COMP3011 Web Services and Web Data

Team 2: Flight Search and Booking System

Table of Contents

1. Detailed analysis of the business situation (background research)

1.1 - Airline research

Airlines allow for independent flight bookings, where a user is limited by the range of available flights, due to only the given airline's flights being available. The user would select the flights that they are interested in, and later are able to select specific seats and luggage arrangements to apply to their travel plans. Prior to the travel date, the user is also prompted to 'check-in', where they confirm personal details.[1]

Payments are handled by external vendors, by utilising their payment APIs to handle payment transactions, with different airlines providing a range of different payment options to choose from.[2] Once this payment has been confirmed, the user will be sent confirmation of their booking to the email account that was used for the booking process.

When looking at buying a ticket, most airlines will reserve a seat once a user selects a seat and starts the payment process.[3] Our application does not allow for selection of seats, and instead, we will be creating a temporary reservation of seats for a flight once the user starts the payment process within the airline API.

In the real world, flight providers will interact with aggregators by providing their flight details to different Global Distribution Systems [GDS]. These systems are then responsible for sharing this information with the aggregators, that will display these flights to their users and the aggregator will then book flights with an airline for the user, but the user will still later check in with the airline directly.[4]

In terms of cancelling bookings, most airlines allow users to manage their bookings without accounts, allowing users to book their current bookings by providing a booking ID and some form of personal details, such as their last name or email address used when making the reservation.

1.2 - Flight Aggregator research

The first step required by a flight aggregator to be successful is to collect relevant information from the user booking the flight. For example, the aggregator Skyscanner[5] requests the following information:

- Origin and Destination – The user can select a city, country or just an individual airport for both the origin and destination of their journey.
- Departure Date – The user can select either one day or an entire month as their departure date.
- Service Class – The user can choose between economy, premium economy, business class and first class, each differing in price point and on-flight amenities.
- Ticket type and number of tickets – The user can choose how many adult/child tickets they would like to buy.

The aggregator then uses this information to query all the airlines they are associated with through APIs and receive available flights matching the criteria. For example, airlines such as British Airways, Easy Jet, and Air Europa, would return available flights from London Heathrow to Madrid. All the flights meeting the requirements are shown to the user with information including airline, departure/arrival time, date, flight duration, departure/arrival airport, and flight cost.[6]

There is an option to filter for every flight detail, for example one could exclude flights which are longer than 3 hours and leave after 6pm in the evening. This feature essentially works by adding additional parameters to the queries made to airlines. Furthermore, the user has the option to sort by cost, duration, or departure time.

If the user finds a flight which satisfies all their requirements, aggregators such as Skyscanner, Kayak and Google Flights will redirect them to the airline (or other booking platform) to complete the booking.

However, with some aggregators such as Booking.com they user can select their seats, baggage, and other details before the payment process begins.[7]

Once the payment has been completed and the booking is confirmed, the aggregator itself sends a confirmation email to the user which includes the electronic tickets. If the user has an account and is logged in, they will be able to see this booking along with flights they have previously booked and manage them accordingly.

Overall, flight aggregators provide an efficient way for users to search and book flights by collecting and comparing data from various airlines. This makes it easier for a user to find the best deals, saving them time and money.[8]

1.3 - Payment research

Background research on flight aggregators such as Sky Scanner has shown that after having chosen their flight and specified the search parameters e.g., seat number and number of passengers, the users is redirected to the payment page. There, the user is given the choice of several companies through which to book the flights such as Mytrip, GotoGate and BudgetAir. The user is then given the choice between four payment options: credit/debit card, Apple/Google Pay, PayPal, and Klarna.[9] Once selected, each of these options provides its individual interface.

Credit/Debit card – all major debit and credit cards are accepted such as Visa, Mastercard, and American Express. The details that the user must provide include: 15/16-digit card number, expiry date, and 3/4-digit CVV code. The first digit of the card number distinguishes between different card types, for example, Visa cards always start with the number 4, while Mastercard cards start with the number 5. The final digit of the card number is a check digit and is used to validate the authenticity of the credit/debit card number based on the Luhn algorithm.[10]

Apple/Google Pay – both methods serve as a more user-friendly interface to the traditional card payment method. If a user has already added their card details to the wallet app on their device, when making a payment, the user must verify the transaction. Verification options include receiving a verification code from their bank or verifying their identity through biometric verification options if available. This provides a more convenient payment method to most users due to the ease of access.[11]

PayPal – works similarly to Apple/Google Pay as a user has the option to add their card details, however, it is verified through logging into your PayPal account using an email and password. PayPal also provides its own balance that users can top-up using their credit/debit card, which can also be used as a form of payment. When confirming the payment, the user is given the option between using their PayPal account balance or their credit/debit card.[12]

Klarna – similarly to PayPal, Klarna will prompt a user for their login credentials and provide them with several instalment options. One example is ‘Pay in 4’ where a user pays an initial sum at checkout and pays the remaining over the next six weeks in two-week increments.[13] Late fees are applied when a payment is not made on time as most of their payment options are interest-free. A credit check is done on users by Klarna to check whether a user is eligible for a specific payment plan. A user will be reminded throughout the plan of upcoming payments.

After a payment has been completed, a transaction confirmation or invoice is sent to the user via email, SMS, or other means. Competitors provide their users with a variety of payment options, improving the ease of use of the websites and in turn improving customer experience.

2. Detailed description of the working principle of the proposed web-based system

2.1 High-level overview of the payment system

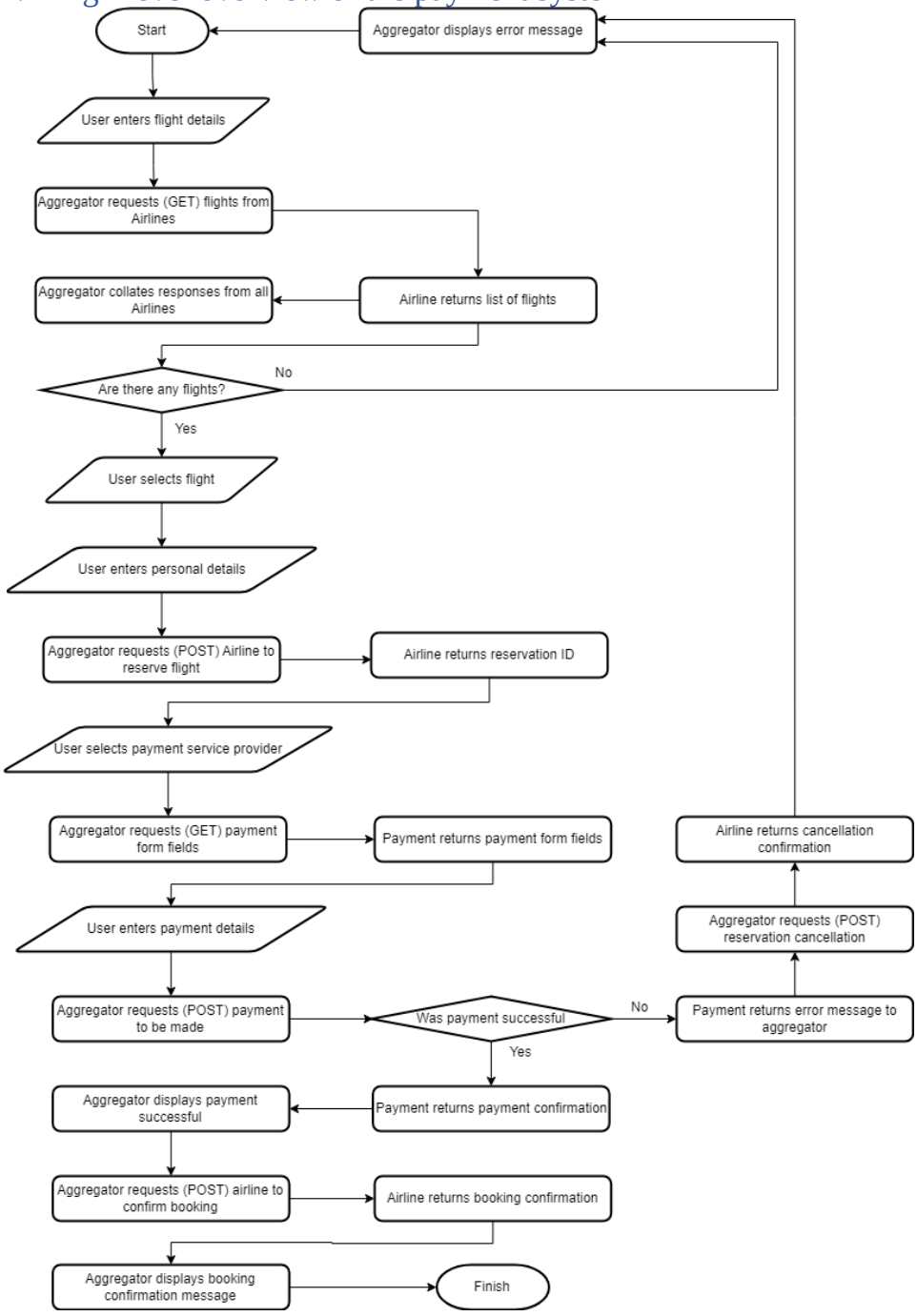

Figure 2.1.1 High-level flowchart of flight booking process

Figure 2.1.1 shows a high-level flow of events and API interactions in our proposed system. As the flight aggregator is the only client interface, this overview focuses on the system interactions from the aggregator's perspective. When a user first opens the flight aggregator application, they will be asked to enter flight details. The aggregator will then send get requests to each airline with these details to obtain a list of flights it can display to the user. Once the user picks a flight from the list, the aggregator will send a post request to the airline temporarily reserve the seat(s).

The user will then be presented with a list of payment option. The user choosing a payment option will initiate a handshake where the aggregator will send a GET request, a receive a list of payment form fields from the given payment service provider. The aggregator will then prompt the user to fill in the payment form and send a POST request to the payment API which will process the payment. If the payment is successful, the aggregator will send a POST request to the airline to confirm the booking, otherwise it will display an error message.

The following sections will explain in-depth how each sub-component of the system will be implemented.

2.2 Aggregator Description

2.2.1 Acquiring Flights

For users to search for flights across all 4 airlines the aggregator needs to query each airline API, passing the travel date, location and passenger number preferences. To do this it will issue a GET request to the airline request information on all flights, fitting the filtered query, available from that airline. Metadata information should include:

- Date
- Price
- Origin and Destination (airport/city)
- Available seats
- Type of seats, first class etc.
- Flight duration
- Estimated Time of Departure (ETD)
- Airline

A function '*search()*' should systematically search all airlines for matching flights and temporarily store all the above information. An example of this can be seen below:

Flights = [

['origin 0', 'destination 0', 'date 0', 'ETD 0', 'available seats 0', 'type of seat 0', 'price 0', 'flight duration 0', 'airline 0'],

['origin 1', 'destination 1', 'date 1', 'ETD 1', 'available seats 1', 'type of seat 1', 'price 1', 'flight duration 1', 'airline 1'],

['origin 2', 'destination 2', 'date 2', 'ETD 2', 'available seats 2', 'type of seat 2', 'price 2', 'flight duration 2', 'airline 2']

]

2.2.2 Selecting your Flight

The user should then be able to browse from a displayed list of available flights that match their search. Therefore, the function *'search()'* should pass the following command line arguments, provided by user input:

- Departure date
- Origin
- Destination

An example of the interface can be seen in figure 2.2.2.1 below.

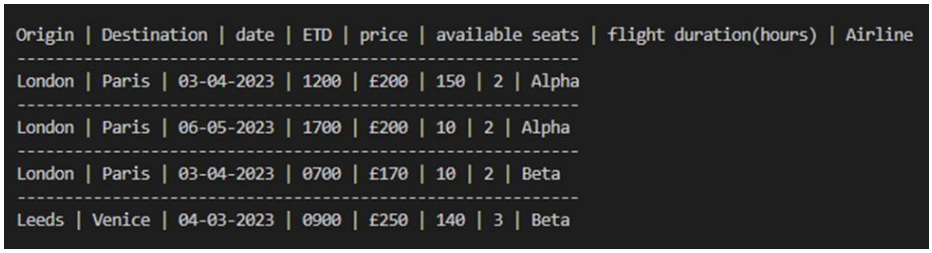

Figure 2.2.2.1 Flight selection menu

Once the correct information is displayed, the user should then be able to sort the data, using the *'sort()'* function, according to the parameters the user chooses, such as:

- Price
- Flight duration
- Time of departure
- Airline

We could use a known sorting function based on this information, such as quick sort.

2.2.3 Booking and Paying for a Flight

Once the user has decided on the flight they wish to take, they should then be able to book a specific ticket on a given flight. This should connect the user to the payments API and should pass on all the necessary payment information.

A given ticket should possess the following information:

- Seat number
- Seat type

Once the booking process has begun a temporary reservation is made for that flight. Passenger and selected flight information is sent to the airline API, and a pending ticket is created in the database.

The user should also be able to specify which payment type they wish to use.

Once this ticket has been paid for and a confirmation is received from the payment API, a confirmation is sent to the airline API, and the ticket status is changed from pending to confirmed, while the number of seats left on the corresponding flight is updated. If this was the last seat on the plane, the flight will no longer be displayed as an available flight in feature search queries.

(It is a given that a user should not be able to buy more tickets than available seats on the flight despite the flight being available).

The function *'book_flight()'* should first request the required details for the chosen payment service provider (PSP), then prompt the user for command-line input to provide these details, before sending them to the selected payment API to be verified.

For example, a standard, card-based PSP would request:

- Card number
- Sort code
- CVV
- Billing address
- Email

- Transaction amount
- Name of buyer

```
Enter origin and destination of flight: London Paris
Enter date of flight (format: dd-mm-yyy): 03-04-2023
Here are your flights:

Origin | Destination | date | ETD | price | available seats | flight duration(hours) | Airline
-----------------------------------------------------------------
1: London | Paris | 03-04-2023 | 1200 | £200 | 150 | 2 | Alpha
-----------------------------------------------------------------
3: London | Paris | 03-04-2023 | 0700 | £170 | 10 | 2 | Beta


Enter the flight number you wish to select: 1
You have selected flight:
 ['London', 'Paris', '03-04-2023', '1200', '£200', '150', '2', 'Alpha']
We have created a reservation for you.
1 - Direct Debit
2 - Paypal
3 - Klarna
Please enter payment type: 1
You have selected:  Direct Debit
Please enter card details:
```

Figure 2.2.1 Example flight search results

Once payment details are verified and the transaction is completed, a confirmation message is sent back to the aggregator, that is then displayed to the user and passed to the airline API to confirm the pending status of the reserved flight in the database. Finally, the aggregator will display an electronic ticket including crucial information such as

The user's flights are now fully booked, and they are ready to fly!

2.2.4 Cancelling a booking

When the user first launches the aggregator, they will be given the choice to either make a new booking or to cancel an old one. If a user wishes to cancel their booking, they will be able to do so through the aggregator. As user accounts will not be stored by the aggregator, the user will need to provide the reservation ID shown on their electronic ticket as well as the transaction ID from the invoice sent by the payment service.

The aggregator would then send a POST request to the airline providing it with the reservation and transaction IDs. If the reservation ID is valid, the airline sends a POST request to the payment API which checks the transaction ID against its database. If the transaction ID is correctly verified the payment API reverses the transaction on its side, while the bank API is requested to do the same.

If everything is completed successfully, a series of success messages is sent by each successive API until the aggregator receives and displays a cancellation confirmation message. The booking has been cancelled and the user has been refunded, however the refund policy and amount will be decided by each airline individually.

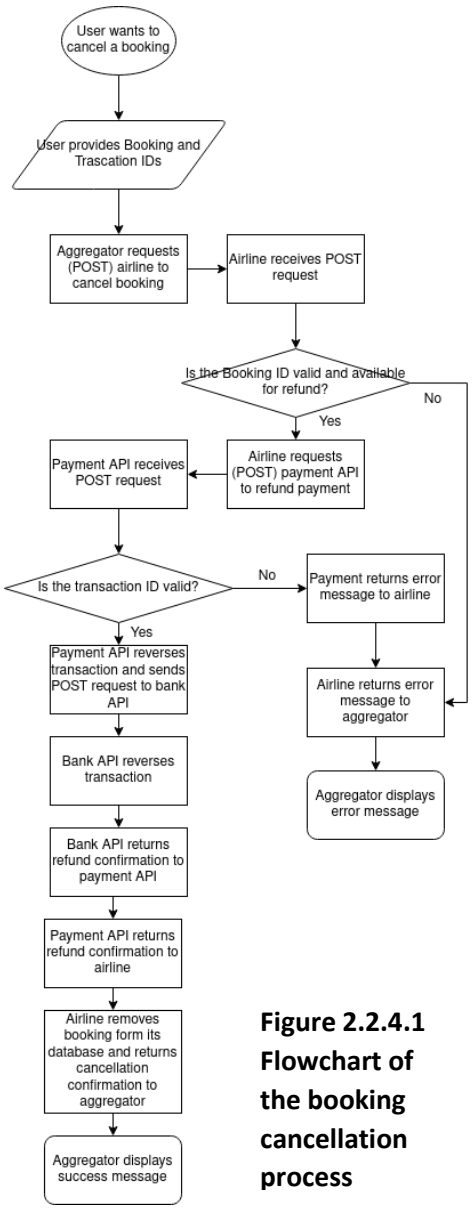

Figure 2.2.4.1 Flowchart of the booking cancellation process

2.3 - Payments Description

2.3.1 Payment Process Overview

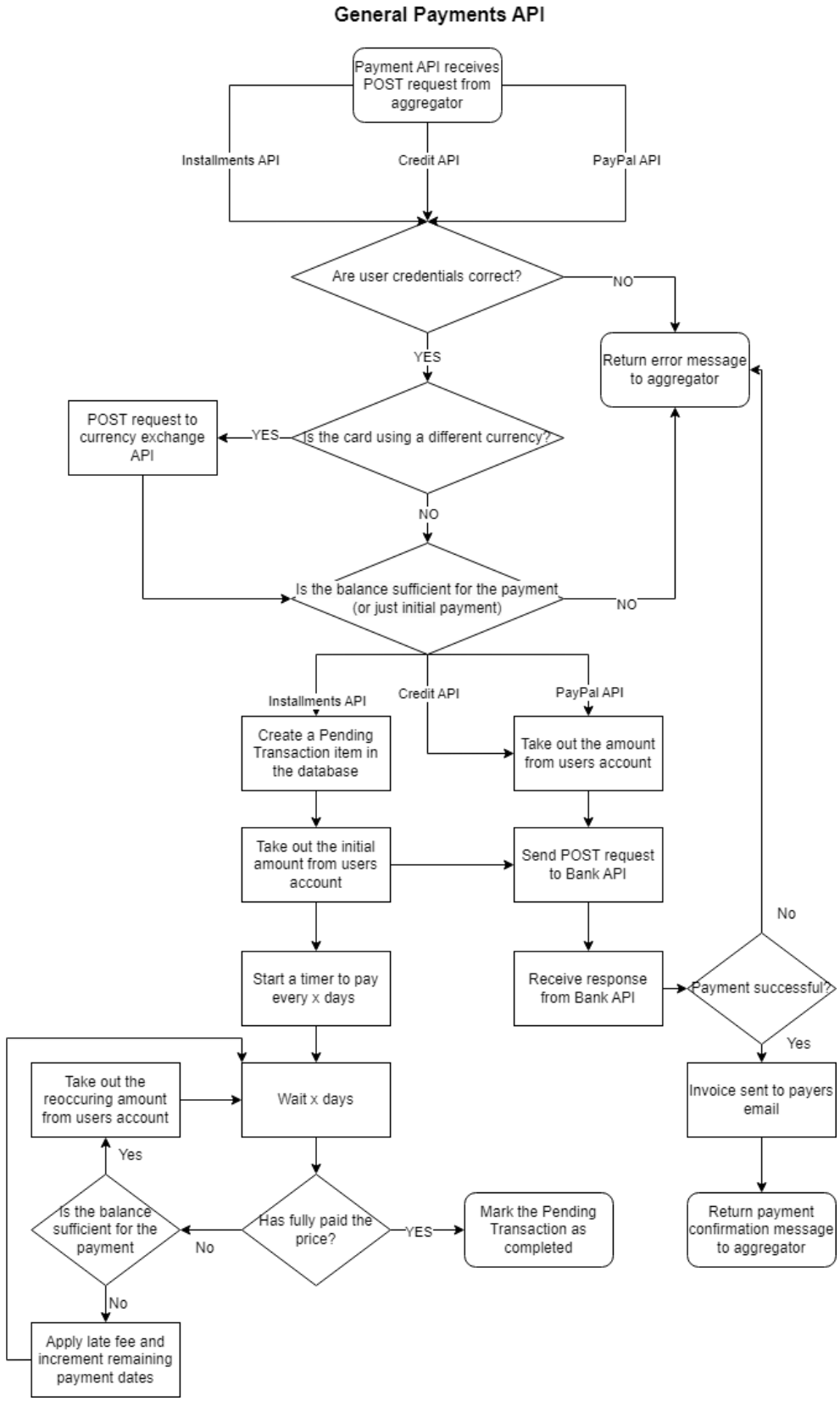

Figure 2.3.1 Flowchart of the payment process

All APIs under Payment category will be implemented using the Django python framework and REST architecture. We have decided to build 3 APIs handling 3 different types of payments: instalments, direct debit and PayPal. As well as this, a 4th API that will mimic a real bank will be implemented, so that the testers don't have to use real money.

The bank API will have 2 tables. One will be prefilled with credit accounts of flight providers and the other one will be used for storing completed transactions. The API will have 1 POST endpoint to handle payments. After getting the POST payment request with flight details, user's bank details and cost amount it will add the cost amount to flight providers bank balance, create a new Transaction record with all provided data, and make a POST request to the payment service provider confirming a successful transaction.

The user flow of all 3 payment APIs will start the same way. From the client, based on the payment type, a GET request will be sent to a certain API and it will return form fields that will need to be filled for that payment type. The client will then send a POST request with the form data to one of the payment type APIs.

Any API will check if a card with those details exists, if the money is in a different currency, it will use an external currency exchange API to convert the money and then if the user has enough balance. If any of the checks fail, we will notify the user with a helpful error message indicating at which point did the request fail. If we pass the checks, then the logic starts to differ.

2.3.2 Credit or PayPal

They will use individual databases of users' accounts; however, both will take out the money amount from the user's balance and send an invoice to user's email. After all this, it will send a POST request to the Bank API with flight details, user details and cost amount.

2.3.3 Instalments

It will first create a 'PendingTransaction' item in the database with the provided fields and then send an invoice to user's email. Because we're paying in instalments, it will send the full amount to the Bank API to book the flight and then charge the initial amount from the user. This is where we will start a scheduler to wait for a select number of days and then if the user hasn't paid the full amount, we will charge again. This loop will continue until the user has paid the full amount and then we will mark the pending transaction as complete.

2.4 – Airline Description

2.4.1 - Overview of Airline sub-component.

The Airline API will contain 2 tables: a table holding all the reservations and a table containing all the flights. The reservations table contains many instances of the reservations class that are either in progress or completed. There are no cancelled reservations stored as, if the reservation is not completed, it will be purged from the table.

The Airline API will only communicate directly with the flight aggregator. The aggregator will POST the details of a client's requested flights which the airline will then GET. After that, the Airline API will POST a list of flights suiting the client's request to the aggregator. At this point, the client should select a flight along with how many tickets they wish to purchase that is then POSTED to the Airline API. The Airline will then add this reservation to the reservations table. If the booking is successful, the Airline will GET a confirmation from the aggregator, if not, it will GET a cancellation of the reservation from the aggregator.

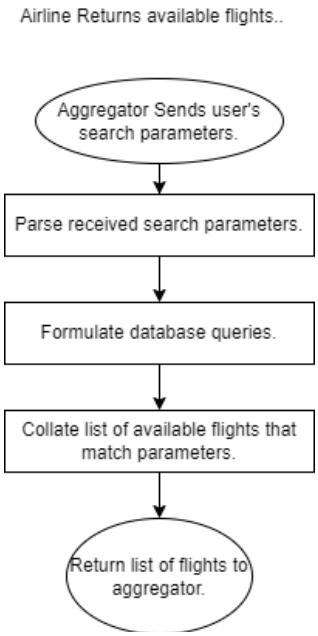

Figure 2.4.1.1 Flowchart of the airline returning flights

2.4.2 - Flowcharts describing Airline sub-component processes.

2.4.2.1 Airline Sends list of flights to Aggregator:

Once the aggregator receives the user's flight preferences, these will then be relayed to the airline. The airline will first check for any reservations that have exceeded the 10-minute reservation appeal. This is necessary, as the reservations ensure concurrent bookings do not cause oversubscription issues, but need to be checked regularly to ensure flight seats are not being lost to unfinished/failed reservations. To do this, the airline will query the reservation table for any flights that have the confirmed booking set as 'False' Once this occurs, the airlines will take the user preferences and query the flight database, returning the matching flights back to the aggregator.

2.4.2.2 Airline makes temporary reservation during booking process

Once the airline receives the reservation request from the aggregator, flight availability will be checked. This includes both checking for expired reservations, and checking the requested flight has enough seats available for the user. If this flight is no longer available, the airline will respond with an error message to the aggregator. Otherwise, the airline will be responsible for generating a unique Reservation ID, which is used in multiple stages of the booking process. This reservation ID is then used to make a temporary reservation within the reservations table, with the confirmed status being set to 'False'. The seat availability on that flight is updated, and the airline returns the reservation ID to the aggregator, alongside a confirmation message that the reservation has been made.

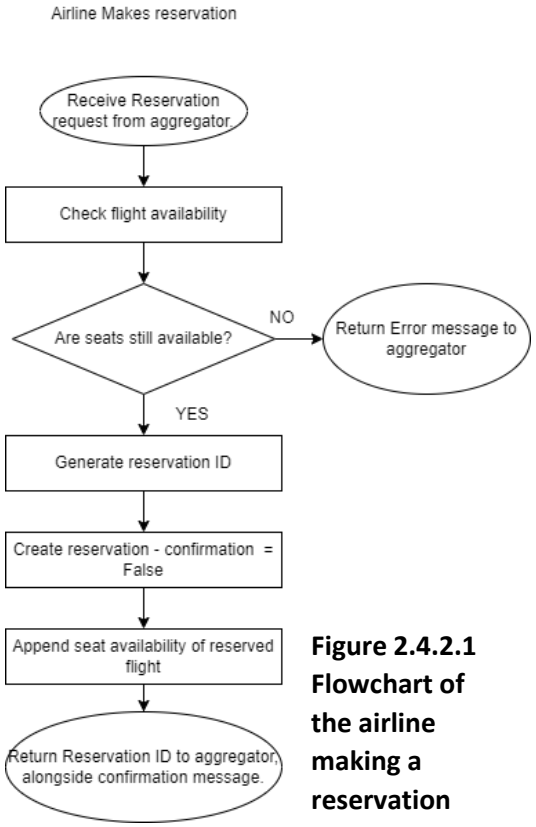

Figure 2.4.2.1 Flowchart of the airline making a reservation

2.4.2.3 Airline finalises booking process.

Here, the aggregator will send a payment update to the airline, which contains two possibilities: payment successful and payment failed. If the payment has failed, the airline will look for the flight that this concerns using the reservation ID and reassign the reserved seats as available for booking. The airline will then remove this reservation from the reservation table and send a response to the aggregator that confirms this has occurred.

If the payment was successful, the airline will need to check if the reservation is still active and has not timed out. If the payment has timed out, the airline will notify the aggregator that the booking can no longer proceed. If the reservation exists, the airline will append the confirmation status within the booking to 'True' and email the user a booking confirmation using the email that the reservation was made under. Finally, the airline will return a booking confirmation to the aggregator.

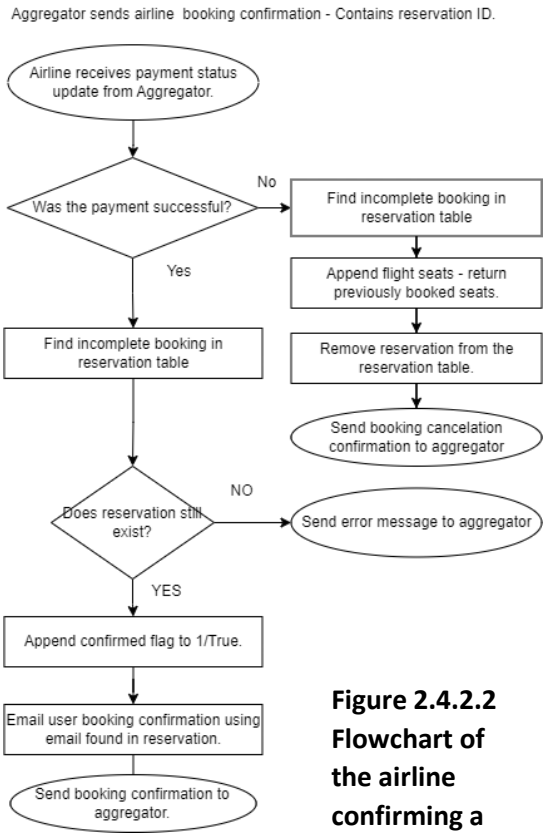

Figure 2.4.2.2 Flowchart of the airline confirming a booking

3. Databases

3.1 - Payments

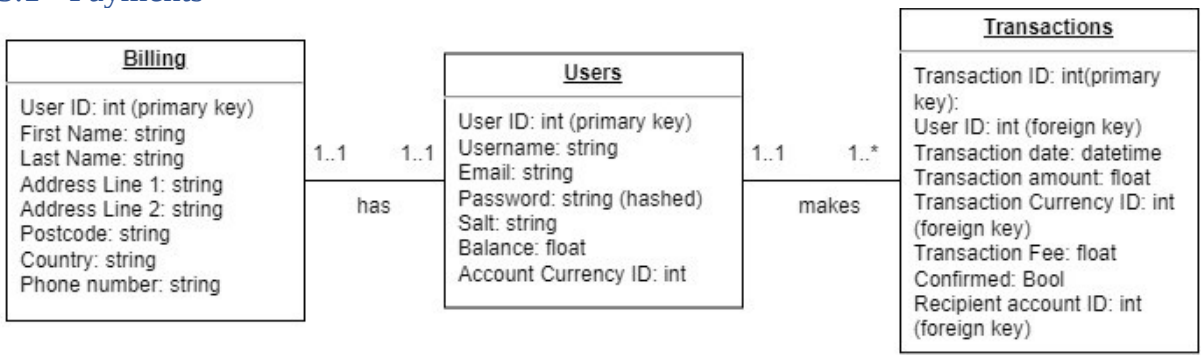

Figure 3.1.1 database schema for account-based PSP

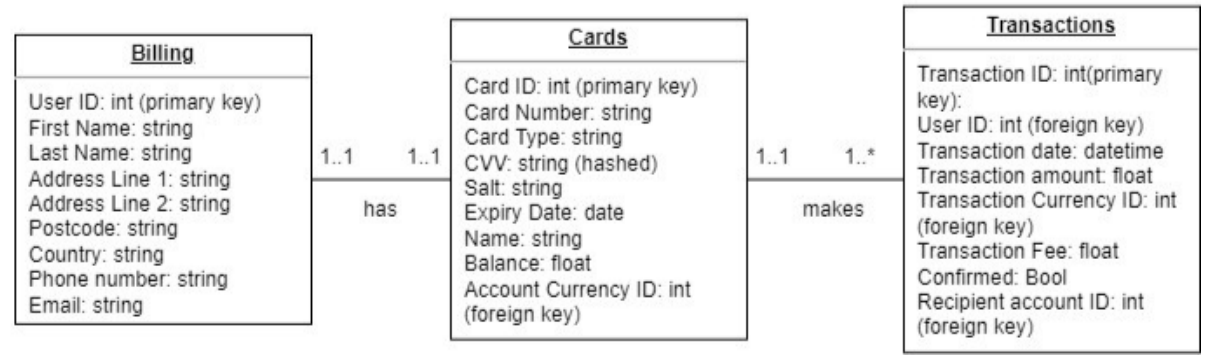

Figure 3.1.2 database schema for card-based PSP

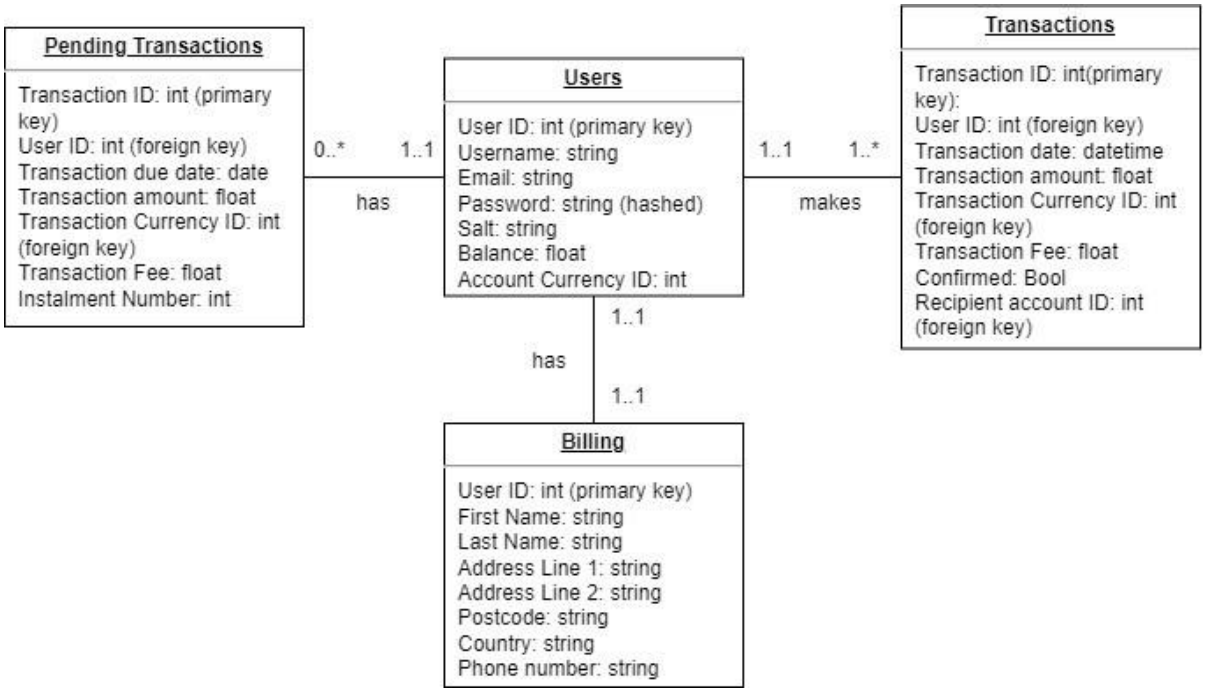

Figure 3.1.3 database schema for instalment-based PSP

As the payment service providers (PSPs) will vary in both their business logic and implementation in our proposed system, three distinct database schemas have been created: one for each payment service provider category. Each category of payment API will share two database tables in common: the *Billing*

and *Transactions* tables. The former will store billing information about each payer whereas the latter will hold a history of past transactions.

The main difference between the three different PSPs will be the columns of each respective *Users* table. For both the account-based and instalment-based PSPs, the *Users* table stores each user's login credentials including a hashed password and a random salt for security. In the card-based PSP the *Users* table is replaced by a *Cards* table storing credit/debit card details including a hashed card verification code (CVC) for user authentication.

The instalment-based PSP will have an additional *Pending Transactions* table reserved for future transactions that are yet to be completed. Each pending transaction will have a due date. Once the due date passes, the transaction is completed, and its details are moved to the *Transactions* table. It is important to note that a user can have zero pending transactions.

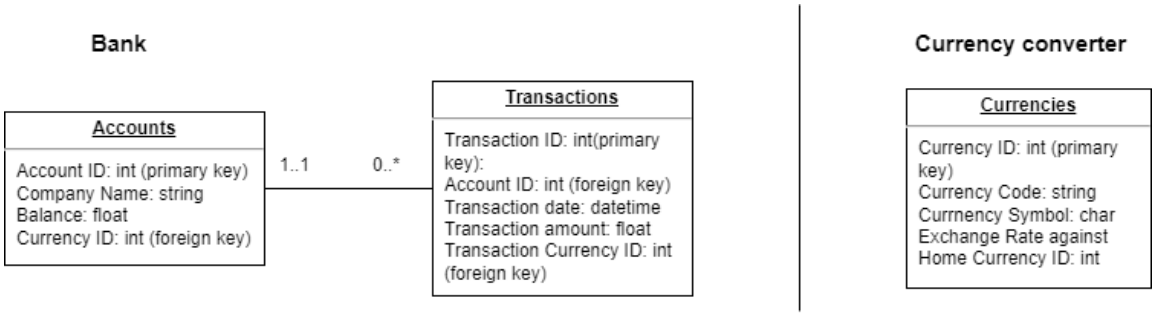

Figure 3.1.4 database schema for bank (left) and currency exchanger (right)

Additionally, the payment system will require two additional APIs, each maintaining its own database as shown in figure 3.1.4. To distribute the process of currency conversion, crucial for each working PSP, the currency converter database will contain a *Currencies* table of exchange rates against a chosen currency, in this case, the pound sterling. The Bank API on the other hand has two tables, one for storing the airline bank accounts and one for storing the transaction history.

3.2 - Airlines

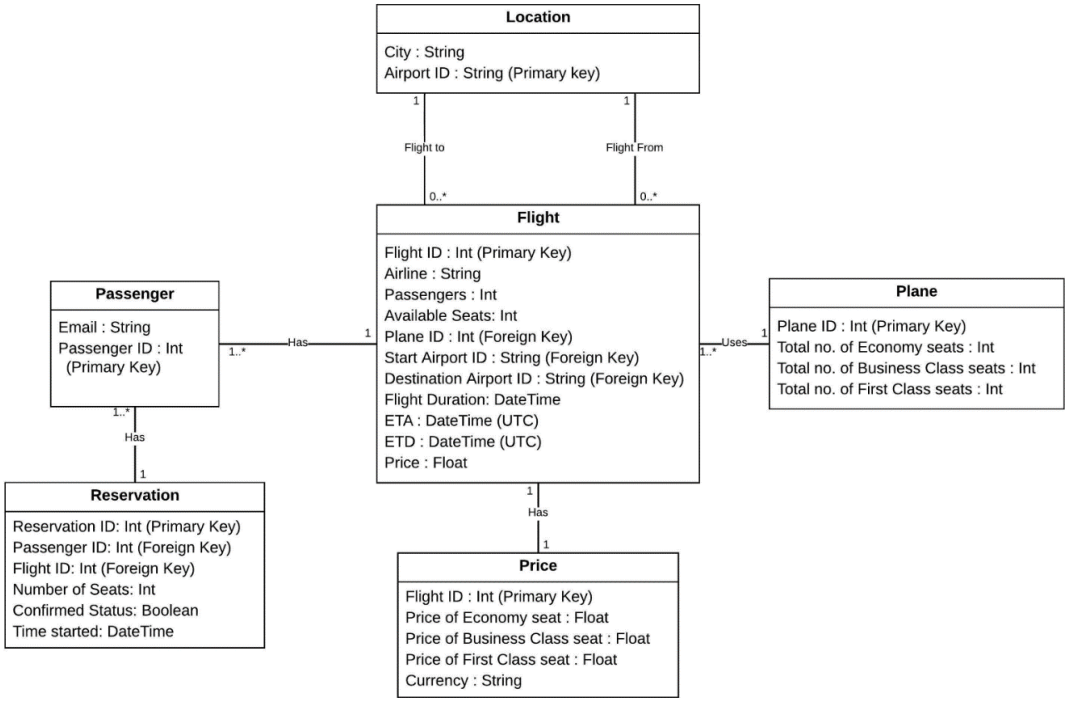

Figure 3.2.1: Database Schema for an Airline

The Airline API will handle all the information about the flights that the flight aggregator will then request for. Each class has a unique ID within it that distinguishes it from others of the same class for ease of access for the aggregator.

The 'Flight' class contains all the information about a passenger's flight including the price; the physical plane being flown; the start location and destination, the number of passengers that are set to fly, etc. This is a complete class that pulls information from the other classes.

A flight may have many passengers and, for each of these passengers, a reservation is held during the booking process. The reservation class ensures that a passenger may finish booking a flight without the seats being given up to another passenger during the booking process. It has a 'datetime' object within it that acts as the starting time of the reservation. There will be a time limit of 10 minutes to book the flight. If a flight is successfully booked, the reservation confirmed status will be true and it will be stored, else it will be false and the data from it will be purged.

Both the 'Plane' and 'Price' class contain the details for each type of seat within an aircraft: Economy, Business and First-class. When booking a seat on a plane, the customer may choose the class of seat they are purchasing which will then have a price for the seat dictated by the 'Price' class.

4. Endpoints

4.1 Payments:

The following API endpoints will be implemented for each payment API. Each API will support the same requests and methods, however, payloads and responses may differ based on the type of payment service API.

4.1.1 Request payment form fields

Endpoint URI: /payments/form:

Method: GET

Parameters/ Payload (in JSON): N/A

Response (in JSON): List of payment form fields:

- PayPal, Instalments: {'fields':{'Username': 'string', 'Password': 'string'}}
- Direct debit: {'fields':{'Card number': 'string', 'CVV': 'string', 'Expiry Date': 'date', 'Name': 'string', 'Email': 'string'}}

Description: API is requested a list of form fields required to authenticate payment. The request takes no argument and returns as a response a list of fields, specifying the type of response required for each field. For example, the response specifies that the expiry date field requires a date object as its input.

4.1.2 Make payment

Endpoint URI: /payments/pay

Method: POST

Parameters/ Payload (in JSON): User's payment details, transaction details.

- PayPal, Instalments: {'form':{'Username', 'Password'}, 'transaction':{'amount', 'currency', 'recipient account'}}
- Direct debit: {'form':{'Card number', 'CVV', 'Expiry Date', 'Name', 'Email'}, 'transaction':{'amount', 'currency', 'recipient account'}}

Response (in JSON): Payment status, Transaction ID, (for the instalment plan: details of future payments)

- PayPal, Direct debit: {‘status’: (success/failed),‘TransactionID’}
- Instalments: {‘status’: (success/failed), ‘TransactionID’, ‘payment Plan’: {‘instalments’: {‘today’: (amountDue),‘week2’: (amountDue),‘week4’: (amountDue),‘week6’: (amountDue)}}

Description: API is requested to make a payment from the user’s account to a specified recipient's account. The request takes the user’s credentials and the transaction details as its arguments. If the user credentials are verified, the API makes a POST request to the bank to make a transaction to the recipient's bank account. If the payment is successful, the API returns a ‘success’ status code, otherwise a ‘failed’ status code is returned instead. Additionally, the transaction ID is also returned. For the instalment plan payment service, details of the instalment plan are additionally returned.

4.1.3 Refund payment

Endpoint URI: /payments/refund

Method: POST

Parameters/ Payload (in JSON): User’s credentials, Transaction ID.

- PayPal, Instalments, : {‘form’:{‘Username’, ‘Password’}, ‘transactionID}
- Direct debit: {‘form’:{‘Card number’, ‘CVV’, ‘Expiry Date’, ’Name’}, ‘transactionID’}

Response (in JSON): Refund status. {‘status’: (success/failed)}

Description: The API receives a request to The API makes a refund request to the destination bank account from the payment with the transaction ID. User must be authenticated to make a refund request. For the instalments API, the user will only be refunded the amount they have paid in their instalments. Their payment plan will be terminated if unfinished.

4.2 Additional API’s supporting the payment system

Furthermore, the following API endpoints will be supported by the Bank and currency exchange API’s to facilitate the payment process.

4.2.1 Make transaction

Endpoint URI: /payments/pay

Method: POST

Parameters/ Payload (in JSON): transaction details. {‘transaction’:{‘amount’, ‘currency’, ‘recipient account’}}

Response (in JSON): Transaction status, Transaction ID {‘status’: (success/failed),‘TransactionID’}

Description: The Bank API is requested to make a transaction to the specified recipients account. The API increments the recipients balance by the transaction amount. The API then returns a status code to confirm whether the transaction was successful.

4.2.2 Refund transaction

Endpoint URI: /bank/refund

Method: POST

Parameters/ Payload (in JSON): Transaction ID. {‘transactionID’}

Response (in JSON): Refund status. {‘status’: (success/failed)}

Description: The Bank API receives a request to refund a transaction. The API queries its database of transactions and reverse the transaction matching the specified transaction ID. It then returns a status code to confirm whether the refund was successful.

4.2.3 Exchange currency

Endpoint URI: /currency_exchanger/exchange

Method: GET

Parameters/ Payload (in JSON): amount (required): the amount of money to exchange

from_currency (required): the currency to exchange from

to_currency (required): the currency to exchange to

Response (in JSON): Original Exchange amount, Original Exchange Currency, Exchanged Amount, exchanged currency {"original_amount": 100, "original_currency": "USD", "exchanged_amount": 83.58,"exchanged_currency": "EUR")

Description: This endpoint allows you to exchange one currency for another in your online payment application. The request should include the amount of money you wish to exchange, the currency you wish to exchange from, and the currency you wish to exchange to. The response will include the equivalent amount of money in the desired currency.

Other technical specifications needed for Payment Provider APIs:

- **Secure connection (HTTPS) when sending requests**
- **May need to send username and password in authorization header rather than POST payload for added security**
- Transactions required for payments – money should be taken from user's account and added to Airline bank accounts but only if both happen, i.e., one should not happen without the other

4.3 Airline API:

The API endpoints listed below will be supported by each airline in order to facilitate the flight booking process.

4.3.1 List all available flights

Endpoint URI: /airline/form:

Method: GET

Parameters/ Payload (in JSON): **List of flights**: {'FlightID', 'No. Available seats', 'Departure time', 'Arrival time', 'Price'}

Response (in JSON): {'Date of departure', City of departure', 'City of Arrival', 'Tickets' : *{*'No. economy', 'No. Business Class', 'No. First Class'*}}*

Description: Flight aggregator sends the details of the flight the customer wants and the number of ticket's to be booked. It then queries its database of flights to retrieve flights that match requested parameters. It returns this list of flights meeting the aggregator's requests.

4.3.2 Confirm Payment

Endpoint URI: /airline/confirm_payment:

Method: POST

Parameters/payload: {‘Reservation ID’}

Response: {‘Flight ID’, ‘List of seats’, ‘Passenger Information’, ‘Reservation ID’}

Description: Confirmation that a customer has paid for a flight. Returns booking details.

4.3.3 Reserve seat on flight

Endpoint URI: /airline/reserve:

Method: POST

Parameters/payload: {‘Flight ID’, ‘List of seats’, ‘Reservation ID’}.

Response: {‘Reservation ID’}

Description: Reserves seats until payment is processed successfully. The API adds a reservation object to its database. The call returns a reservation ID.

4.3.4 Cancel user’s reservation

Endpoint URI: /airline/cancel:

Method: POST

Parameters/payload: {‘Reservation ID’}

‘Response: {‘Status (success/fail) ‘Reservation ID’}

Description: Cancel reservation on request of the aggregator. Removes reservation object matching reservation ID from the database. Returns status code.

4.3.5 Confirm booking:

Endpoint URI: /airline/confirm_booking:

Method: POST

Parameter/payload: {‘Reservation ID’}

Response: {‘Status (success/fail)’, Booking ID’}

Description: Sends confirmation to the aggregator that a booking has been made. Adds booking object to the database. Returns status code.

4.3.4 Cancel user’s booking

Endpoint URI: /airline/cancel:

Method: POST

Parameters/payload: {Booking ID’, ‘Transaction ID’}

‘Response: {‘Status (success/fail), Booking ID’}

Description: Cancels a booking on request of the aggregator. Sends a request to payment API to reverse the transaction matching the transaction ID. Removes booking object matching booking ID from the database and returns a status code.

5. Bibliography

1. Editor, A.S. 2020. Flight booking process: Structure, steps, and Key Systems. *AltexSoft*. [Online]. Available from: https://www.altexsoft.com/blog/engineering/flight-booking-process-structure-steps-and-key-systems/.

2. Editor. Booking guidelines and tips. *British Airways*. [Online]. Available from: https://www.britishairways.com/travel/pop_faretips/public/en_gb.

3. Lufthansa n.d. Advance seat reservation | lufthansa. *lufthansa*. [Online]. Available from: https://www.lufthansa.com/ge/en/seat-reservation.

4. Elphick, D. 2023. What is a global distribution system (GDS)? *SiteMinder*. [Online]. Available from: https://www.siteminder.com/r/global-distribution-system/#:~:text=A%20Global%20Distribution%20System%20is,and%20other%20travel%20related%20services.

5. Skyscanner. *Compare Cheap Flights & Book Airline Tickets to Everywhere*. [Online]. Available from: https://www.skyscanner.net/.

6. Stevenson 2020. How do flight aggregators get airlines to participate in their website? *Medium*. [Online]. Available from: https://medium.com/@stevenson14789632/how-do-flight-aggregators-get-airlines-to-participate-in-their-website-c08ca4692f80.

7. Paradise, A.N. 2023. 16 flight booking sites to help you find the best deals (2023). *Nomad Paradise*. [Online]. Available from: https://nomadparadise.com/flight-booking-sites/.

8. Haus, R. 2019. The Pros and cons of booking with fare aggregators. *JTB Business Travel*. [Online]. Available from: https://jtbbusinesstravel.com/pros-cons-booking-fare-aggregators/.

9. CellPoint Digital 2022. The Airline Payment Guide for 2022: Blog. *CellPoint Digital*. [Online]. Available from: https://cellpointdigital.com/articles/blog/airline-payment-guide-2022.

10. Marketing 2022. Anatomy of a credit card: The luhn algorithm explained. *Ground Labs*. [Online]. Available from: https://www.groundlabs.com/blog/anatomy-of-a-credit-card/#:~:text=The%20account%20number%3A%20The%20number,based%20on%20the%20Luhn%20algorithm).

11. Raghavan, K. 2023. Apple pay vs. Google Wallet. *Investopedia*. [Online]. Available from: https://www.investopedia.com/articles/personal-finance/010215/apple-pay-vs-google-wallet-how-they-work.asp.

12. Tipalti 2023. Complete guide to what is PayPal and how does it work in 2023. *Tipalti*. [Online]. Available from: https://tipalti.com/en-uk/how-does-paypal-work/#:~:text=PayPal%20works%20as%20an%20intermediary,PayPal%2C%20instead%20of%20your%20bank.

13. Klarna and Klarna Klarna 2021. How does klarna really work? – klarna UK. – *Klarna UK*. [Online]. Available from: https://www.klarna.com/uk/blog/how-does-klarna-really-work/.

6. Appendix of minutes

Minutes of Group - Meeting 1

COMP3011 – Web Services and Web Data Group Project for Coursework 1

Date: 23.*02.23*

Start Time: *13:10*

End Time: *13:40*

Location: 2.05 Lab – William Bragg Building

Present:

1. *Kevin Braszkiewicz*
2. *Adam Zbikowski*
3. *Tamir Cohen*
4. *Maverick Low*
5. *Krzysztof Kamola*
6. *Safwan Chowdhury*
7. *Jordan Bushnell*
8. *Joseph McGowan*
9. *Canaan Weise*
10. *William Fleming*
11. *Olumide Akanbi*
12. *Felix Goldsmith*

Absent with apologies: *Jokubas Tolocka, Teck Ang*

Absent without Apologies: *anyone who has not turned up and not sent apologies in advance*

Chair: *Adam Zbikowski*

Minutes: *Kevin Braszkiewicz*

6.1.1 AGENDA

-Recording attendance and apologies

-Minutes of the last meeting *(read these together and approve or correct if not accurate)*

-Actions from last meeting *(read these actions and mark as "Complete" or "Outstanding")*

-Discussion on progress

-Risks and Issues *(identify and major risks to doing a good job and discuss solutions)*

-Actions arising *(record any actions agreed, who will do them and when they will be done)*

-Any Other Business *(discuss any other issues not covered above)*

-Date and Time of Next Meeting

6.1.2 REVIEW OF MINUTES OF THE MEETINGS

Not Applicable.

6.1.3 REVIEW OF ACTIONS FROM PREVIOUS MEETINGS

Actions carried over from previous meetings					
	Date	Action	Who	When	Status

6.1.4 DISCUSSION OF PROGRESS

Sorted group members into 3 teams. Groups:

Payment	Aggregator	Airline
Jokubas Adam Will Teck Ang Jordan Joe	*Tamir* Caanan Olumide Felix	Maverick Low Safwan Krzysztof Kevin

Discussion and understanding of project and its deliverables.

6.1.5 RISKS AND ISSUES

No risking arising currently.

6.1.6 ACTIONS ARISING

Actions arising from today's meeting					
	Date	Action	Who	When	Status
1	*23.02.23*	*Make shared word document*	*Adam*	*02.03.23*	*Ongoing*
2	*23.02.23*	Make teams group	Safwan	*02.03.23*	*Ongoing*
3	*23.02.23*	Set up Kanban board	Tamir	*02.03.23*	*Ongoing*
4	*23.02.23*	Research around team area	All	*02.03.23*	*Ongoing*
5	*23.02.23*	Think of name	All	*02.03.23*	*Ongoing*
6	*23.02.23*	Make minutes template	Kevin	*02.03.23*	*Ongoing*

6.1.7 Any Other Business (AOB)

N/A

6.1.8 Date and Time of Next Meeting

Date*: 02.03.23*

Time: *13:00*

Location: Lab 2.05 William Bragg Building.

END

Minutes of Group - Meeting 2

COMP3011 – Web Services and Web Data Group Project for Coursework 1

Date: 02.03.23

Start Time: *13:00*

End Time: 13:30

Location: 2.07 Robotics Lab – William Bragg Building

Present:

- Kevin Braszkiewicz
- Adam Zbikowski
- Tamir Cohen
- Maverick Low
- Krzysztof Kamola
- Safwan Chowdhury
- Jordan Bushnell
- Joseph McGowan
- Canaan Weise
- William Fleming
- Olumide Akanbi
- Felix Goldsmith

- Jokubas Tolocka,
- Teck Ang

Absent with apologies: *N/A*

Absent without Apologies: N/A

Chair: *Adam Zbikowski*

Minutes: *Kevin Braszkiewicz*

6.2.1 Review of Minutes of Last Meeting

No issues raised with the minutes created for 'Meeting 1'

6.2.2 Review of Actions from Previous Meetings

Actions carried over from previous meetings					
	Date	Action	Who	When	Status
1	02.03.23	*Make shared word document*	*Adam*	*02.03.23*	*Complete*
2	02.03.23	*Make Teams Group*	*Safwan*	*02.03.23*	*Complete*
3	02.03.23	*Set up Kanban Board*	*Tamir*	*02.03.23*	*Complete*
4	02.03.23	Think of name	*All*	*02.03.23*	*Ongoing*
5	02.03.23	*Make minutes template*	*Kevin*	*02.03.23*	*Complete*

6.2.3 Discussion of Progress

- We shared progress on research towards the first deliverable for the group report (Detailed Analysis of the business situation) for each respective group.

- Discussed some other systems that may need to be included in our architecture:

i) Storing bank accounts (banks)

ii)Currency converter

Iii) Flight detail database (GDS)

6.2.4 Risks and Issues

N/A

6.2.5 Actions Arising

Actions arising from today's meeting					
	Date	Action	Who	When	Status

1	*02.03.23*	*Consolidate research into bullet point format on shared document. Detailed Analysis of the business situation.*	*Each respective team.*	*09.03.23*	*Ongoing*
2	02.03.23	Think of name	*All*	*09.03.23*	*Ongoing*

6.2.6 Any Other Business (AOB)

N/A

6.2.7 Date and Time of Next Meeting

Date*: 09.03.23*

Time: 13:00

Location: William Henry Bragg Building

END

Minutes of Group - Meeting 3

COMP3011 – Web Services and Web Data Group Project for Coursework 1

Date: *09 03 2023*

Start Time: *13:30*

End Time: *14:30*

Location: *In Person - Robotics 2.07*

Present: *All Group Members*

Absent with apologies:

Absent without Apologies:

Chair: *Adam Z.*

Minutes: *Safwan C.*

6.3.1 AGENDA

1. Recording attendance and apologies
2. Minutes of the last meeting (*read these together and approve or correct if not accurate)*

3. Actions from last meeting *(read these actions and mark as "Complete" or "Outstanding")*

4. Discussion on progress

5. Risks and Issues *(identify and major risks to doing a good job and discuss solutions)*

6. Actions arising *(record any actions agreed, who will do them and when they will be done)*

7. Any Other Business *(discuss any other issues not covered above)*

8. Date and Time of Next Meeting

2. Review of Minutes of Last Meeting

Minutes approved/ revised

3. Review of Actions from Previous Meetings

Actions carried over from previous meetings					
	Date	Action	Who	When	Status
1	09.03	*Payment Database schematic made and presented with possible connection routes*	*Payment Team*		*complete*
2	09.03	Airline Databased Schematic made and completed	Airline Team		Complete
3	09.03	Aggregator Database Schematic still required, databases must be mapped by next meeting	Aggregator Team	16.03	In Progress
4	09.03				

4. Discussion of Progress

- Discussed implementation possibilities and how each team intends on building their own systems
- Use of API discussed
- Discussion of complexities with each area of development, such as depth of databases and services provided by each team

5. Risks and Issues

- *Risks of inconsistency and lack of coherence without constant and clear communication between each team*

6. Actions Arising

Actions arising from today's meeting					
	Date	Action	Who	When	Status
1	*09.03*	*Each team is required to produce 4 pages of report excluding the database schematic. This will be required to submit soon as time must be allocated for consolidation*	*All Teams*	*24.03*	*In Progress*
2	09.03	Each time must complete database schematics and diagrams by next meeting	All Teams	16.03	In Progress

7. Any Other Business (AOB)

N/A

8. Date and Time of Next Meeting

Date*: 16 03 2023*

Time: *13:00*

Location: Robotics Lab 2.07

END

Minutes of Group - Meeting 4

COMP3011 – Web Services and Web Data Group Project for Coursework 1

Date: *15 03 2023*

Start Time: *13:00*

End Time: *14:00*

Location: In Person - Robotics 2.07

Present:

• *Kevin Braszkiewicz*

• *Adam Zbikowski*

• *Tamir Cohen*

• *Maverick Low*

• *Krzysztof Kamola*

• *Safwan Chowdhury*

• *Jordan Bushnell*

• *Joseph McGowan*

• *Canaan Weise*

• *William Fleming*

• *Olumide Akanbi*

• *Felix Goldsmith*

• *Jokubas Tolocka*

• *Teck Ang*

Absent with apologies: *N/A*

Absent without Apologies: *N/A*

Chair: *Adam Zbikowski*

Minutes: *Jokubas Tolocka & Joseph McGowan*

6.4.1 AGENDA

1. Recording attendance and apologies
2. Minutes of the last meeting (*read these together and approve or correct if not accurate)*
3. Actions from last meeting *(read these actions and mark as "Complete" or "Outstanding")*
4. Discussion on progress
5. Risks and Issues *(identify and major risks to doing a good job and discuss solutions)*
6. Actions arising *(record any actions agreed, who will do them and when they will be done)*
7. Any Other Business *(discuss any other issues not covered above)*
8. Date and Time of Next Meeting

6.4.2 Review of Minutes of Last Meeting

Minutes approved/ revised

6.4.3 Review of Actions from Previous Meetings

Actions carried over from previous meetings					
	Date	Action	Who	When	Status
1		*Each team must produce 4 pages of report excluding database schematics*	*All*	*23/3/23*	*Outstanding*
2		*Each time must complete database schematics and diagrams by next meeting*	*All*	*16/3/23*	*Complete*

6.4.4 Discussion of Progress

1. Discussion of drawn schemas for each service

2. Discuss potential use cases with appropriate diagrams

3. Discussion of complexities with each area of development, such as depth of databases and services provided by each team

6.4.5 Risks and Issues

N/A

6.4.6 Actions Arising

Actions arising from today's meeting					
	Date	Action	Who	When	Status
1	*today*	*Finalise background research*	*person*	*23/3/23*	*Outstanding*
2	" "	Describe database schemas work / interactions between systems	Teams	" "	Outstanding
3	" "	Draw up use case diagrams	1 person from each team	" "	Outstanding
3	" "	Draw up component diagram	" "	" "	Outstanding

6.4.7 Any Other Business (AOB)

N/A

6.4.8 Date and Time of Next Meeting

Date*: 23 03 2023*

Time: 13*:00*

Location: In Person - Robotics 2.07

END

Minutes of Group - Meeting 5

COMP3011 – Web Services and Web Data Group Project for Coursework 1

Date: *23 03 2023*

Start Time: *13:15*

End Time: *14:00*

Location: *In Person - Robotics 2.07*

Present:

• *Kevin Braszkiewicz*

• *Adam Zbikowski*

• *Tamir Cohen*

• *Maverick Low*

• *Krzysztof Kamola*

• *Safwan Chowdhury*

• *Jordan Bushnell*

• *Joseph McGowan*

• *Canaan Weise*

• *William Fleming*

• *Olumide Akanbi*

• *Felix Goldsmith*

• *Jokubas Tolocka,*

• *Teck Ang*

Absent with apologies: *N/A*

Absent without Apologies: *N/A*

Chair: *Adam Zbikowski*

Minutes: *Joseph McGowan*

6.5.1 AGENDA

1. Recording attendance and apologies
2. Minutes of the last meeting (*read these together and approve or correct if not accurate)*
3. Actions from last meeting *(read these actions and mark as "Complete" or "Outstanding")*
4. Discussion on progress
5. Risks and Issues *(identify and major risks to doing a good job and discuss solutions)*
6. Actions arising *(record any actions agreed, who will do them and when they will be done)*
7. Any Other Business *(discuss any other issues not covered above)*
8. Date and Time of Next Meeting

6.5.2 Review of Minutes of Last Meeting

Minutes approved/ revised

6.5.3 Review of Actions from Previous Meetings

Actions carried over from previous meetings					
	Date	Action	Who	When	Status
1	16/3/23	Finalise background research	All teams	23/3/23	Complete
2	" "	Describe database schemas work / interactions between systems	All teams	" "	Complete
3	" "	Draw up use case diagrams	1 person from each team	" "	Complete
4	" "	Draw up component diagram	" "	" "	Complete
5	" "	Each team must produce 4 pages of report excluding database schematics	All	" "	Complete

6.5.4 Discussion of Progress

1. Group interview booked for Thursday
2. Discussed links between APIs

3. Discussed seat booking process

6.5.5 Risks and Issues

N/A

6.5.6 Actions Arising

Actions arising from today's meeting					
	Date	Action	Who	When	Status
1	23/3/23	Finalise group report	All Teams		

6.5.7 Any Other Business (AOB)

N/A

6.5.8 Date and Time of Next Meeting

Date*:* -

Time: -

Location: -

END