Databases Final Project Document
2020-12-15

Zhenming Wang, Ren Sheng

How to run?

Set the directory at flightBooking where you have a flightBooking folder and runserver.py(outside one), and use the following command to run:

```
python runserver.py
```

1. A list of files
    a. __init__.py: contains the functions for establishing the flask server and the connection between the server and the database.
    b. Controller/controller.py: contains all the flask routes for our functions for the program.
    c. Model/model.py: contains a partly established ORM model using sqlalchemy. We only build a User class for flask_login to do login check, so no other classes are established.
    d. Service: contains customerService.py, agentservice.py, staffService.py. all the three files are expected to store mysql-concerned functions, so that we can save space for controller.py as well as making it better organized. However, as you can see some of the mysql-concerned functions are still inside controller.py. This is because we do not have time to put all of them in order.
    e. Static: contains all static files we write for the program, css, javascript, svg, img, etc.
    f. Templates: contains all html files we write for the program.
    g. Sql/create_table.sql: create tables and insert data for our database. Noting that we HAVE changed some structures of the original database, so you have to load this sql file!!!
    h. Sql/functions_procedures.sql: create some stored procedures and functions for our database. This is used for our mysql calls inside python files, so you have to load this sql file, too!!!
2. Division of Labor
   Zhenming Wang: Frontend (html, css, js), Database Design.
   Ren Sheng: Backend (python Flask), Database Design.
3. Use Case Summary:
    a. Part of our main sql queries are coded inside stored procedures and functions, then called by function inside service.py and referenced from controller.py. Due to time constraints, part of our sql queries are coded straightly inside controller.py.

b. We change the structure of the database to enable multiple tickets in one purchase.
c. We have new tables airline_sequence_num and purchase_sequence_num to automatically assign id to airplane, flight and purchase.
d. Click the logo Eflight to go back to the home page!!!

4. Use Case Introduction:

Use Cases for all

a. Register
Related routes: /eFlight/doRegister
Related stored procedures: create_customer, create_staff, create_agent
Description: insert the new values into customer/staff/agent table and also user table (for login use)
b. Login
Related routes: /eFlight/doLogin
Related stored procedures: login_check
Description: check if the email/username pairs with password, if so, then use flask_login to login the user
c. View Public Info
Related routes: /eFlight/search
Related stored procedures: search_by_city, search_by_num
Description: search the flights (but cannot buy) with city, airports or flight number; note that we use ‘like’ clause to make search smoother, so that you can search for a Shanghai-to-NY flight by a rough searching like “lalaShanghaiiii’ and ‘new york city’ while the city is actually ‘Shanghai’ and ‘New York’.
d. Logout
Relate routes: /eFlight/logout
Description: use flask_login to log out. Jump back to the home page.

Use Cases for customer:

a. View my flights
Related routes: /eFlight/viewMyFlights
Related stored procedures: customer_view_flights
Description: You can see this by clicking “my record”. It searches the purchase (natural join ticket, flight) with the email of the customer, filtered with different conditions. Note that for status, since we have a code for each status, you need to type in the integer code!!! 1-upcoming, 2-departured, 3-arrived, 4-delayed, 5-cancelled.

b. Purchase Tickets
   Related routes: /eFlight/purchaseTicket
   Related stored procedures: insert_ticket
   Description: insert ticket and purchase. Here we have ticket_id as the primary key for both of the tables, but in purchase different tickets bought at one time are assigned an identical purchase_id.
c. Search for flights
   Related routes: /eFlight/purchaseSearch
   Related procedures: search_by_city
   Description: search by departure city, arrival city and date. Note that three inputs should all be given to continue here, as only here you have a chance to buy tickets. Users can type in airports or cities in departure and arrival.
d. Track my spending
   Related routes: /eFlight/trackDateSpending, /eFlight/trackMonthSpending
   Relate procedures: get_date_spending, get_month_spending
   Description: you can see this by clicking “my record” and then “my spending”. Here default display is not done, so you may type in the date range to see the spending. They work!

Use Cases for booking agent:

a. View my flights
   Related routes: /eFlight/viewMyFlights
   Related stored procedures: customer_view_flights
   Description: It shares the same route with customers, but not the same procedure. You can see this by clicking “my record”. It searches the purchase (natural join ticket, flight) with the booking agent id, filtered with different conditions.
b. Purchase tickets
   Related routes: /eFlight/purchaseTicket
   Related stored procedures: insert_ticket
   Description: It shares the same route with customers. Insert ticket and purchase, with booking_agent_id as the user’s own id.
c. Search for flights
   Related routes: /eFlight/purchaseSearch
   Related procedures: search_by_city
   Description: search by departure city, arrival city and date. Note that three inputs should all be given to continue here, as only here you have a chance to buy tickets. Users can type in airports or cities in departure and arrival. It shares the same route and procedure with customers.

d. View my commission
   Related routes: /eFlight/viewCommission
   Related procedures: the viewCommission function inside agentService.py. Just raw sql sentences in python. No stored procedure.
   Description: It returns the past 30 days' commission by default. You can also set a date range.
e. View top customers
   Related routes: /eFlight/viewTopCustomer
   Related procedures: get_top_ticket_customer, get_top_commission_customer
   Description: Here it returns two bar charts as required.

Use Cases for airline staff:

Click the logo Eflight to go back to the home page!!!

a. View my flights
   Related routes: /eFlight/staffViewFlights, /eFlight/getInfo
   Related procedures: view_my_flight function inside staffService.py. Just raw sql sentences in python; also procedure get_passengers_by_flight.
   Description: By default it gives flights in the past 30 days. As long as one condition is not empty, it will follow the condition. You can see an info button for every flight, which contains the name of all passengers.
b. Create new flights
   Related routes: /eFlight/addNewFlight
   Related procedures: create_flight
   Description: you can see 'create' columns, and then choose 'flight'. Here date and time are chosen, but for airports you need to be sure what you type is a valid airport code, otherwise it will get stuck.
c. Change status of flights
   Related routes: /eFlight/changeStatus
   Related procedures: raw sql sentences inside the route. No service function or stored procedure.
   Description: For every flight you can see an update button, by clicking which you can change the status of the flight.
d. Add airplane in the system
   Related routes: /eFlight/addNewAirplane
   Related procedure: create_airplane
   Description: you can see 'create' columns, and then choose 'airplane'. Note that airplane_id is automatically assigned, so you only need to give seats.
e. Add airport in the system

Related routes: /eFlight/addNewAirport
Related procedure: create_airport
Description: you can see ‘create’ columns, and then choose ‘airport’. Then give airport code and city.

f. View booking agents
Related routes: /eFlight/viewBookingAgent
Related procedure: get_top_commission_agent, get_top_ticket_agent
Description: you can see ‘view’ columns, and then choose ‘view booking agent’. Then it gives three top 5 booking agents as required.

g. View frequent customers
Related routes: /eFlight/viewFrequentCustomer
Related procedure: most_frequent_customer
Description: you can see ‘view’ columns, and then choose ‘view customer’. Then it gives the most frequent customer with the amount of tickets he bought.

h. View reports
Related routes: /eFlight/viewMonthReport, /eFlight/viewDateReport
Related procedure: get_month_report, get_date_report
Description: you can see ‘view’ columns, and then choose ‘view report’. Here default display is not done, so you may type in the date range to see the spending. They work!

i. Revenue comparison
Related routes: /eFlight/compareRevenue
Related procedure: revenue
Description: you can see ‘view’ columns, and then choose ‘revenue comparison’. Here you can see two brilliant pie charts done by Zhenming!

j. View top destination
Related routes: /eFlight/viewTopDestinations
Related procedure:get_top_destinations
Description: you can see ‘view’ columns, and then choose ‘view top destinations’. It will give you three top cities, and the times they are visited. We have tried two airports in the same cities, and it works well with this condition.

5. Summary:
We did not do well in time management, so we did not optimize all the functions. If we have more time, I think we will definitely make the code and the interface better organized. Thank you.