

Airline Enterprise Solution - Fly Like T6

G7T6

**IS213 Enterprise Solution Development**

Assignment

Bak Ing Sin

Bui Phuong Thao

Loh Xiao Binn

Teo Jia Cheng

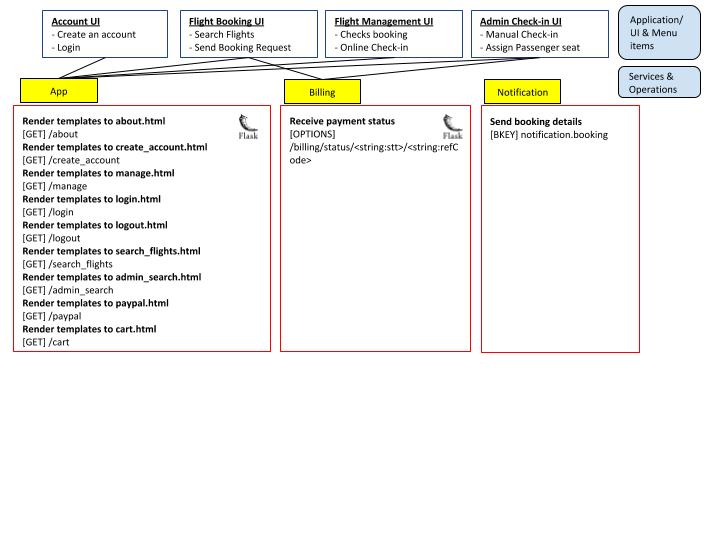
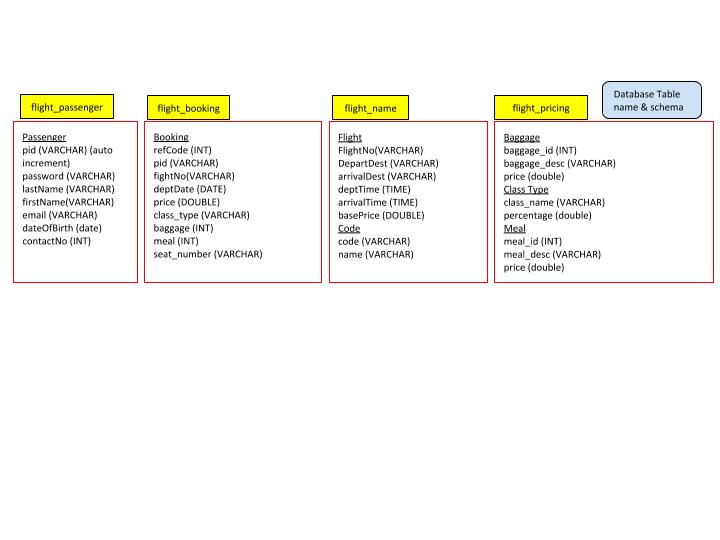
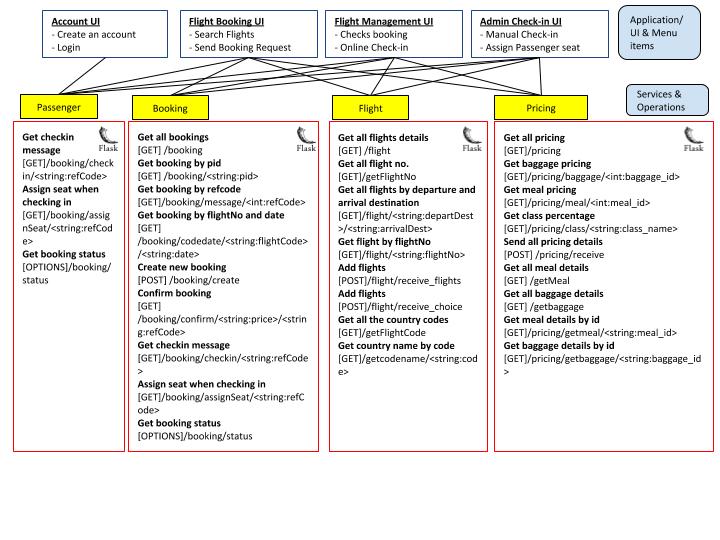
Vo Pham Thao Vi

# 

# Introduction

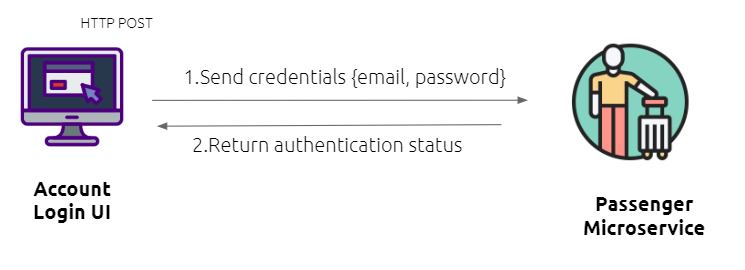
Fly like T6 is an airline company which provides flight services with a streamlined online booking system. Users must have an account and passengers will be able to create accounts, login/logout, make a booking, view their bookings, check-in online and view their boarding pass. Users will be notified by email upon the confirmation of a booking. Payments are done through Paypal or Credit/Debit card. Flight admins will also use this system to help passengers to check-in offline.

# Technical Overview Diagram



# User Scenarios

## [User Scenario 1 - Login]



1. Passenger logs in with his existing account via email and password. Upon clicking the “Login” button, the UI invokes the Passenger microservice via HTTP post to verify the credentials by matching hashed password.
2. Upon verification, returns the login status with a status code back to the UI.
3. Upon successful login, passenger’s PID will be stored in session using Javascript.
4. To log out, the passenger will click on the “Logout” button and the UI will remove all session storage items, including PID, and direct the user back to the home page.

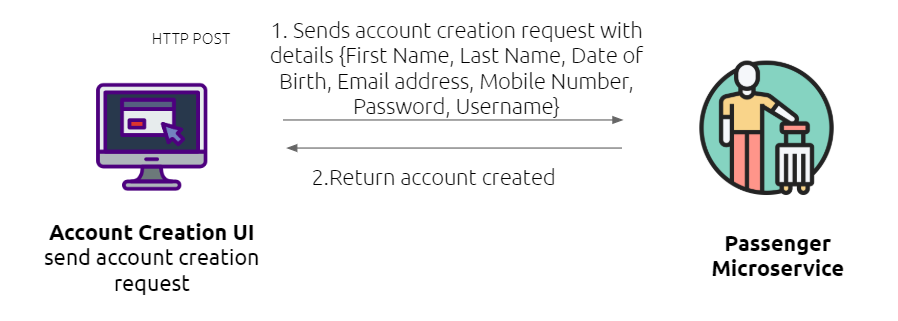
### (Micro)Services - used in this scenario only

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Service | Operational information | Description of the functionality | Input | Output |
| *Passenger* | ***Verify passenger credentials during login***  *[POST] /passenger/login* | *Obtain passenger’s email and password from UI login form and verify passenger’s credentials* | *{email, password}* | *{login status}* |

### Beyond the Labs - Password verification (hashing)

Check if a password inputted by a passenger matches an existing hashed password in a database which is generated during account creation by using Passlib hashing library.

## [User Scenario 2 - Account Creation/ User sends a request to create an account]

**

1. Passenger starts/sends the account creation request by filling in the required fields on the form and then clicking on a button on the Account Creation UI. UI invokes the Passenger service via HTTP POST to add a new account.
2. The Passenger service saves the passenger details into the Passenger database, then returns a status code with the passenger details and the user is brought to the welcome page.

### (Micro)Services - used in this scenario only

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Service | Operational information | Description of the functionality | Input | Output |
| *Passenger* | ***Add new passenger*** *[POST]/passenger/register/<string:email>* | *Takes in the details sent through the create account form. Rejects account creation if email exists. Else, password entered is hashed and passenger details along with hashed password is added to database.* | *{email, password, pid, firstName, lastName, dateOfBirth, contactNo}* | *Passenger details {email, pid, firstName, lastName, dateOfBirth, contactNo},{status of the account creation}* |

### Beyond the Labs *-* Password hashing (from passlib.hash import sha256\_crypt, pbkdf2\_sha256)

Password is hashed before being added to the database to store the password securely. The function used is password\_hashed = sha256\_crypt.hash(pwd)

## [User Scenario 3 - Create Booking]

### 

1. A passenger starts the booking process by selecting their departure and arrival destination from a list of dropdownlist, and clicks on “search flights”. The UI invokes the Flight microservice via HTTP GET to get flight details which returns a list of available flights to the Flight Booking UI.
2. The passenger will select a desired flight, the date of flight, flight class and add on preferences.
3. UI invokes Booking microservice via HTTP GET to retrieve add-on prices from Pricing microservice. Pricing microservice returns add-on price to Booking microservice.
4. Booking UI gets the calculated total price from the Booking Microservice, displays the booking details and total booking price.
5. When the passenger confirms the booking, UI invokes Booking miroservice via HTTP POST to create booking.
6. When the passenger proceeds to pay, Booking microservice calls the Billing microservice which calls the Paypal API and the payment status will be returned to Flight Booking UI.
7. Upon successful payment, booking microservice requests for flight details and passenger details from flight and passenger microservice respectively and sends these details to Notification microservice.
8. Notification microservice prepares email body and calls MailJet API to send booking confirmation to the passenger and return the email status back to the notification microservice to display in the terminal.

### 

### (Micro)Services

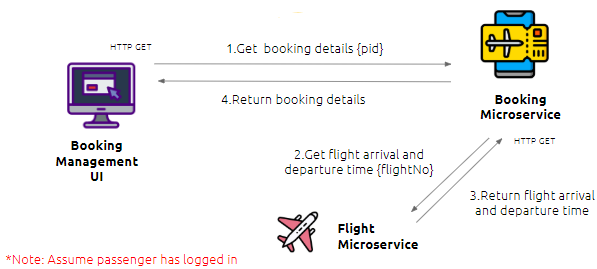
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Service | Operational information | Functionality description | Input | Output |
| *Flight* | ***Get all flight code***  *[GET] /getFlightCode* | *Get list of available flights* |  | *list of available flight details {code, name}* |
| ***Get all flight code by departure and arrival***  *[GET]/flight/<string:departDest>/<string:arrivalDest>* | *Get specific flights* | *{departDest, arrivalDest}* | *list of specific flight details {flightNo, departDest, arrivalDest, deptTime, arrivalTime, basePrice}* |
| ***Get flight info by flightNo***  *[GET]flight/<string:flightNo>* | *Get flight info in order to email the passenger* | *{flightNo}* | *flight details {flightNo, departDest, arrivalDest, deptTime, arrivalTime, basePrice}* |
| *Booking* | ***Create booking***  *[POST]booking/create/* | *[No additional description required as the Operation is clear enough]* | *{PID, flightNo, base\_price, departDate, class\_type, baggage, meal}* | *booking details {refCode, PID, flightNo, departDate, total price, class\_type, baggage, meal}* |
| ***Get booking status***  *[OPTIONS] /booking/status* | *[No additional description required as the Operation are clear enough]* | *{status, refCode}* | *{status, refCode}* |
| *Pricing* | ***Get meal pricing***  *[GET]/pricing/getmeal/<string:meal\_id>* | *[No additional description required as the Operation is clear enough]* | *{meal\_id}* | *meal details {meal\_id, meal\_desc, price}* |
| ***Get baggage pricing***  *[GET]/pricing/getbaggage/<string:baggage\_id>* | *{baggage\_id}* | *baggage details {baggage\_id, baggage\_desc, price}* |
| ***Send pricing details***  *[POST]* | *Send pricing details of meal, baggage and class* | *{meal\_id, baggage\_id, class\_type}* | *price details {meal\_price, baggage\_price, class\_type percentage}* |
| *Billing* | ***Receive payment status*** *[OPTIONS]/billing/status/<string:stt>/<string:refCode>* | *[No additional description required for this operation as the Operation is clear]* | *{status, refCode}* | *{status, refCode}* |
| *Passenger* | ***Get Passenger***  *[GET]/passenger/<string:pid>* | *Get information about a passenger for email notification* | *{pid}* | *passenger details {pid, firstName, lastName. email}* |
| *Notification* | ***Send booking details***  *[BKEY] notification.booking* | *Send booking details and call Mailjet API, fire and forget* | *{firstName,lastName,email, refCode,flightNo,departDate, deptTime,price,class\_type,seatno}* |  |

### Beyond the Labs

### MailJet API - We obtain an unique API key to call Mailjet Api to send booking confirmation email to passenger’s email. The email body content includes passenger’s name, flight number, departure date time and reference code.

### Paypal API - External Paypal API has been used for booking payment. Total booking price and reference code are the input parameters and the API returns payment status and reference code as output.

## [User Scenario 4 - View Booking on Manage Booking UI]

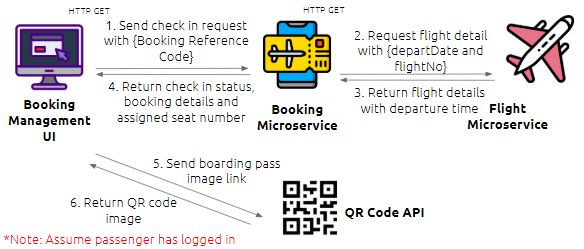
**

1. Passenger starts the request by clicking on the Manage Bookings Tab, then the UI invokes the Booking service via HTTP GET to get the booking details.
2. Upon receiving a UI request, the Booking service invokes the Flight service via HTTP GET to get the flight details (flight arrival and departure time).
3. The Booking service returns the booking and flight details to the UI.
4. If there is no seat assigned to the passenger for that flight booking, there will be a button reflected in that row called “Check in”, else, it will reflect the seat number. A passenger is only assigned a seat upon check in.

### (Micro)Services

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Service | Operational information | Description of the functionality | Input | Output |
| *Booking* | ***Get booking by pid***  *[GET]/booking/<string:pid>* | *Takes in PID and returns a list of all bookings with a matching PID and returns a list of all the matching bookings with the relevant details to be reflected on the UI.* | *{pid}* | *[{ baggage, class\_type, departDate, flightNo, meal, pid, price, refCode, seat\_number},...]* |
| *Flight* | ***Get all country codes***  *[GET] /getFlightCode* | *Gets list of all country codes and its respective name* |  | *[ { code, name }.....]* |
| ***Get flight information by flight number*** *[GET]/flight/<string:flightNo>* | *Takes in a Flight number (eg, MH102) and returns the flight details and request status* | *{flightNo}* | *{ "flight": {arrivalDest, arrivalTime, basePrice, departDest, deptTime, flightNo}, status}* |

## [User Scenario 5 - Online Check in]



1. Passenger starts the request by clicking the Check In button on Booking Management UI, then the UI invokes Booking microservice via HTTP GET to process this request.
2. If the request is accepted, Booking microservice assigns a seat for the passenger, then invokes Flight microservice via HTTP GET to get flight details (departure time).
3. Booking microservice return booking details and flight details to the UI to display on the boarding pass.
4. The UI sends the image link to QR Code API to get the QR Code image to display on the boarding pass.
5. The UI displays the boarding pass to the passenger so that he/she can scan before boarding.
6. If the request is not accepted, the passenger will be redirected back to the Booking Management UI and he/she has to proceed to the check in counter to check in offline.

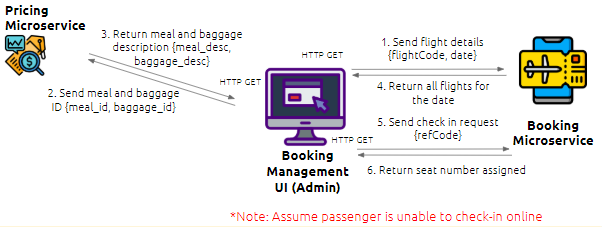
### (Micro)Services

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Service | Operational information | Description of the functionality | Input | Output |
| *Booking* | ***Get check-in message***  *[GET]/booking/checkin/<string:refCode>* | *Process check in request (Simulated Passport Scanning API for verification using random as it is charged)* | *{Booking Reference Code}* | *{check in status}* |
| ***Assign seat*** *[GET]/booking/assignSeat/<string:refCode>* | *Assign seat to passenger for that booking identified by booking reference code* | *{refCode}* | *{seat\_number}* |
| *Flight* | ***Get flight by flightNo***  *[GET]/flight/<string:flightNo>* | *Get flight details* | *{flightNo}* | *{departTime}* |

### Beyond the Labs *-* QR Code API

To resemble the journey of a passenger from booking to boarding, passengers will scan the QR code before boarding like in the airport where they have to scan their boarding pass at the gantry before actually getting into the plane.

# [User Scenario 5 - Offline Check in by Admin]



1. Passenger shows the confirmation email to admin.
2. Admin starts the request by selecting the departure date and flight code as stated in the email on the Booking Management UI (Admin) UI. By clicking “Search Flight”, the UI invokes Booking Microservice via HTTP GET to get the booking details of selected bookings and returns the bookings details to the UI.
3. Upon receiving a UI request, the Booking service invokes the Pricing microservice via HTTP GET to get the meal and baggage description of each booking which returns the details to the UI.
4. After manually verifying passenger’s information (passport check), the admin can click on the “Assign Seat” button whereby the UI then invokes Booking microservice to assign a seat for the booking which then returns a seat number to the UI.

### (Micro)Services

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Service | Operational information | Functionality description | Input | Output |
| *Booking* | ***Get booking by flightNo and date*** *[GET] /booking/codedate/<string:flightCode>/<string:date>* | *[No additional description required for this operation as the Operation, Input & Output are clear enough]* | *{flightCode,date}* | *{booking details}* |
| ***Assign seat*** *[GET]/booking/assignSeat/<string:refCode>* | *{refCode}* | *{seatNumber}* |
| *Pricing* | ***Get all meal details***  *[GET] /getMeal* |  | *{all meal details}* |
| ***Get all baggage details***  *[GET] /getbaggage* |  | *{all baggage details}* |

# Remaining Beyond the Labs not covered above

* Deploy a **Docker** Container on Docker Hub at [https://hub.docker.com/r/vptv1310/flightapp](https://hub.docker.com/r/thaobui/flightapp) which can be pulled using “docker pull vptv1310/flightapp” command. The Docker Repository contains 6 image tags of 6 microservices (app, pricing, flight, passenger, booking, billing).
* Use **Docker Compose** to deploy multiple microservices at once, using command “docker-compose up” in flightapp directory. However, we encounter a ConnectionError (Max retries exceeded with url) when connecting to booking and billing microservice. Still, you can explore other functions from our microservices based on our technical overview diagram.
  + (Note that the error would occur only when pulling the image from the Docker Hub mentioned above, so everything should work fine if you follow our README.md file to run on your localhost)
* Use **Amazon Relational Database Service** (RDS) to allow remote access from everyone in our project team. Therefore, any changes in the database when we work on the project can be updated instantly. Also, this enables us to go deployment without worrying about database dependency. We don’t need to package any local database from our machines to deploy.
* Use **Jinja** (Python template language) so as for microservices, which are developed on Flask framework, to evoke UI template. Also, it provides us with a consistent template throughout the web pages by extending a base template layout.html. Logics and contents are wrapped in blocks (content, script, title, …) in child templates. In other words, Jinja provides template inheritance for code reusability and readability.