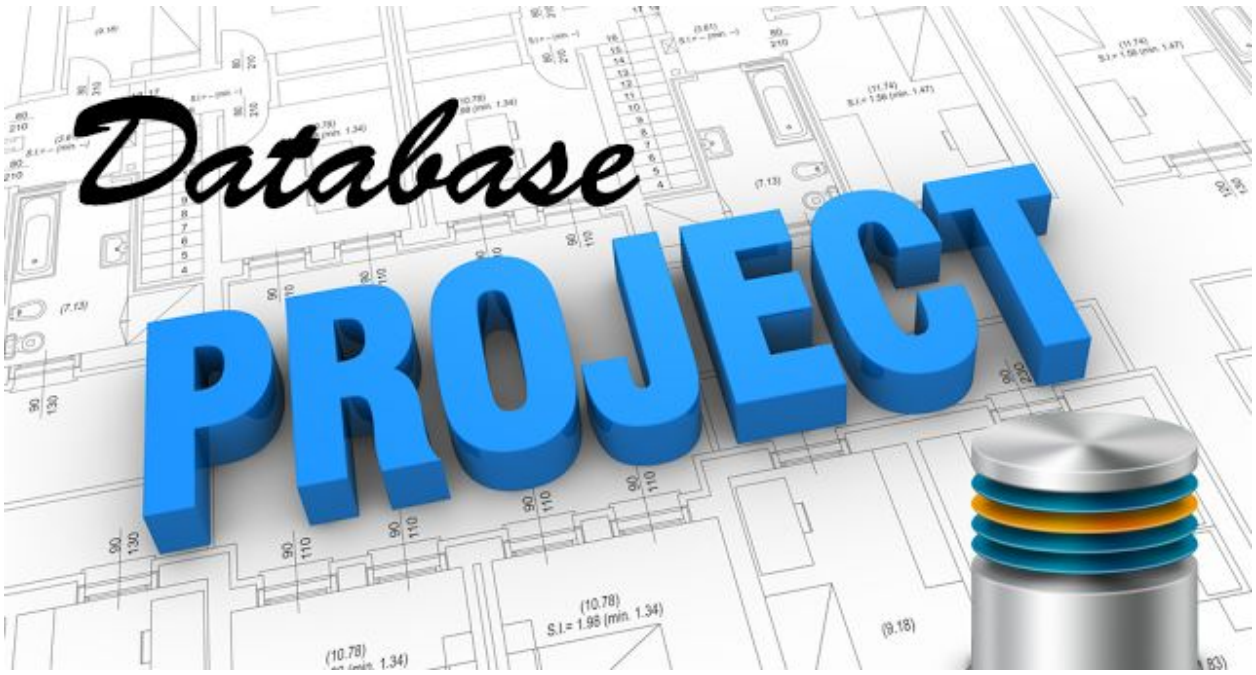


e Rail Seva

Wednesday, 23rd November 2016

Guide : Prof. S Sudarshan

Team Details:

B Avinash -140050029
K Sahil - 140050030
A Goutham - 140050065
B Chaitanya - 140050073

Motivation

One of the main problems faced in real life is the quality of food served on the trains which is much worse than IITB Hostel food . Every time when we go home, we travel for almost 20 to 30 hours in train and suffer a lot for food more than when we are in institute. The people don't have enough options to choose from they have access to a limited menu and food provided in trains is another big problem. This made us come up with a one stop solution to these problems.

Introduction

Through this project work, we aim to create a simplified version of a *Web App* and an *Android App*. These applications will act as a portal which provides it's users with Food ordering service from their seats just using PNR numbers. It also provides Wallet services with order tracking system if possible from very popular restaurants. The aim is to learn how to create an application which uses a database in its backend. We also plan to make the application safe and secure so that it is not susceptible to vulnerabilities and security attacks.

The Web App also provides an interface for vendors to register their shops and menu.

In order to make the application safe and secure for the users, we will try to implement several security measures on the application. For e,g.

- ★ We try to avoid SQL injection attacks by using prepared statements for this purpose

Goals

1. Learn about Django framework and Database Management in the process
2. To create a Web App which uses database in its backend with good user Interface and efficient Database Management System
3. Learn Android Programming

Functionalities

Users (Android App)

- Register, Login
- Place an order at a shop near one of the forthcoming stations
- Track the status of placed Order
- Cancel the Order
- Post a Review
- View all previous orders

ShopKeepers (Web App)

- Register, Login
- Update shop's Menu
- Accept Orders
- Update the status of an accepted Order
- See Reviews Posted by Users

Tables

- **train**
  - train_no varchar(5);
  - train_name varchar(50);
  - from_station_id varchar(6);
  - to_station_id varchar(6);
  - start_time timestamp;
  - end_time timestamp;

  primary_key(train_no);

  foreign _key(from_station_id) references station;

  foreign _key(to_station_id) references station;

- **station**
  - station_id varchar(5);
  - station_name varchar(50);
  - no_of_platforms numeric(2);

  primary_key(station_id);

- **shop**
  - shop_id varchar(7);
  - shop_name varchar(60);
  - station_id varchar(6);

  primary_key (shop_id);

  foreign_key(station_id) references station;

- **food_item**
  - Item_id varchar(4);
  - Item_name varchar(60);
  - shop_id varchar(7);
  - cost_per_plate numeric(3);

  primary_key(item_id);

  foreign_key(shop_id) references shop;

- **pnr**
  - pnr_no varchar(9);
  - coach_no numeric(2);
  - berth_no numeric(2) between 1 and 72;
  - train_start_date timestamp;
  - created_at timestamp;

  primary_key(pnr_no);

- **customer**
  - cust_id varchar(10);
  - cust_name varchar(30);
  - email varchar(50);
  - mobile numeric(10);
  - wallet_amount numeric(4);

  primary_key(cust_id);

- **stops**
  - train_no varchar(5);
  - station_id varchar(6);
  - stop_num varchar(2);
  - arr_time timestamp;
  - dept_time timestamp;
  - day_of_journey numeric(1);

  primary_key(train_no, station_d);

- **review**
  - cust _id varchar(10);
  - shop_id varchar(7);
  - msg varchar(1000);

  foreign_key(cust_id) references customer;

  foreign_key(shop_id) references shop;

- **order**
  - pnr_no varchar(9);
  - cust_id varchar(10);
  - item_id varchar(4);
  - shop_id varchar(7);
  - status numeric(2);

  foreign_key(pnr_no) references pnr;

  foreign_key(cust_id) references customer;

  foreign_key(item_id) references food_item;

  foreign_key(shop_id) references shop;

Setting up and Running

Instructions [For UBUNTU] To Run Django Server, Configure the database and Populate the Database with Sample Data:

(1) The following packages are required for running the server:

python-pip, virtualenv, python-dev, postgresql-server-dev, libpq-dev

Run 'sudo apt-get install python-pip virtualenv python-dev postgresql-server-dev libpq-dev' to install the packages.

(2) Open file 'src/WebApp/db-erailseva/erailseva/settings/localsettings.py':

Set the postgresql database name, host, port, username, password in the DATABASES["default"] dictionary in localsettings.py file

(3) cd into 'src/WebApp/db-erailseva/'

(4) Run 'virtualenv myenv'

(5) Run 'source myenv/bin/activate'

NOTE: Proceed with further instructions only after checking that the current working directory is 'src/WebApp/db-erailseva/' and the virtual environment is activated.

(6) Run 'pip install -r requirements.txt'

NOTE: The above step requires an Internet Connection

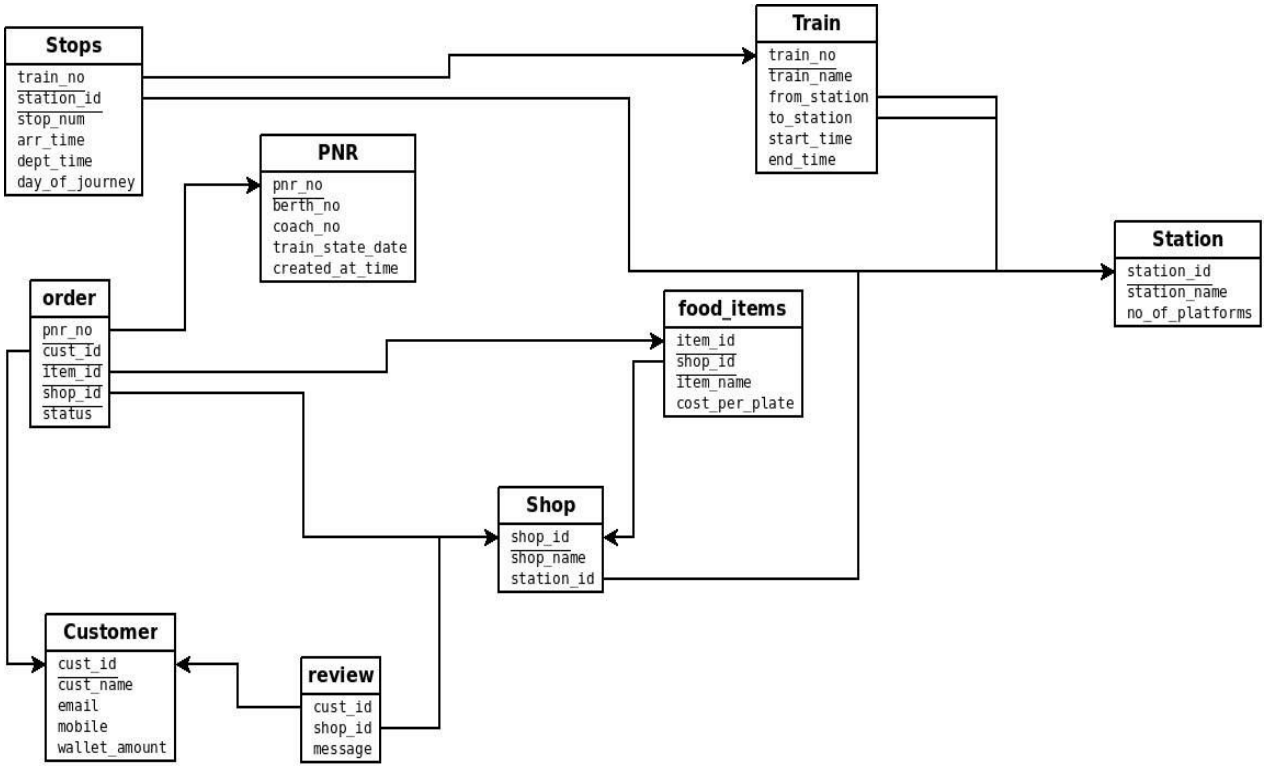
(7) Run 'python manage.py migrate'

(8) Run 'python populateDatabase.py' to populate the database with sample data.

(9) Run ‘python manage.py collectstatic’

(10) Run 'python manage.py runserver' to start the server.

Schema Diagram



APIs (for customer)

^ : webapp only

- customer/register/

  method: POST
  postdata: {“cust_id”: string, “cust_name”: string, “password”: string, “email”: string, “mobile”: string}
  response: {“status”: string, ** “msg”: string}

- customer/login/

  method: POST,
  postdata: {“cust_id”: string, “password”: string}
  response: {“status”: string, ** “msg”: string}

- customer/logout/

  method: GET
  response: {“status”: string, ** “msg”: string}

- customer/getwalletamount/

  method: GET
  response: {“status”: string, ** “wallet_amount”: int, ** “msg”: string}

- customer/setwalletamount/

  method: POST
  postdata: {“amount”: int}
  response: {“status”: string, ** “msg”: string}

- customer/placeorder/

  method: POST
  postdata: {“pnr”: string, “shop_id”: string, “items”: [{“id”: string, “quantity”: int}]}
  response: {“status”: string, “order_ids”: [int,], ** “msg”: string}

- customer/getorderstatus/

  method: POST
  postdata: {“order_ids”: [int,]}
  response: {“status”: string, ** “statlist”: [string,], ** “msg”: string}

- customer/cancelorder/

  method: POST
  postdata: {“order_id”: int}
  response: {“status”: string, ** “msg”: string}

- customer/allorders/

  method: GET
  response:
  {“status”: string,
  “ongoing_orders”: [{“order_id”: int, “shop_id”: string, “shop_name”: string,
  “item_name”: string, “quantity”: int, "station_name": string, “status”: string, “showrb”: bool, “showcb”: bool},],
  “completed_orders”: [{“order_id”: int, “shop_id”: string, “shop_name”: string,
  “item_name”: string, “quantity”: int, "station_name": string, “status”: string},“showrb”: bool, “showcb”: bool],
  ** “msg”: string}

- customer/postreview/

  method: POST
  postdata: {“shop_id”: int, “msg”: string}
  response: {“status”: string, ** “msg”: string}

- train/getstations/

  method: POST
  postdata: {“pnr”: string}
  response:{“status”: string, ** “stations”: [{“id”: string, “name”: string},], ** “msg”: string}

- shop/getshops/

  method: POST
  postdata: {“station_id”: string}
  response:{“status”: string, ** “shops”: [{“id”: string, “name”: string},], ** “msg”: string}

- shop/getitems/

  method: POST
  postdata: {“shop_id”: string}
  response:{“status”: string, ** “items”: [{“id”: string, “name”: string, “cost”: int},], ** “msg”: string}

Following APIs are for WebApp only

- shop/register/

  method: POST
  postdata: {“username”: string, “password”: string, “shop_name”: string, “station_id”: string}
  response:{“status”: string, “msg”: string}

- shop/getallstations/

  method: GET
  response: {“status”: string, “stations”: [{“id”: string, “name”: string},]}

- shop/login/

  method: POST, GET
  response: {“status”: string} for POST; HttpRedirect to /shop/home for GET

- shop/home/

  method: GET
  response: home.html

- shop/fetchorders/

  Response:
  {“status”: string,
  “ongoing_orders”: [{“order_id”: int, “shop_id”: string, “shop_name”: string, “item_name”: string, “quantity”: int, "station_name": string, “status”: string, “showrb”: bool, “showcb”: bool},],
  “completed_orders”: [{“order_id”: int, “shop_id”: string, “shop_name”: string, “item_name”: string, “quantity”: int, "station_name": string, “status”: string},“showrb”: bool, “showcb”: bool],
  ** “msg”: string}

- shop/statuschange/

  method: POST
  postdata: {“order_id”: string, “status”: string}
  response: {“status”: string, “statuschanged”: string}

- shop/reviews/

  method: GET
  response: review.html

- shop/logout/

  method: GET
  response: HttpRedirect to /shop/login

- shop/getmenu/

  method: GET
  response: menu.html

- shop/fetchmenu/
  method: GET, POST
  postdata: {“updates”: [{“id”: string, “cost”: int, “deleted”: bool}]}
  response: {“status”: string} for POST, {“status”: string, “rlist”: [{“item_id”: string, “item_name”: string, “cost_per_plate”: int}]}

- shop/add_menu

  method: POST
  postdata: {“item_name”: string, “cost”: int}
  response: {“status”: string}

Future Work

1. Would implement the webapp of the customer completely i.e integrate it with backend api's using ajax.
2. Integrate both our app's with minimal changes with Live IRCTC api so that we can track the live status of the train and also get all the pnr, station details from it.
3. Use some payment gateway to add money to wallet and also make UI more user friendly with complete form validations

Future Scope

We feel that if the project is implemented completely with the points included in the future work , the app would be a very good product to use in real life. This makes Food just at a one click away when you travel in a train. This is a very good marketing strategy and can raise it to a startup

Of course, there should be some "Terms and Conditions" that all the shops using the app should follow to indeed fulfill all the accepted orders. All the shops which are in the near range of railway stations could use this app and also for passengers travelling in trains the app would be very handy.