UNIVERSITY OF LEEDS

School of Computing

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

COMP3011 Web Services and Web Data

Coursework 1

Ahmed Elkafraw,

Amy Porter,

Damien Ruban,

Edward Day,

Emily Kerkhof,

Harry Davies,

Harry Owens,

Luke McDowell,

Luke Wood,

Michael Wright,

Samuel Poirier,

Scott Owen James,

Shamita Datta

UNIVERSITY OF LEEDS

Contents

Business Situation Analysis

When designing a web-based flight aggregator for booking and searching for flights, it is important to notice that there are three main components:

1. The design of the flight aggregator itself
2. The flight options from various airlines that are provided to the user by the flight aggregator
3. The payment service that is used to pay for the flight and confirm the booking.

Our group was divided into these three separate teams and decided to design 5 flight aggregators, 5 different airlines and 3 payment services.

To competently design a good flight system, we began by investigating current flight aggregators and seeing how they work. One very popular and well-known flight aggregator is Skyscanner (Skyscanner, 2023), which is a flight aggregator used by many to search and book for flights all over the world for the best price. For our investigation, our team outlined the process flow observed from the Skyscanner system.

Skyscanner Process Flow

Flight aggregator

1. A user enters details into Skyscanner to query for flights:
    - Departure airport
    - Arrival airport
    - Departure date
    - Return date
    - Cabin class (economy, premium economy, business class, first class)
    - Number of passengers (adults and children)
2. Skyscanner returns deals from various airlines and holiday companies. The user can also give more specifications to ensure the flights are catered for their requirements:
    - Departure time
    - Arrival time
    - Number of stops (direct, 1+ stop, 2+ stops, etc.)
    - Airlines (easyJet (easyJet, n.d.), British Airways (British Airways, n.d.), etc.)
    - Travel duration
3. The user selects the flight combination that is most appropriate for them.
4. Skyscanner lists the different prices from various airlines (e.g. easyJet (easyJet, n.d.), British Airways (British Airways, n.d.)).
5. The user selects which deal they prefer.

Airline

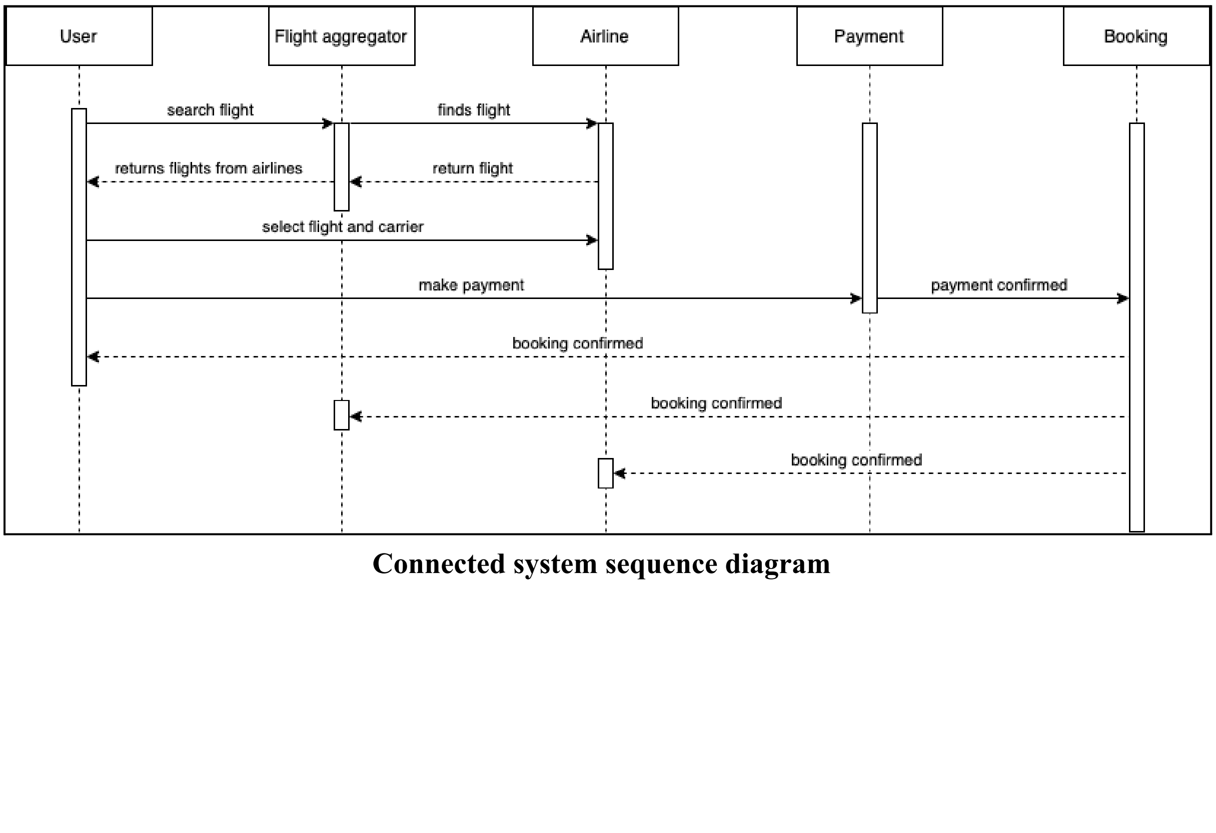
1. Skyscanner sends the user over to the airline's API to confirm the flight details.
2. The airline requests booking details and any additional add-ons from the user:
    - Passenger details (name, address, email, etc.)
    - Luggage requirements
    - Seating
    - Car rental

    - Travel Insurance
3. To complete the booking, the user has to either register as a new customer on the site or login as an existing customer.
4. The airline provides the user with a summary of their booking.
5. The user confirms their booking and has to select their payment method.

Payment service

1. The user is taken to the payment service website, where they enter their details:
    - Name
    - Card number
    - Expiration date
    - CSV number
2. The payment service confirms that the details are correct and the payment is confirmed.
3. Once the payment is confirmed, a confirmation is returned to the airline and flight aggregator confirming that those seats are booked to that flight and that the transaction was a valid one.
4. The user receives a payment confirmation and a booking confirmation.

Our research showed that there are in fact many steps in the booking process, and linking all of the different services together could prove to be quite complicated, so we had to make sure that our designs were accurate and well linked together. We created a sequence diagram so that we could start designing the interactions of each system:



Connected system sequence diagram

Feature Analysis

Once the process flow of our competitors had been determined, we completed a feature analysis to fully understand what functionality each service provides, to assist in creating our own system requirements.

Service	**Feature**	**Description**	**Competitor**
Flight Aggregator	Searches for flights	User can input data, such as destinations and dates, to receive a list of applicable flights	Skyscanner, Google Flights
Flight Aggregator	Sorts flights	The flights returned are sorted based on price or flight duration	Skyscanner
Flight Aggregator	Filters flights	The flights suggested can be further filtered by the price, flight time, or destination	Skyscanner, Google Flights
Flight Aggregator	Flight availability	Flights have seating availability matching the user's requests sold out flights are omitted	Skyscanner
Flight Aggregator	Select flights	User can select the chosen flight from the list returned	Skyscanner, Google Flights
Flight Aggregator	Connecting flights	Is able to suggest combining several connecting flights to the chosen destination	Skyscanner, Google Flights
Airline	Flight information	Provides detailed information on the flight, including the airline, price, and travel times	EasyJet, Ryan Air
Airline	Passenger Details	Takes details from the passenger such as their name, email, and passport number	EasyJet, Ryan Air
Airline	Bookings	Can take bookings on flights, as well as edit and remove passengers	EasyJet, Ryan Air
Airline	Seat Selection	Allows users to select preferred seats on the flight from the available options	EasyJet, Ryan Air
Payment	Card Details	Allows users to enter card details	Paypal, Klarna
Payment	Bank Selection	A range of banks are accepted and can be used through the payment provider	Paypal, Klarna
Payment	Process Payments	Securely processes payment requests between bank accounts	Paypal, Klarna
Payment	Validate funds	Reject the transaction if the payment account does not have sufficient funds, and alert the user, so they can change payment methods	Paypal, Klarna

<u>Working Principle Description</u>

Functional Requirements

Having completed our market research on our competitors, we were able to start building out our own high-level system requirements using the provided stakeholder specification (mark scheme) and the feature analysis above. Our proposed solution has the following system requirements:

I. Search - Users can supply search queries, with information such as the dates of travel, number of passengers, the arrival and destination airports.
II. Sort - The flights returned to the user are organised and sorted based upon the flight price and duration, to help users find the most suitable flight.
III. Suggestions - If no applicable flights are found, alternate dates / destinations are suggested
IV. Booking - Users can use the aggregator to select flights and book their seats.
V. Payment - The user can select a payment method and use it to purchase their tickets.
VI. Tickets - The user's tickets, flight details, and a confirmation will be displayed.
VII. Receipt - A record of the payment transaction will also be sent to the user.
VIII. Cancel Bookings - Using the booking information, users can cancel their bookings.

The flight aggregator, and subsequent airline and payment providers, will be considered successful if they meet the overhead criteria.

Persona User Stories

To create a deeper understanding of the connected system and gain further insight into its behaviour, we created user stories (Lucassen et al., 2016) across a broad range of customer types. This user-centred design approach ensures development is driven by the needs of users, while still meeting the business requirements. Each persona below demonstrates the different preferences and priorities of customers and highlights a system requirement. Several additional in-depth user stories can also be found under 'Detailed User Stories" in the Appendix, which were especially useful for team members to empathise with the perspective of possible system users and align with their potential motivations and needs.

Persona: Full Time Employee
I want to: <u>Choose specific dates</u> for my flights
So that: I can use my available days off to travel

Persona: Business Traveller
I want to: Book a flight to a work conference in another city and <u>receive billing details</u>
So that: I can submit the receipt for my expenses and be reimbursed by my company

Persona: Single Parent with Young Children
I want to: Book a family holiday to Spain and <u>select what seats we are in</u>
So that: I can sit with my children and look after them

Persona: Large Family of Six
I want to: Browse flights with six seats and receive <u>suggestions on alternative dates at lower costs</u>

So that: I can book an affordable holiday

Persona: Honeymoon Couple
I want to: Choose a specific destination airport, such as Paris (CDG)
So that: We can have our dream romantic holiday

Persona: Non-Driving Traveller
I want to: Look at what flights are available from my local airport
So that: I can travel to and from the airport as quickly, easily, and cheaply as possible

Persona: Young Travel Enthusiast
I want to: Sort flight prices by low to high
So that: I can quickly find the cheapest flight options

Persona: Student Traveller
I want to: Have many options for payment methods
So that: I can defer my payment by using a credit card or splitting it into several instalments

Persona: Lottery Winner
I want to: Book expensive first class seats
So that: I can have a luxurious travelling experience

Persona: Independent Child Traveller
I want to: Know my flight details, including flight number, departure, and arrival times
So that: I can share the details with my family and meet them in the airport

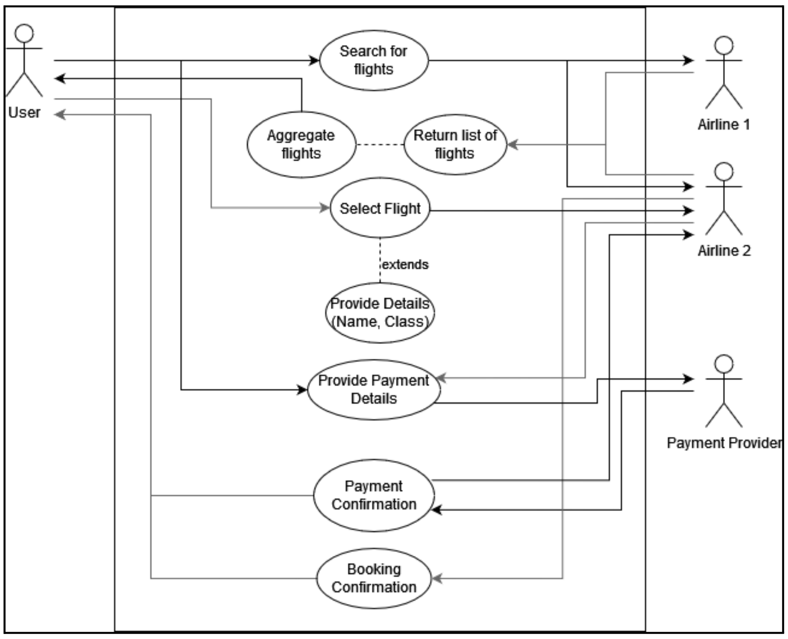
Persona: Cancelled Romantic Holiday
I want to: Cancel our flights
So that: We do not have to go on holiday and can receive a refund

Relevant System Actors

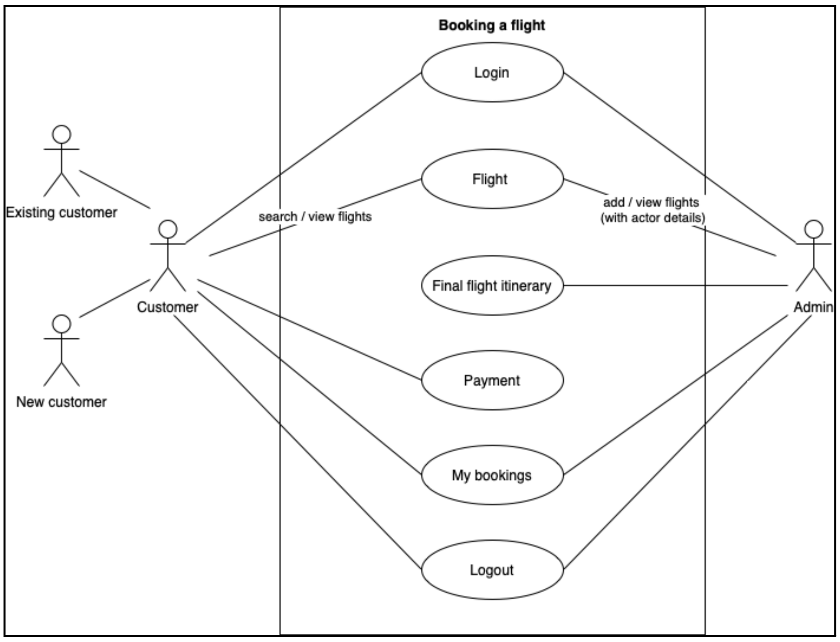
- **Customer users:**
  - The customers who access the connected system through the aggregator to search for flights, compare prices between airlines and book seats and plane tickets.
- **Flight aggregation service providers**:
  - The companies who collect airline information and provide a platform to customers to query and compare prices and schedules for flights.
- **Airline providers**:
  - The companies that operate the running flights and provide information on flights to the aggregation services.
- **Payment Gateways providers**:
  - The companies that provide the functionality for the secure processing of payments for online monetary transactions.
- **Governments and other regulatory bodies**:
  - The organisations that provide regulation for the aviation and payment industries and impose regulations to ensure compliance with safety, privacy and security standards.

Use Case Diagrams

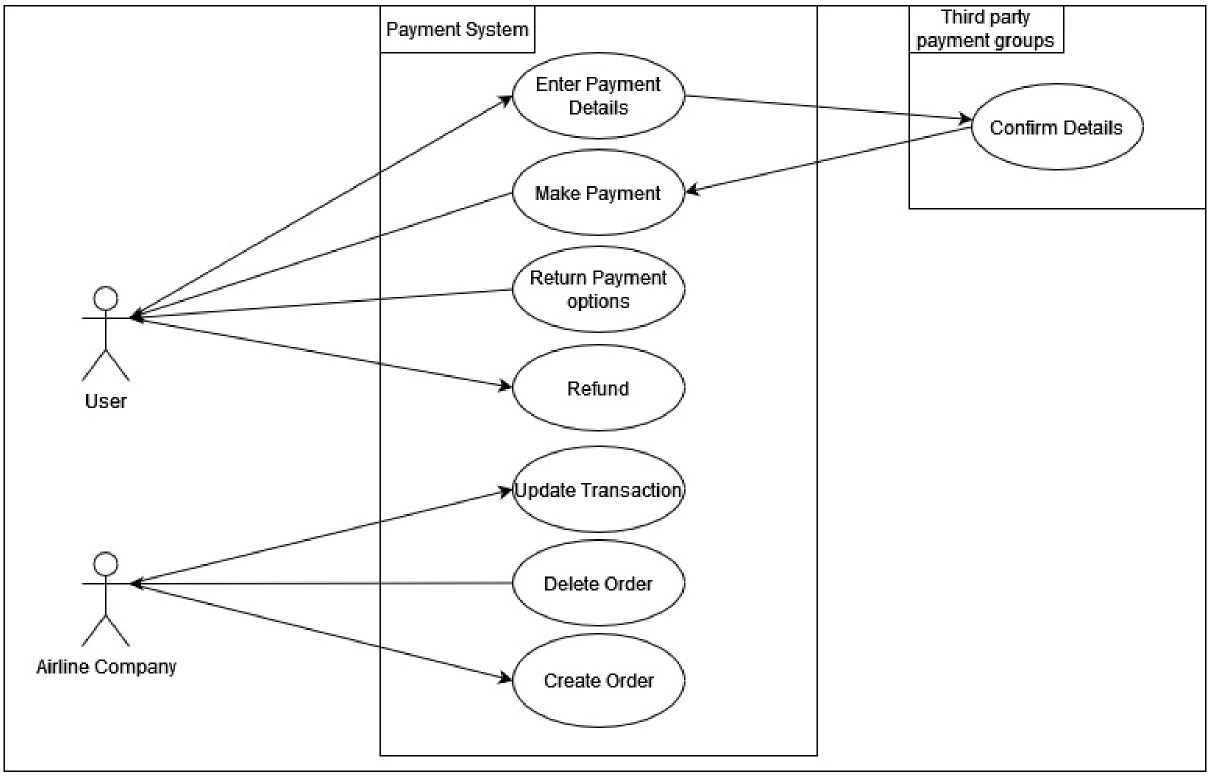
With the relevant system actors identified, UML use case diagrams can be created to visualise the proposed functional requirements of the system and represent the system's scenarios, goals and scope (Lucidchart, 2023). As well as aiding in the understanding of requirements and supporting system design, use case diagrams are also useful in identifying potential errors in the proposed solution, and for communicating the functional requirements to any possible stakeholders.



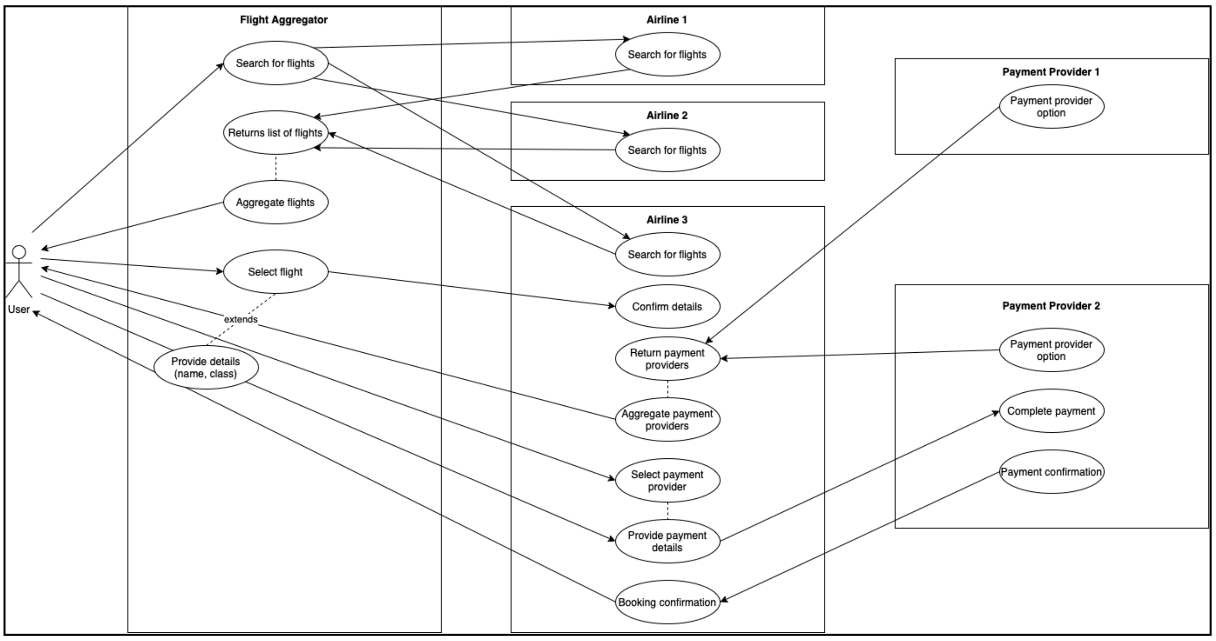
Use case diagram showing the interactions with the flight aggregator and the relevant system actors.



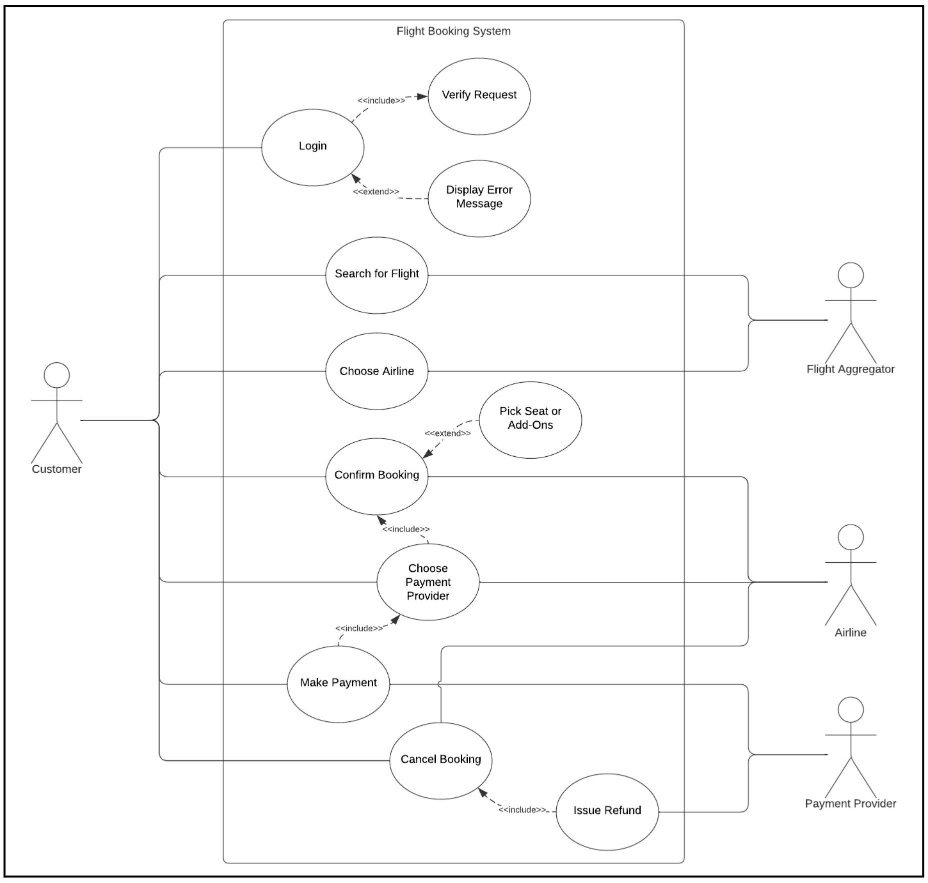
Use case diagram showing interactions between a user and an airline.



Use case diagram showing interactions between a user and payment service



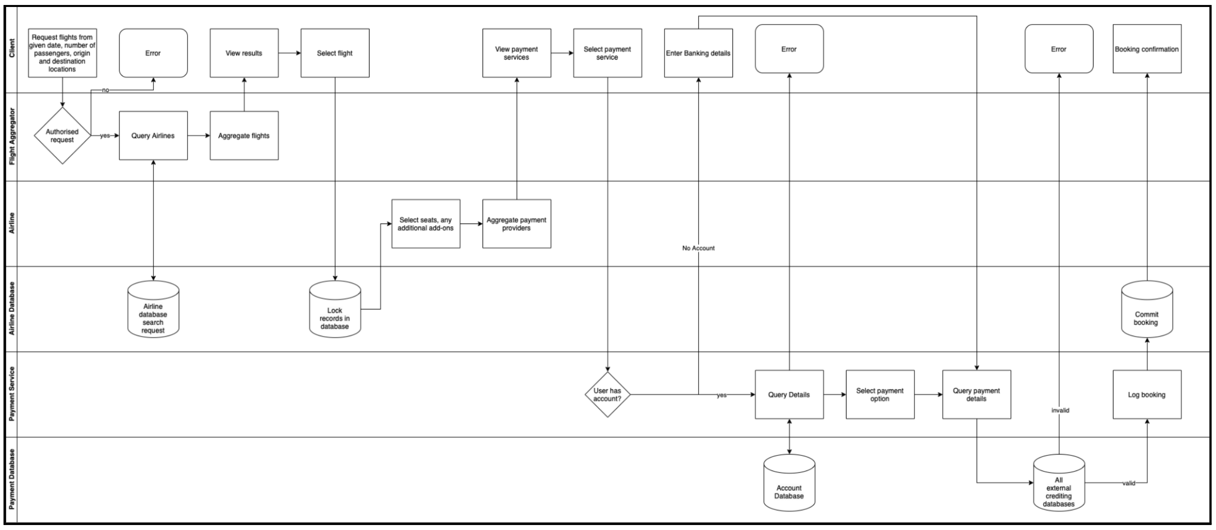
Use case diagram showing the interaction between users, flight aggregator, airlines and payment providers



Simplified UML Use Case Diagram for the entire process, visualising how all systems and stakeholders interact with the app.

System Architecture

Following our capturing and defining of requirements, as well as the completion of various use cases and system actor stories, work can be commenced with regards to practical implementation of the system. This section will define the overall proposed system architecture and how the different components of the system will interact with each other to fulfil the requirements captured in the use case diagrams.



Overall proposed system architecture diagram

(Note: Higher resolution version available under 'Overall Proposed System Architecture Diagram' in the Appendix)

Flight aggregator Architecture

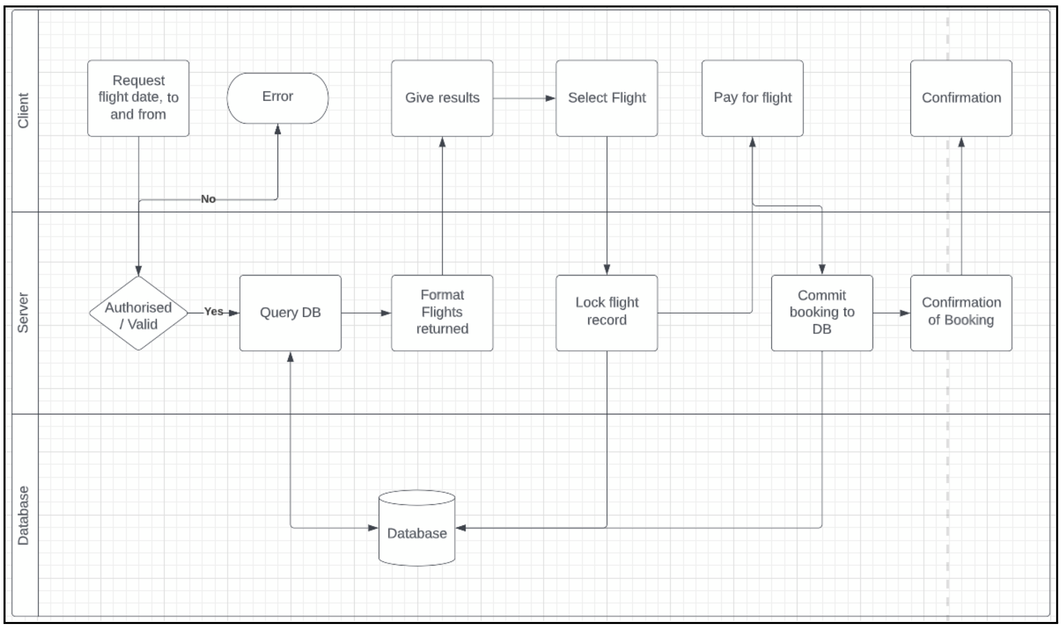
The flight aggregator services will be designed and implemented to act as the client in the system. They should provide a platform that allows users to search and compare airfares from various airlines, then book selected flights and seats. The system should collect flight data from a variety of airline services, which can then be processed and queried by the end-users. The infrastructure to enable this functionality will include a server/servers to run the service on and a relational database to store log data such as user queries.

The following steps outline the proposed flow of requests between APIs for the airline aggregator service:

1. The user submits a flight search query into the aggregator's command-line interface which is received by the system.
2. From the search query received, requests are sent to the airline APIs to retrieve information about flights that match the criteria specified by the user.
3. The search results are displayed to the user and the user can select a flight from those displayed.
4. Once the user has selected their flights, booking and payment are initiated through communication with the airline's seat reservation users and the chosen payment service's APIs.
5. Assuming the booking and payment processes are successful, confirmation and flight details are shown and sent to the user.

Airline Provider Architecture

When designing our five airlines, it was essential to make the overall design and structure of each airline the same to allow the aggregator to easily interface with them all. Each airline would receive flight requirements and details from the flight aggregator and would search the database for flights with these specific details. The database would include the flights with each record having the departure airport, arrival airport, departure date/time, arrival date/time, and the number of seats available on each flight. The following flowchart shows the requests between the client, server and database for each query.



Airline Request Flowchart

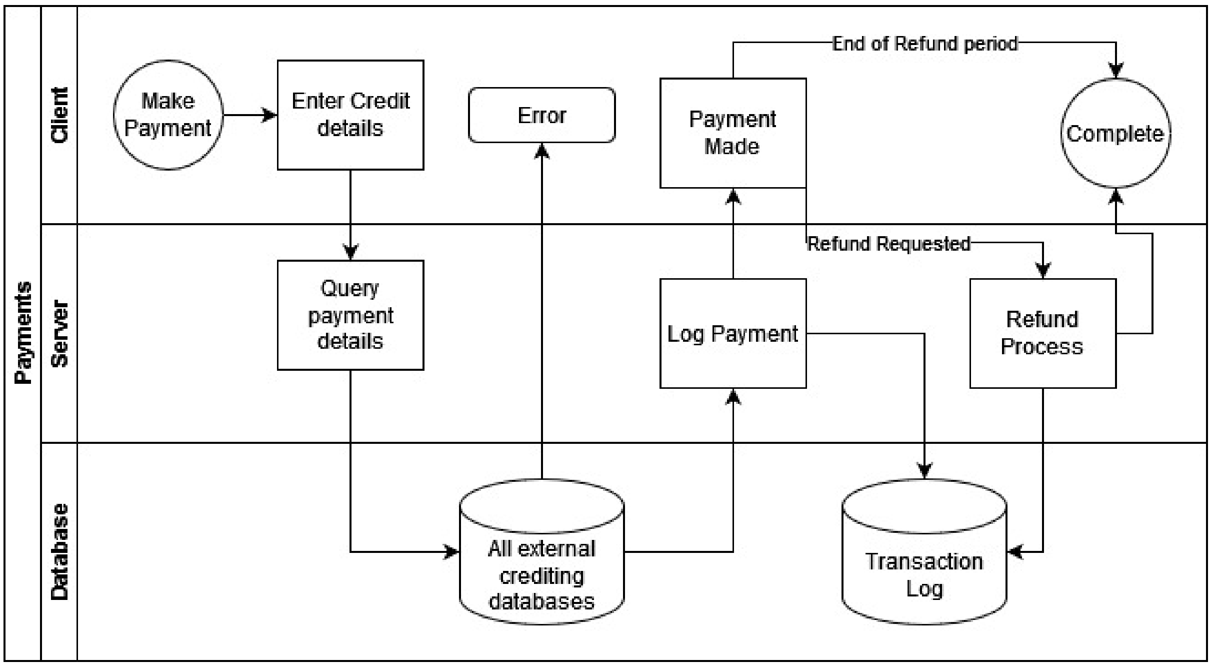
The following steps outline the proposed flow of requests between APIs from the aggregator to the airline service:

1. The query comes in from the aggregator, and the DBMS is used to query with the request parameters.
2. Once the flight details are returned, they are sent back to the aggregator for the user to pick which flights they would like to book.
3. Once a request comes in to book the flights, lock the seats and the transaction for a certain time.
4. Once the confirmation comes in for the booking, commit details to the database and return the confirmation of booking to the aggregator.

The system crucially relies upon the database of flights, which is explored in greater detail further on.

Payment Service Architecture

Each payment service provider would provide the user with an array of options on how to pay for their flights; from these options they can select one and enter their relevant credit card details. If accepted, the service provider would log the payment credentials and method used, and return a message stating confirmation.



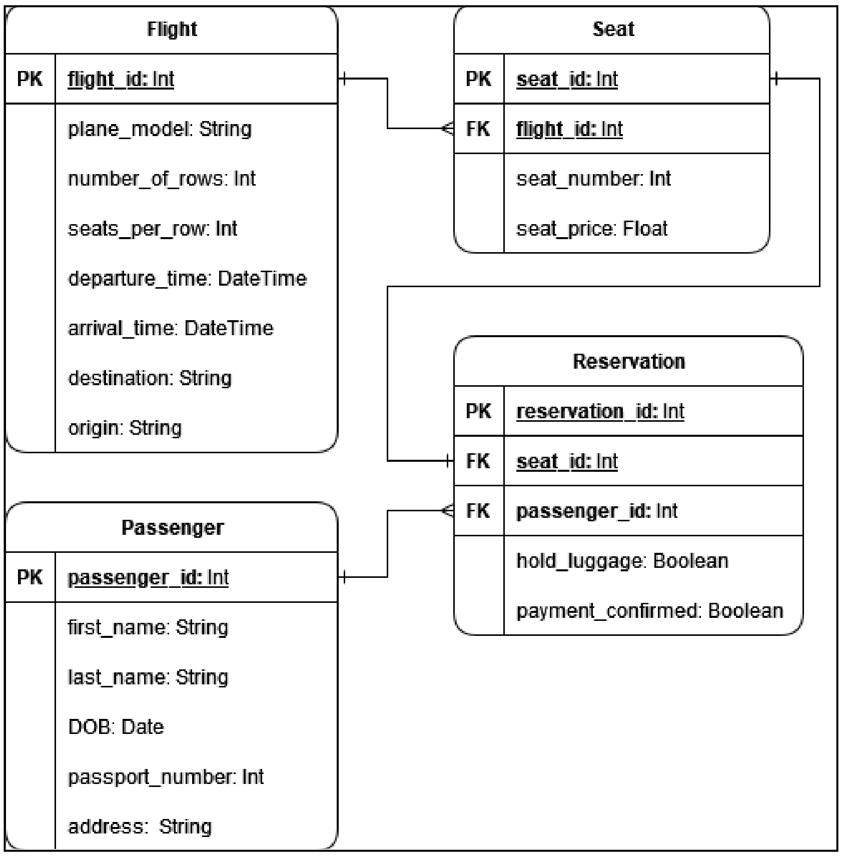
Payment Service Flow

1. The query comes in from the user, relating to basic information on transactions.
2. Once the details are verified, a user can select a payment method then enter their payment details.
3. The server queries the payment details; if they return verified for that payment option it can proceed with the transaction, if not then it returns an error to the user.
4. Waiting period for any refunds or updates that need to be made to the transaction, will either pass the waiting period or be refunded/updated, may be updated multiple times.

Database Design

Airline Database

For the database associated with the airline, four tables have been designed: Flight, Seat, Reservation and Passenger. The following entity-relationship diagram shows the structure and relations between these tables in the proposed Airline API databases.



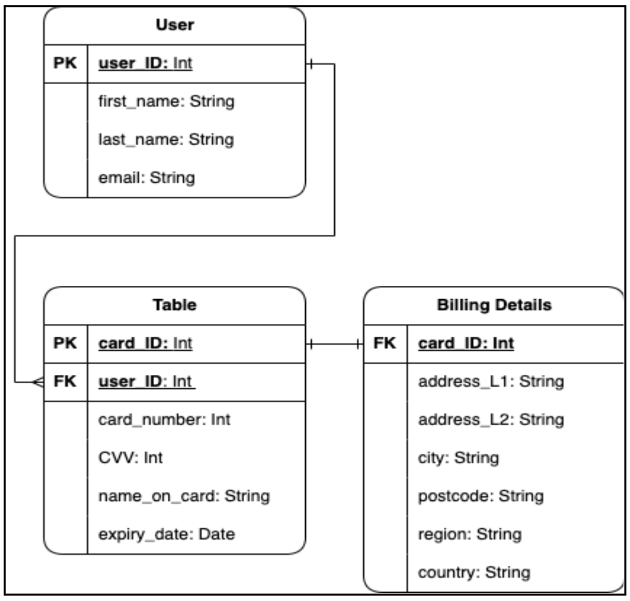
Airline Services Entity-Relationship Diagram

The flight table will contain all of the available flights that a specific airline provider offers, this will include the time of departure and arrival, the location of origin and destination as well as the number of seats on the plane. To find the available seat you will need to query the seat table which will store the booked seats for all flights with the airline provider. All of the reservations will be stored in the reservation table, this will contain foreign keys that point to all of the relevant information for the passenger and the seat that they have booked. One reservation can only store one seat as shown by the entity relationships on the diagram.

This database has been designed with efficiency, scalability and simplicity in mind. The database is very intuitive and should make sense to developers who are implementing the model; this is apparent with the naming of the database fields, as well as the table names which relate directly to the real world data that they represent. Also, due to the setup and layout of the entity relationships, the database will scale efficiently and performance should be fast.

Payment Service Database

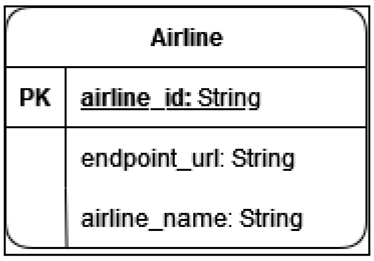
For the payment services design, a database with three relations has been proposed, namely ‘User’, ‘Card’ and ‘Billing Details’. The following diagram shows the entities and relationships for the suggested design, including that one user can have many cards associated with their account and each card has one billing address associated with it.



Payment Service Entity-Relationship Diagram

Flight Aggregator Database

Our proposed database for the flight aggregator system consists of one simple table for keeping track of airline endpoints.
The Airline table has three fields: ‘endpoint_url’ (a URL represented as a string), ‘airline_name’ (a string) and 'airline_id'. The 'airline_id' column serves as the primary key for the table, ensuring that each airline is uniquely identified within the system. The ‘endpoint_url’ column stores the endpoint URL for each airline, which is used by the system to communicate with the airline's API.



Airline Services Entity-Relationship Diagram

Client & API Specification

Payment Service API

https://app.swaggerhub.com/apis/EDWARD57DAY/PaymentSystem/1

Overview

After the details of the order have been taken from the booking system, a new Order object will be created and stored in the database, containing an OrderItem object for each item in the order.
When the user then proceeds to make a payment, a new Transaction object will be initialised, referencing the Order object that the transaction is to pay for and also containing the additional details of the transaction. Initially, this will be created with a status of “new”. When a user then makes a payment using a payment provider, this will reference the transaction ID, and if the payment is successful then the status of the transaction will be updated to “paid”. From here, the callback link associated with the transaction will be called which will complete the booking and notify the user. The full specification for the payments API is outlined within the OpenAPI specification above. The key functionality is as follow:

Create Order
When a customer places an order from the aggregator, this function is called to create an order object which holds the details of each of the items contained within the order in the database.

Create Transaction
Once an order has been created, a transaction can then be initialised which links an order to a user and contains any other details that will be needed to complete the order, such as the delivery address. The transaction is initially created with a status of “New”, and as the transaction progresses, this status will be updated accordingly.

Make Payment
This endpoint is used to take the payment from the customer and complete the transaction, updating the order status to completion if successful. This will then trigger the transaction’s callback URL to be called which will trigger any other operations necessary for a completed order, such as sending the booking confirmation to the user.

Update Transaction
This allows any of the details of the transaction to be updated if necessary.

Refund Transaction
Process a refund to the customer’s payment method and change the order status to “Refunded” if successful.

Delete Order/Transaction
Remove an incomplete order/transaction from the database.

Airline

https://app.swaggerhub.com/apis/LUKEMCDOWELL2014/FlightsAPI/1.0.0

Overview

The database can be searched for flights by specifying departure and arrival locations and the day of travel. Once a flight is selected, a reservation is created in the database, which includes the flight and passenger information. This reservation can be viewed, updated, or deleted anytime. A reservation is created as soon as the passengers personal details have been entered, but payment can be made at a later stage, in which case the reservation is updated to reflect the payment. The full specification for the airline API is outlined within the OpenAPI specification above. The key functionality is as follow:

Search for Flights

This endpoint will return all flights from a specified departure location to an arrival location on a specified date.

Create Reservation

Once a flight has been selected, a reservation can be created that will contain the flight information, seat information and customer details. Payment is not taken at this stage.

Find Reservation

This returns all of the information about a specified reservation.

Update Reservation

This allows any details of a reservation to be updated if necessary.

Delete Reservation

Removes a reservation from the database.

Confirm Reservation Payment

Once a payment has been processed, this endpoint can be called to then mark the reservation as paid.

Aggregator

As discussed previously as part of system architecture, the flight aggregator service acts as the client in the proposed system, hence it has no API endpoints of its own. Instead, the flight aggregator service communicates with the other services in the system through their respective APIs. This approach helps to ensure loose coupling between the different components of the system, making it easier to update or replace individual services without affecting the overall system (Gorton, 2016). Moreover, it allows our flight aggregator services to be lightweight and focused on core responsibilities, rather than attempting to handle all aspects of the system's functionality. Through careful design of the APIs and the use of industry-standard communication protocols, the various services can work together seamlessly to deliver a high-quality experience for users of the system.

References

- *Skyscanner* (2023) *Compare Cheap Flights & Book Airline Tickets to Everywhere.* Skyscanner Ltd. Available at: https://www.skyscanner.com/ (Accessed: March 7, 2023).
- Lucassen, G., Dalpiaz, F., Werf, J.M.E.M.v.d., Brinkkemper, S. (2016). *The Use and Effectiveness of User Stories in Practice*. In: Daneva, M., Pastor, O. (eds) *Requirements Engineering: Foundation for Software Quality*. REFSQ 2016. Lecture Notes in Computer Science(), vol 9619.
- *Lucidchart* (2023) *UML use case diagram tutorial- Why use a UML diagram?*. Available at: https://www.lucidchart.com/pages/uml-use-case-diagram (Accessed: March 13, 2023).
- *British Airways.* n.d. "*Book Flights, Holidays, City Breaks & Check In Online.*" British Airways | Book Flights, Holidays, City Breaks & Check In Online. Available at: https://www.britishairways.com/travel/home/public/en_gb/ (Accessed: March 25, 2023).
- *easyJet*. n.d. "*easyJet*." easyJet | Cheap flights ✈️ Book low-cost flight tickets 2023. Available at: https://www.easyjet.com/en/ (Accessed March 25, 2023).
- *Gorton, I.* (2016). Essential Software Architecture (2nd ed.). Springer International Publishing. p. 18.

<u>Appendix</u>

Meeting Minutes

Meeting 1

Date: 23/02/2023 **Time:** 14:00-15:00

Minutes taken by: Amy Porter

Attendees: Amy Porter, Shamita Datta, Ahmed Elkafrawy, Samuel Poirier, Harry Davies, Emily Kerkhof, Damien Ruban, Edward Day, Harry Owens, Luke Wood, Luke McDowell, Michael Wright and Scott James **(13/13 present)**

Agenda:

- Group 11 Team Introductions
- Run through the coursework specification as a team
- Team name
- Discuss group divisions
- Address and assign first tasks to be completed

Notes:

- Each member introduced themselves to the wider team
- Conducted brief overview of specification document
- Team name discussion:
    1. Decided on a variation of *'REST Easy'*
- Group divisions:
    1. Discussion: 5/5/3 split for groups may be most effective considering group structure and the required tasks
    2. Business alignments for each member:
        - Airline Companies (such as British Airways or Turkish Airlines)
            - Shamita, Damien, Harry O, Luke M, Michael
        - Flight aggregator developers (such as Skyscanner)
            - Sam, Harry D, Luke W, Emily, Amy
        - Payment Providers (such as PayPal or Klarna)
            - Ed, Ahmed, Scott
- Decided that the second assessment task can be started as part of the first task
    1. *Examine the websites of various airlines, and carefully analyse the processes of finding and booking suitable flights*
    2. *Document the sequence of the above processes from start to finish and use appropriate representation methods such as flow charts and use-case diagrams. Also, as part of this activity identify all relevant actors (e.g. servers, human users, client software, etc) involved in the above processes, the role of each actor in a process, and the nature of the data*
- Every member to begin research and analysis, and begin drafting/noting relevant flows and actors discovered as part of this investigation, focusing on their assigned business alignment.

- Discussed the differences in the flows from a user perspective vs the flow of data
- Refined sequence documentation can be addressed after meeting 2

Tasks for next Meeting:

- Assessment tasks one and two
    - Initiate research and analysis of existing services
    - Create notes in preparation for task 2

Meeting 2

Date: 02/03/2023 **Time:** 13:00-14:00

Minutes taken by: Amy Porter

Attendees: Amy Porter, Shamita Datta, Ahmed Elkafrawy, Samuel Poirier, Harry Davies, Emily Kerkhof, Damien Ruban, Edward Day, Harry Owens, Luke Wood, Luke McDowell, Michael Wright and Scott James **(13/13 present)**

Agenda:

- Present research insights from each subgroup to wider team
- Align on documentation approach for research findings
- Move onto next assessment tasks appropriately

Notes:

- Presentation of research insights from each subgroup to wider team
    - Flight aggregator summarised flow
    - Payment providers eg Klarna, Paypal, Stripe looked at
        - Discussed issue of bank verification → "dummy verification"
    - Airline providers:
        - discussed data flow and API interface
- Discussed and aligned on what documentation needs to be provided for each group's research
- Moving onto task 3 - database models
    3. *Design and document appropriate and efficient database models for each web service in the system, i.e. for airlines, payment providers, and any additional supporting services (if needed).*
- Any additional supporting services?
    - At this stage we cannot foresee any required additional services to support the connection of the payment providers, airlines and flight aggregators.

Tasks for Next Meeting:

- schedule and book meeting room for next week
- each subgroup to add their sequence of processing notes to the google doc for the report:
    - **summary** of research/findings

  - data sequence **flowchart**
    - API endpoints
  - **use case diagram**
  - list of **relevant actors** and their role
  - data dictionary→ **type of data exchanged**
    - include considerations for database model
  - **user story / stories**
- Come to the next meeting ready to design and document appropriate and efficient data models

Meeting 3

Date: 09/03/2023 **Time:** 13:00-14:00

Minutes taken by: Luke Wood

Attendees: Amy Porter, Shamita Datta, Ahmed Elkafrawy, Samuel Poirier, Harry Davies, Emily Kerkhof, Damien Ruban, Edward Day, Harry Owens, Luke Wood, Luke McDowell, Michael Wright and Scott James **(13/13 present)**

Agenda:

- Discussion of UML use case diagrams
- Discussion of database design
- Decision on next steps

Notes:

- UML Diagrams
  - Each team happy with states, transitions, and actors
  - Initial compilation of all UML diagrams has been created
  - Inconsistency around visual elements
- Database Design
  - Discussed current state: combination of human language descriptions and database design diagrams
  - Each team happy with their individual designs
- Requirement/endpoints for Airline companies
  - Searching for flight
  - Paying for flight
  - Specific details
  - **Cancel flight** (identified from spec as 'ability to manage bookings')
- Requirements/endpoints for payment services
  - Create order
  - Create payment
  - Make payment
  - Refund payments

Tasks for Next Meeting:

- Create API specification for all 3 components, following the structure created by the payment processing team
    - https://app.swaggerhub.com/apis/EDWARD57DAY/PaymentSystem/1
- Work on documentation around database design, ensure that each database has consistent foreign / primary key notation (using LucidChart)
- Combine and clean up existing UML diagrams: can be added to the Google Doc

Meeting 4

Date: 16/03/2023 **Time:** 14:00-15:00

Minutes taken by: Amy Porter

Attendees: Amy Porter, Shamita Datta, Ahmed Elkafrawy, Samuel Poirier, Harry Davies, Emily Kerkhof, Damien Ruban, Edward Day, Harry Owens, Luke Wood, Luke McDowell, Michael Wright and Scott James **(13/13 present)**

Agenda:
- Check overall team's morale and progression
- Document current-state review
- Database state discussion and alignment
- API end-points technical discussion

Notes:
- First point: morale and progress
    - all members feel happy on progress
    - aligned on how CW1 will flow into CW2
- Opened current state of the report and comparison to coursework specification
    - referencing → add some more references to sections, especially first section
    - added tagged comments to address specific report details
    - Changes needed to ensure all diagram figures are consistent in approach
    - Changes for report space eg. move user stories to appendix and replace with summary table
- Database state:
    - is it aggregator terminal based or is terminal talking to aggregator?
    - defines if we need aggregator end-points → command-line part of it or not
        - Answered on Teams: **aggregator is client → no endpoints**
- API state
    - need more content added to report to clearly outline and define endpoints for API specification

Tasks for Next Meeting:
- Each team to refine their group's work in the document
- Refine diagrams and add data types to database models
- Check all comments
- All API endpoints in document
- Ensure consistency in report
- Come to next meeting ready with report essentially in state **ready for submission**

Meeting 5

Date: 23/03/2023 **Time:** 12:00-13:00

Minutes taken by: Amy Porter

Attendees: Amy Porter, Shamita Datta, Ahmed Elkafrawy, Samuel Poirier, Harry Davies, Emily Kerkhof, Damien Ruban, Edward Day, Harry Owens, Luke Wood, Luke McDowell, Michael Wright and Scott James **(13/13 present)**

Agenda:

- Check document state- *Is it ready for submission?*
- Discuss interview: how will we split, structure and present
  - Interview slot booked: **Thursday 30th March 13:00-14:00**
- Any outstanding issues before submission of report and interview
- Conclusion and team feedback to wrap-up last official team meeting
  - Celebrate: full team attendance to all meetings?

Notes:

- Interview?
  - All can attend in person apart from Harry who will join virtually
  - 1 person- business sitch- Sham
  - 1/2 person - working principle- Emily
  - Then 1 from each team for architecture, database and api spec
    - Everyone should speak and divide into groups

- Ran through Document addressing last issues
- ***Well done team for brilliant attendance and progress!***

Outstanding team tasks:

- Do changes today/tomorrow
- Run through of interview

Detailed User Stories

Persona: Business Traveller
Gemma is a businesswoman, has an upcoming client conference in another city, and needs to book a flight for this. Through the aggregation service, Gemma wants to be able to find the best flight options for her, book efficiently and pay securely. so requires that booking and payment confirmation details and receipts be accessible to her after her booking so that she can submit them to get her expenses reimbursed by her employer.

Gemma starts her flight search by opening the flight aggregation service's user interface and entering the details of her proposed travel, including the travel dates and origin and destination cities. Gemma would like to book for one passenger (herself) and would prefer access to a business-class seat if possible. According to the entered search criteria, available flights from multiple airlines are displayed to Gemma with a clear and organised structure. Gemma filters the results to only show flights with business class seats available. As Gemma hates long flights, she sorts the filtered results by ascending flight length and then selects the shortest viable flight option for her business-class journey. With the ideal flight selected, Gemma enters her personal information and is then guided through a secure payment process by the payment provider she selects. Confirmation of her booking is displayed to Gemma, and sent to her email address, along with an itinerary of her trip that details the flight details.

On the day of travel, Gemma can easily check in for her flight using the itinerary details sent to her with her booking confirmation. She boards her flight and arrives at the conference destination on time. As the flight booking confirmation was sent to her via email, Gemma can easily pass this on as a receipt to get her travel expenses from her employer after her flight.

Persona: Family Holiday Booker
Mo is a husband and father of three young children and wishes to find and book flights for their upcoming family holiday to Spain. He wishes to use a flight aggregation service that is connected to airlines and payment providers so that he can find flight options that are convenient and affordable, and pay securely. For their holiday, they will be flying from Manchester to Madrid, but returning two weeks later from Barcelona. Mo needs to find a flight there and a flight back, whilst ensuring that his family has seats on the plane that are close to each other and that the flight choice is economical.

Mo initiates his flight search for his outbound journey to Madrid on the flight aggregator service, specifying two adult and three child passengers, the departure and arrival destination, and the desired date. He sorts the presented matching flight results by price to try and find the cheapest viable option, but he still feels all the options for the specified date are too expensive, even with economy class seats. The aggregator interface highlights to him that if he travelled the day before the date he selected, flight seats that fit his specification are cheaper. Happy with this alternative as the outbound date is flexible for his family, he selects a flight on the day prior. Within booking this flight, he specifies that he would like the seats to be as close together as possible, and continues with the booking -entering the details for each of his family members- and payment process, as facilitated by the aggregator service. Confirmation is displayed and sent to Mo, at which point he begins looking for and booking an equivalent flight for the return journey, now from Barcelona.
The sent confirmations and itineraries make it simple to check in and board their flights, so Mo can relax and enjoy their family holiday.

Persona: Frequent Traveller
Alex is a young travel enthusiast who is excited to plan their next adventure. They live in Birmingham and want to explore flight options departing from Birmingham International airport. Alex searches for a flight aggregator service that can help them find the best flight deals quickly and easily.

When Alex opens the flight aggregator service interface, they are greeted with a simple and intuitive user interface. They enter their departure point at Birmingham International airport and select the dates for their trip. The interface displays all the available flight options for the next week, including the price, airline, duration, and layovers. Alex can sort and filter the results based on their preferences and, as Alex likes to keep up with what is popular, Alex sorts the results by what locations are currently trending in flight bookings, in descending order. As Alex browses through the flight options, they find one that fits their budget and schedule to an exciting location they had seen on social media. They select the flight and proceed to the payment page. The flight aggregator service connects to a payment provider API and prompts Alex to enter their payment details. The interface is secure and easy to use, which makes the payment process hassle-free. Once Alex completes the payment, the flight aggregator service sends a request to the airline API to confirm the booking. The airline API responds with a confirmation message, which the flight aggregator service displays to Alex. The interface also sends a confirmation email to Alex with all the relevant flight details and booking reference numbers.

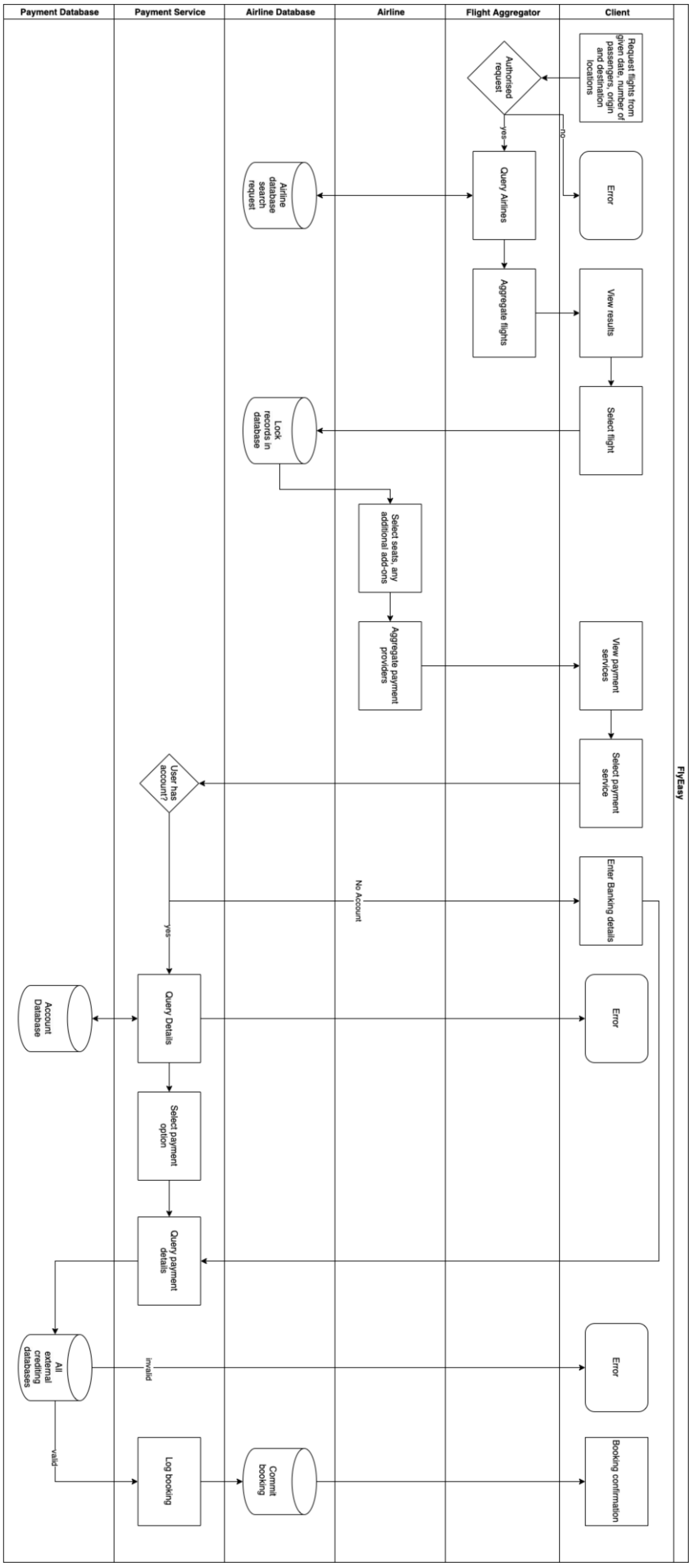
Alex is delighted with the seamless booking process and appreciates the ease of use of the flight aggregator service. They can now focus on planning the remainder of their trip and making the most of their adventure.

Persona: Romantic Holiday

A young couple decides to book a holiday together, they want to book somewhere for a short weekend away so decide to limit their search to Europe. They live in the North West, so decide to research flights to and from Manchester airport for the maximum number of options. They use the flight aggregator to select a cheap flight to Paris and book it a few months in advance.

Unfortunately, before the holiday arrives, the couple breaks up, and neither wants to go on the holiday any more. One of the partners logs into the flight aggregator flight and sends a request to cancel the holiday. The flight aggregator updates the flight company that the seats are no longer wanted, and the flight company requests a partial refund (after charging a cancellation fee). The payment company receives the refund request and returns the money to the original payment method.

Overall Proposed System Architecture Diagram


Payment Database
Payment Service
Airline Database
Airline
Flight Aggregator
Client
FlyEasy
Request flights from given date, number of passengers, origin and destination locations
Authorised request
yes
no
Error
Query Airlines
Airline database search request
Aggregate flights
View results
Select flight
Lock records in database
Select seats, any additional add-ons
Aggregate payment providers
View payment services
Select payment service
User has account?
No Account
Enter Banking details
yes
Query Details
Account Database
Error
Select payment option
Query payment details
All external crediting databases
invalid
Error
valid
Log booking
Commit booking
Booking confirmation