# 

**Programming for Information Systems Assignment**

**Easy Travels Web Portal**

**Module Title: Programming for Information Systems (B9IS123)**

**Module Leader: Paul Laird**

**Group Members: Manik Mahashabde (10518579)**

**Saurabh Devade (10531140)**

# TABLE OF CONTENTS

[TABLE OF CONTENTS 2](#_Toc27512266)

[1. Introduction 3](#_Toc27512268)

[2.Technologies used in the information system 3](#_Toc27512269)

[3. Requirements of the information system 3](#_Toc27512271)

[4. User of the information system 4](#_Toc27512272)

[4.1 Employees 4](#_Toc27512273)

[4.2 Customers 4](#_Toc27512273)

[5. Data requirements of the information system 4](#_Toc27512274)

6. Implementation of the information system

[6.1 Customer Controller 5](#_Toc27512273)

[6.2 Enquiry Controller 6](#_Toc27512273)

[6.3 Employee Controller 7](#_Toc27512273)

[6.4 Package Controller 9](#_Toc27512273)

[7. Validation of the information system 9](#_Toc27512289)

[10. BIBLIOGRAPHY 31](#_Toc27512291)

[12. APPENDIX B 34](#_Toc27512294)

**1: Introduction:**

We have designed and implemented a travelling management system for a travel company named as “Easy Travels Web Portal”. This portal provide a variety of services related to a travel package booking for our customers. Also, the services of handling the enquiries given by the customers. We have employees in the organization with each employee having a unique role such as sales, operation, admin, operation head, HR. Each employee is having a unique role and the same role can have multiple employees. An Employee with a role sales can create and handle enquiries for the customers, an employee with role operation can create packages. Once a package is created by an employee with a role operation it is approved by operation head employee before releasing.

Customer can visit our website and ask us for registering themselves in the portal. Once registration is done then a customer can login into the portal and then select from the variety of packages from the book and book it. Customer can also contact us in case of any issue. All of the information of the employees and customers are stored in the database. Hence all the CURD ( create, update, retrieve, delete) are done on it.

This System is designed for a Tour packages management company

**2: Technologies used in the Information System**

For the creation of the information system we have used below technologies:

(Copperwaite and Leifer; Dwyer; Lutz; “Welcome to Flask — Flask Documentation (1.1.X)”)

|  |  |
| --- | --- |
| **Backend** | Python Flask framework. |
| **Database** | MySQL |
| **Frontend Client** | Postman |
| **Design architecture** | Model-View-Controller |
| **Tools used** | Pycharm, Visual Studio Code, MySQL Workbench, Navicat |
| **Versioning-Tool** | GitHub |

**3: Requirements of the Information System**

**Functional Requirements:**

3.1: Employees and customers should be able to login to the system, and their session should be maintained.

3.7: Employee with role admin should be able to create other employees and assign roles to them.

3.3: There should be provision for employees to change password with the OTP and otp should be sent to the registered email id.

3.2: Employees with role operation should be able to create holiday/tour packages.

3.3: Employees should be able to update package details

3.4: There should be functionality to search through itinerary of the packages.

3.5: There should be functionality for searching through all the customers.

3.6: There should be functionality for searching through Employee.

3.5: Employee with role sales should be able to create and handle enquires asked by the customers.

3.6 Employee with role sales should be able to register a new customer.

3.8: On Deletion of Customer entry, all the enquiries associated with that customer will be deleted.

3.9: Customer should be able to book their desired packages.

3.10 There should be a functionality to get all customers, all enquires raised, all packages (for showing on website )created.

3.11 While retrieving, all the entries should be sorted by the latest creation.

**Non- Functional Requirements:**

3.10: Creating a frontend for the information systems so that it is easily accessible.

3.11: Using an appropriate data structure for storing and retrieving data from the database in the data access object (DAO) class.

3.12: Using appropriate SQL functions to fetch records from the database based upon API.

**4: Users of the Information systems**

Following are the two major users of the information systems:

**4.1 Employees**

Employees act as an actor or user of the easy travels web portal information systems by managing the web portal. They are the one responsible for handling requests which comes from the clients to the server. Request such as enquiry creation, customer creation, customer deletion, package creation etc. are handled by the employees.

**4.2 Customers**

Customers also act as a user of the system by interacting with the employees. They send requests to the server by acting as a client. Request such as enquiry creation, new package creation, getting the booking details etc are raised by the customers. These requests are handled by the employees at the server end.

**5: Data requirements of the Information systems**

For data collection of the information system, we have created a database in MySQL. There are 8 tables in the database such as customer, employee, package, employee\_role, employee\_role\_mapping, booking, package\_itenary and enquiry. Each table is having a unique primary key for easy data processing and manipulation. Each table is also having one or more foreign key of another table to retrieve data and cascade manipulation of data. We have added the database and all the tables in the GitHub at below URL. Kindly refer to it for more details related to the data requirement.

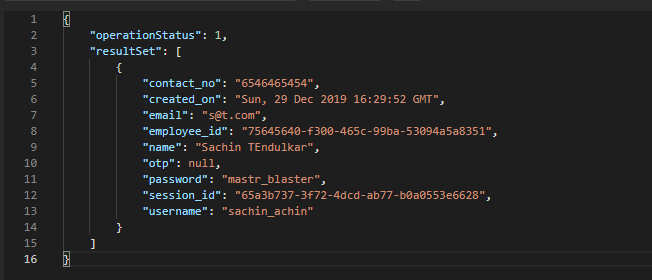
<https://github.com/mahas500/Python_CA_Easy_Travels>

**6: Response Object & Validations**

For Uniformality we want to send the same object as a response in every API

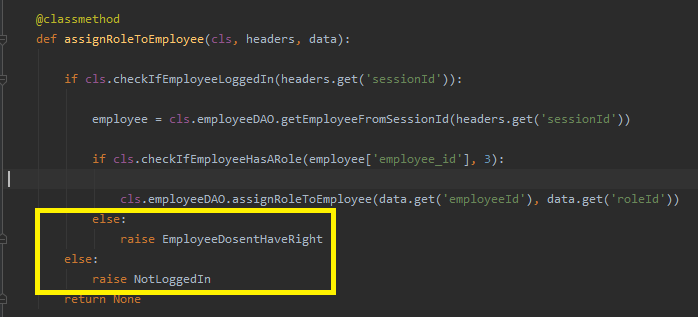
wsResponse = {"resultSet": None, "operationStatus": None}

this **wsResponse** contains **resultSet** which can be any retrieved or updated data. It can be null also. And **operationStatus** is the status code returned with that API. If the API executes successfully it will return 1 else it will return the status code based on the current failure (status codes are discussed more in next section).



**7: Validation of the Information systems**

Validations are one of the most important factors in any Information System. For this system also we have implemented some validations. For implementing those we throw exceptions at the service level and then we catch those exceptions in controller level and assign them the status codes shown below. Further in front-end we can show specific error messages based on those errors.





SUCCESSFUL = 1  
SOMETHING\_WENT\_WRONG = -1  
WRONG\_CREDENTIALS = -2  
EMPLOYEE\_DOSENT\_HAS\_RIGHT = -3  
EMPLOYEE\_NOT\_LOGGED\_IN = -4  
PACKAGE\_DOES\_NOT\_EXIST = -5  
CUSTOMER\_DOES\_NOT\_EXIST = -6  
EMAIL\_SENDING\_FAILED = -7  
EMPLOYEE\_WITH\_EMAIL\_NOT\_EXIST = -8  
CUSTOMER\_NOT\_LOGGED\_IN = -9

**6: Implementation of the Information systems**

Implementation is done by using python flask framework. Reason for using flask is that, as the aim is to develop a backend system of a web application **flask** suits the requirement as it is a lightweight framework. It provides ease with creating restful API web service. It also has good inbuilt libraries for database connectivity and frontend. Hence dynamic API’s are created in a short period time using flask.

Below are the details of the design architecture used:

**(MVC) Model View Controller**: It is a design pattern which gave structure to the information system. In MVC, the controller maintains the communication between a model and a view. It makes sure that a model and a view doesn’t have direct communication with each other due to security reasons. (Mufid et al.; Selfa et al.)

In our application, we have 4 controllers namely:

* CustomerController
* EmployeeController
* EnquiryController
* PackageController

RestAPI calls are written in each controller class. Then there are models which are implemented in the service classes. Service class contains the logic of API and CURD(create, update, retrieve, delete) are performed in DAO classes. Hence, frontend client we have used is postman for sending GET and POST HTTP requests to the endpoints present in the controller classes.

The Flow of information for a **GET** request is:

PostMan(request) 🡪 Controller class 🡪 Service class 🡪 DAO class 🡪 Operation on the database.

The Flow of information for a **POST** request is:

PostMan 🡪 Controller class 🡪 Service class 🡪 DAO class 🡪 Operation on the database(response fetched) 🡪 DAO class 🡪 Service class 🡪 Controller class 🡪 PostMan( received response)

Below is the implementation for major entities:

**1) Customer Controller**: There are multiple APIs endpoint in this class

1. createCustomer

Endpoint : **/createCustomer**

Create a customer is created by an employee with role ID 4. The function in the customer controller class calls the function in the customer service class. Customer service class contains the logic of customer creation which then calls the function in DAO class which consist of insertion of a customer record in the customer table.

1. Get all customers

Endpoint : **/getAllCustomers**

It consists of CURD operation READ/RETRIEVE to retrieve all customers records from the customer table and sort them based on the latest creation.

1. Customer login

Endpoint : **/customer login**

It consists of passing a username and the password in the POST request and logging in the portal. Session ID is generated once the customer is logged in. For generating session we have used UUID version 4 to generate unique session token for employees and customers

1. Customer delete

Endpoint : **/deleteCustomer**

It consists of a CURD operation DELETE to remove customer record by the database. It is performed by an employee with role ID 4. It is a cascade delete operation i.e once customer record is deleted from the customer table using primary key customer ID then records in all the tables which contain customer ID as a foreign key are deleted automatically.

1. Endpoint : **/searchCustomer**

As mentioned in the business requirement employees should be able to search customers to perform further operations on them. For that, we pass a search query which can be name, email, username etc. We added MySql full-text binary search index in table so that it will retrieve all the matching records which contain similar pattern like provided search query.

**2)Enquiry Controller**: There are 2 API in this class.

1. Create an Enquiry.

Endpoint : **/createEnquiry**

It consists of an enquiry creation in the enquiry controller class. An enquiry is created by an employee with role ID 4. So, authentication is done for it. After this enquiry creation record is inserted into the enquiry table.

1. Get all enquiry

Endpoint : **/getAllEnquiries**

It consists of a function which retrieves all the enquiry records from the enquiry table sort them based on the latest creation and returns it to the user.

3)**Employee Controller**: There are 7 API in this class.

1. Create an Employee

Endpoint : **/createEmployee**

It consists of a function to create a new employee. This operation is only performed by an employee with role ID 3(admin). After authentication of the admin employee, a new employee is created by inserting a record in the employee table. Employee ID is generated randomly using python UUID library.

1. Employee login

Endpoint : **/employeeLogin**

It is similar to the customer login. Employee sends a username and password in the POST request. After authentication of the credentials from the employee table record is returned. Session ID is created automatically in this function using python UUID and it is updated in the employee table.

1. Get all employees

Endpoint : **/getAllEmployee**

In this, all the employees are returned by sorting them in latest creation order.

1. Assign role to employee

Endpoint : **/assignRoleToEmployee**

This operation is performed by the admin employee. In this function different role is assigned to each employee after authentication and after making sure employee doesn’t contain an existing role.

1. Search from employees

Endpoint : **/** **search employee**

Admin can search from available employees passing search query from the request body.

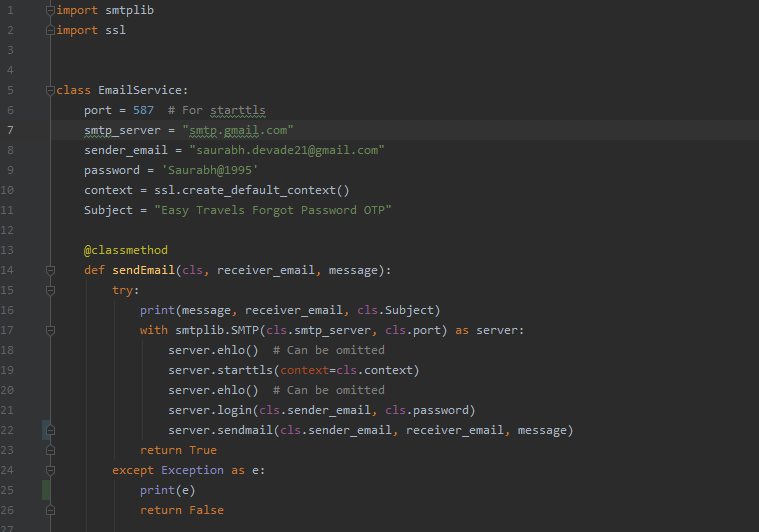
1. Search from employees

Endpoint : **/** **forgotPassword, /chnagePassword**

If an employee forgot the password then there is functionality of changing a password. Once he clicks on forgot password then system will ask him for a email id. If email id exist in the system then OTP will be sent on that email and will be saved in the database as well. Once the employee receives that email he will go to change password and will enter a new password and OTP. If this provide OTP matches with the OTP in the database the password will be changed.

We have used SMTP provided by smtplib

(“20.12. Smtplib — SMTP Protocol Client — Python 2.7.17 Documentation”)



**Validations / Conditions checked–**

1. Employee Should exists with that email

5)**Package Controller** – Consist of 6 API

1) Create a package

Endpoint : **/createPackage**

To enter the details of the new package this function is implemented. While calling this API we have to pass session\_id and role\_id of the employee as a header along with package data as a request body. Before package creating employee login status and if he is authorized to perform this operation based on his role is checked, if all the condition satisfies then the new package is created.

**Validations / Conditions checked–**

1. Employee should be logged in

2. Employee should have a valid role

2) Get All Packages

Endpoint : **/getAllPackage**

To retrieve all the packages entry this function is implemented . While retrieving, the latest created packages are retrieved first i.e sorting is done based on created\_on date,

3) **Create Package itinerary**

Endpoint : **/createIternaryForPackage**

Every package will have a separate day-wise itinerary. For this sperate table is created, in that every row contains details of one day mapped against the package id. For this data is to be passed as an Array of JSON and in the service, layer data is being inserted day-wise using for loop

**Validations / Conditions checked–**

1. Employee should be logged in

2. Employee should have a valid role

3. Package should exist

4) Getting a package and itinerary details

Endpoint : **/** **getPackageWithIternaryDetailsFromPackageId**

In this API package, details with all of its itinerary details are returned.

5) Searching Through a package

Endpoint : **/** **searchPackage**

If one has to find for specific package there should be functionality implemented for this. For this, we are passing a search query and using that search query and MySql full-text search we are returning all matched records against that search query

1. **Booking a package**

Endpoint : **/** **packageBooking**

If a customer wants to book a package from available packages this functionality is implemented. A booking entry is added in the booking table against the customer\_id and package\_id

**Validations / Conditions checked–**

1. Customer should be logged in

2. Package should exist

**9: Bibiliography**

“20.12. Smtplib — SMTP Protocol Client — Python 2.7.17 Documentation.” Python.Org, 2020, docs.python.org/2/library/smtplib.html. Accessed 6 Jan. 2020.

Copperwaite, Matt, and Charles Leifer. Learning Flask Framework : Build Dynamic, Data-Driven Websites and Modern Web Applications with Flask. Birmingham, Uk, Packt Publishing, 2015

Dwyer, Jack. FLASK : Building Python Web Services. 2017.

Lutz, Mark. Programming Python. Sebastopol, Ca, O’reilly, 2006.

Mufid, Mohammad, et al. Design an MVC Model Using Python for Flask Framework Development.

Selfa, Diana, et al. A Database and Web Application Based on MVC Architecture.

“Welcome to Flask — Flask Documentation (1.1.X).” Palletsprojects.Com, 2010, flask.palletsprojects.com/en/1.1.x/.