XJCO3011: Web Services and Web Data

Session #8–

Designing a RESful System

Instructor: Dr. Guilin Zhao

Spring 2023

SOUTHWEST JIAOTONG UNIVERSITY • UNIVERSITY OF LEEDS
西南交通大学－利兹学院
SWJTU-LEEDS JOINT SCHOOL

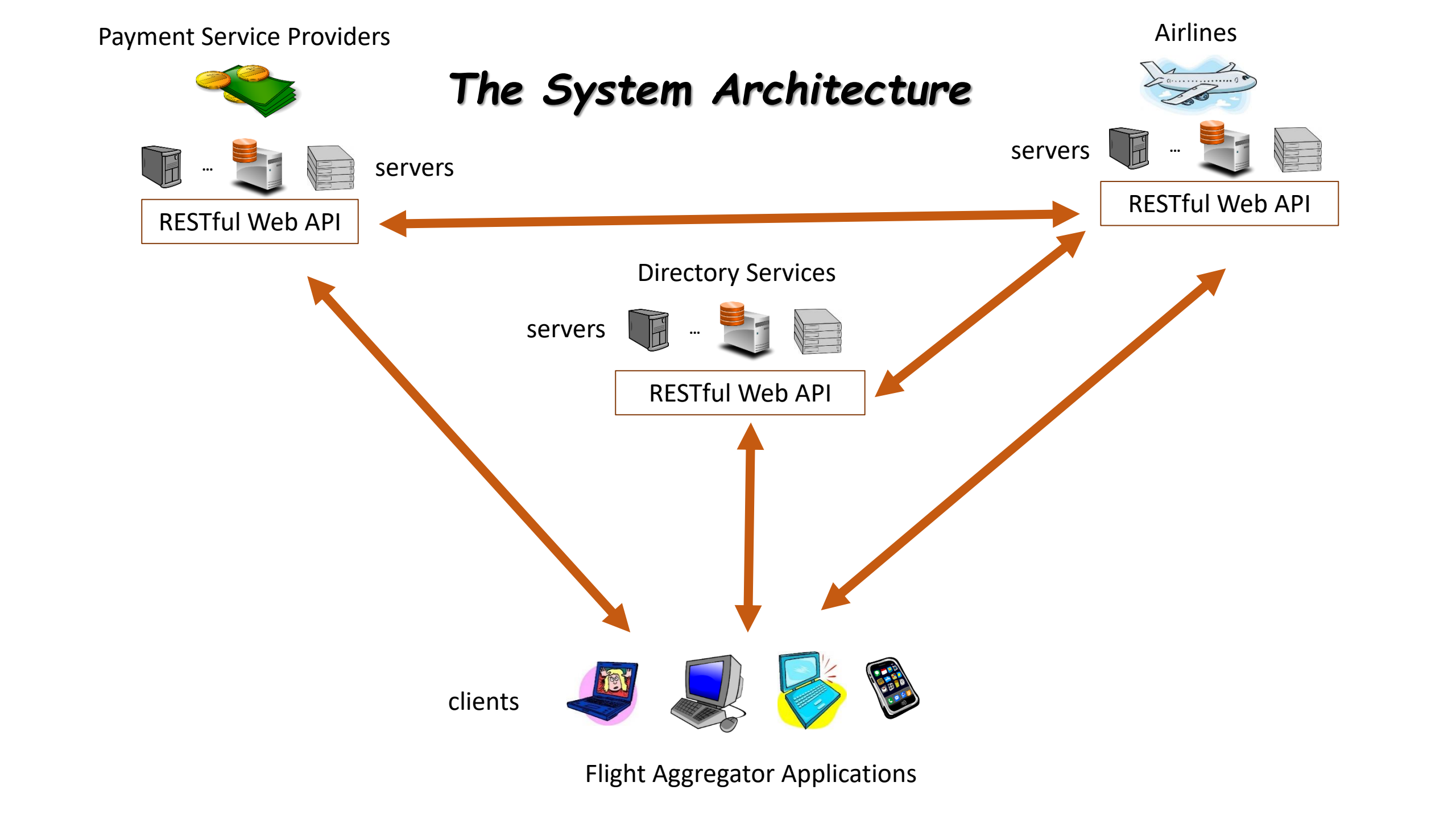
Design Process

1. Determine overall system architecture.
2. Clearly describe processes and procedures.
3. Design the database models.
4. Design the Web API.
   - Determine required services.
   - Assign URLs.
   - Choose suitable media types.
     - Media types are used in HTTP to indicate the type of content being transferred in the body of an HTTP message. Media types are used to identify these different types of data. such as HTML documents, images, audio, video, and so on.
   - Clearly define request and response messages.

Example
Flight Aggregation Systems

- People trying to book a flight do not search the website of one single airline
- Most people shop around in many airlines, then choose the flight that is most suitable for them in terms of price and total journey time (and some other factors).
- Flight search aggregator applications have emerged to assist customers to find the best flight for their needs, and then to book and pay for it online.

The System Architecture



The Booking Process

1. *Client sends a request to find a flight (request)*
2. *Airline server responds with a list of candidate flights (response)*
3. User selects a flight to book
4. Client sends a request to book the flight *(request)*
5. Airline server responds with booking number *(response)*
6. Client sends a request for available payment methods *(request)*
7. Airline responds with list of payment providers *(response)*
8. User chooses a payment provider
9. Client sends a request to pay for the booking with this provider *(request)*
10. Airline server contact payment server to create an electronic bill
11. Payment server responds to airline server with electronic bill
12. Airline responds to client with electronic invoice *(response)*
13. Client sends a request to payment provider to pay the electronic bill
    A. If the user (client) is not logged in to the payment server the server responds with a prompt to log in
    B. The user logs in (if necessary)
14. Payment server makes sure that *the user does have an account and enough balance*
15. Payment server responds with electronic receipt (confirmation of payment)
16. Client sends receipt to airline server *(request)*
17. Airline finalizes booking and responds with the confirmed booking details *(response)*

The Booking Process

User

1. Find me a flight

4. select a flight

5. flight selected

Client Software

2. find a flight

3. list of flights

6. book a flight

7. booking number

8. request payment methods

9. list of payment providers

10. select a payment provider

11. provider selected

12. pay for booking

15. electronic invoice

16. pay invoice

log in

user name & pass.

Airline Sever

13. request electronic invoice

14. electronic invoice

17. Not logged in

user name and password

log in success

resend pay invoice request

18. electronic stamp

19. electronic stamp

Payment Service Server

The Airline Database Model

- A database model is a blueprint or a set of rules that defines the structure, relationships, and constraints of data stored in a database.
- It's essential to have a database model to store and manage data efficiently (in data organization, data consistency, scalability, security, maintenance
- It provides a framework for organizing, securing, scaling, and maintaining data. It ensures that the system can handle large amounts of data, enforce data consistency, and protect data from unauthorized access.

Aircraft

- An airline business needs a number of *aircraft*
- For each aircraft the airline must maintain the following information:
  - The aircraft type (e.g. Airbus A320)
  - A unique tail or registration *number*
    - An aircraft tail number or registration number.
    - Each aircraft has a unique identifier to ensure accurate identification and tracking within the aviation industry.
  - The seating capacity (e.g. 150)
  - Other technical information (number of engines …)
  - This data can be maintained in a relational data base table such as this:

Primary Key	Tail Number	Type	Seats	Other
	G-STBA	Airbus A320	150	

Airports

- An airline must maintain a list of airports travelled to by the company
- For each airport the airline must maintain the following information:
  1. The airport unique name (e.g. New York JFK, Chengdu CTU)
  2. Country (e.g. USA)
  3. The time zone of the airport (e.g. USA EASTERN EST (Eastern Standard Time)), why ?
  4. Other information
- This data can be maintained in a relational data base table such as this:

Primary Key	Name	Country	Time Zone	Other
	New York JFK	USA	USA EST	

Flights

- An airline must provide flights
- For each flight the following information must be maintained:
  1. A unique flight number to distinguish it from other flights (e.g. BA1349)
  2. Departure airport (e.g. London Heathrow LHR)
  3. Destination airport (e.g. New York JFK)
  4. Departure data-time (e.g. 2018.04.01, 14:45:00)
  5. Arrival date-time
  6. Flight duration (e.g. 5 hours)
  7. The aircraft used for this flight
  8. Seat cost
  9. Seat price
  10. Other

Primary Key	Flight Num.	Departure Airport (FK)	Destin. Airport (FK)	Departure DateTime	Arrival DateTime	Duration	Aircraft (Foreign Key)	Seat Price	Seat Cost
	BA1349	→	→	2018.04.01	-	5:00	→	£500	£450
	BA1349	→	→	2018.04.02	-	5:00	→	£500	£450

FK

- "FK" typically refers to a foreign key.
- A foreign key is a column or set of columns in a database table that refers to the primary key of another table. It is used to establish a link or relationship between two tables in a database. FK can be used to link related resources and ensure consistency between them.
  - E.g., consider a database with two tables: "Orders" and "Customers". The "Orders" table has a foreign key column "customer_id" that refers to the "id" column in the "Customers" table. This establishes a one-to-many relationship between the two tables, where each order is associated with a single customer, and each customer can have many orders.

Primary Key	Flight Num.	Departure Airport (FK)	Destin. Airport (FK)	Departure DateTime	Arrival DateTime	Duration	Aircraft (Foreign Key)	Seat Price	Seat Cost
	BA1349	→	→	2018.04.01	-	5:00	→	£500	£450
	BA1349	→	→	2018.04.02	-	5:00	→	£500	£450

Bookings

- To allow passengers to book seats on a flight, an airline provides flight bookings.
- For each flight booking the following information must be maintained:
  1. A unique booking number to distinguish it from other bookings (e.g. WXY12Z)
  2. The flight of this booking
  3. Number of seats booked
  4. Details of each passenger (full name, nationality, passport number)
  5. Booking status (e.g. ON_HOLD, CONFIRMED, CANCELLED, TRAVELLED …)
  6. Other information

This is NOT one of the fields in the relational database table. I have just added it to indicate that there exists a many-to-many relation with the passengers table through the join table

PK	Booking Number	Flight (FK)	No Seats	Passengers	Status	Other
8	WXY12Z	→	2	ManyToMany	ON_HOLD	

passengers = models.ManyToManyField (Passenger)

In the Django model this relation is created like this

PK	Booking FK	Passenger FK
50	8	12
51	8	13

Join Table (passengers on each booking)

PK	Name	Nationality	Passport Number
12	John West	UK	123456789
13	Ben East	UK	987654321

Many-to-Many

- Flight bookings and passengers have a many-to-many relationship, which means that a single flight booking can have multiple passengers, and a single passenger can have multiple flight bookings.
- In the context of a flight booking system, this relationship can be represented by a junction table that links the flight booking table and the passenger table. This junction table typically includes foreign keys to the flight booking and passenger tables, creating a many-to-many relationship between the two.
- This type of relationship is common in many industries, and it's often represented in database systems using a junction or bridging table to establish the association between the two entities.
- It allows for greater flexibility and scalability when managing large amounts of data related to multiple entities, and it also makes it possible to retrieve and query data more efficiently.

Other Tables

Payment providers

To allow an airline to accept payments from customers, an airline must have accounts with a number of payment service providers. For each account the airline must maintain the following data:

- The name of the payment service provider (e.g. SalPay)
- The address of the website of the payment service provider (e.g. ‘www. salpay.co.uk’)
- The account number.
- The login name and password (this is needed when the airline server must access their account to create an electronic invoice for a customer).

Invoices

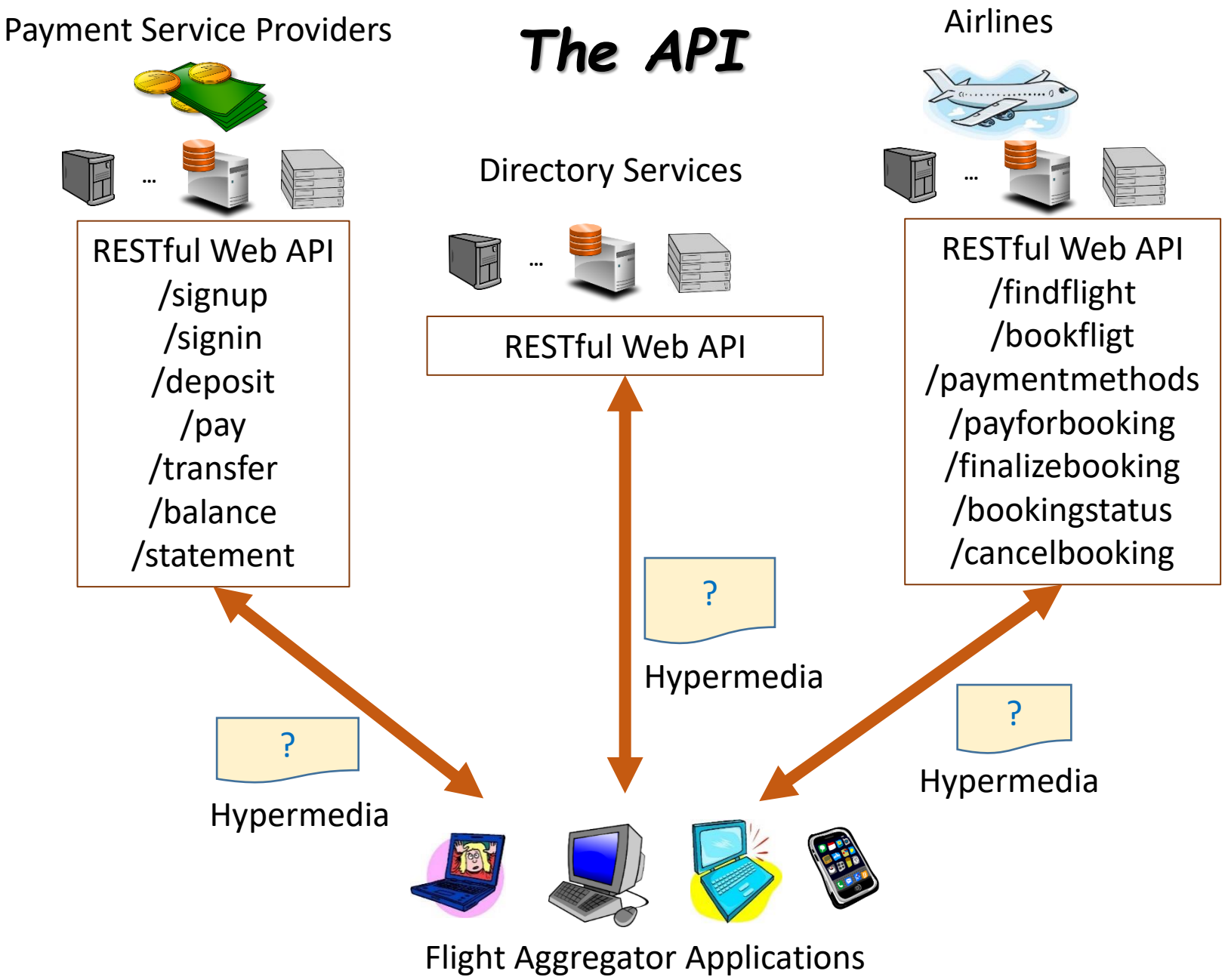
To be able to charge customers and accept payment, the airline must ask a payment service provider to create an invoice (see below). A copy of the invoice is kept at the airline database to match against customer payments. For each invoice the following information is maintained in the database:

- A unique reference number for this invoice at the airline’s database.
- The unique reference number for this invoice within the database of the payment service provider.
- The booking number for which the invoice was issued.
- The amount of the invoice.
- A Boolean value indicating whether this invoice was paid or not.

A unique 10 digit alphanumeric code (electronic stamp). This key is generated by the payment service provider when the invoice is created. The airline should remove this code from the invoice before forwarding it to the client. When payment is made for this invoice through the payment provider’s website, the payment provider releases this code to the client (payer). The client will then send this electronic stamp code to the airline as a confirmation of payment. The airline verifies this code against that stored in its copy of the invoice and confirms payment.

The Airline Web API

- What web services (web APIs) the airline must provide to client applications to facilitate finding and boking a flight, and what data must be exchanged between clients and servers for each service (request)?
- We can answer this question by examining the steps of the booking process and identify those related to the client's interaction with the airline web API, then:
  1. Decide if this should be a GET or POST request.
  2. Assign a resource URI for this service.
  3. Identify the data the client must provide in order for the server to be able to serve the request
  4. Identify the appropriate response and the data the server must send back to the client to fulfil the request
  5. Decide on a media type for the data in the response
- Remember that a GET request should be used when we do NOT intend to change the state of the server (read only), while a POST request are used when we need to change the server's state (write or modify data).


Payment Service Providers
The API
Airlines
Directory Services
RESTful Web API
/signup
/signin
/deposit
/pay
/transfer
/balance
/statement
RESTful Web API
RESTful Web API
/findflight
/bookfligt
/paymentmethods
/payforbooking
/finalizebooking
/bookingstatus
/cancelbooking
?
Hypermedia
?
Hypermedia
?
Hypermedia
Flight Aggregator Applications

Client sends a request to find a flight and server responds with a list of candidate flights

Service Aim: Find a flight from a departure airport to a destination airport at some departure date.

Service Details:

The client sends a GET request to /api/findflight with the following data:

1. Departure airport ("dep_airport", string)
2. Destination airport ("dest_airport", string)
3. Departure date ("dep_date", string)
4. Number of passengers ("num_passengers", number)
5. A Boolean value indicating whether the departure date is exact or flexible ("is_flex" , true or false)

If the request is processed successfully, the server responds with 200 OK and a list of flights in a JSON payload. For each flight in the list, the following data must be provided:

1. Flight identifier that uniquely identifies a flight on a certain date ("flight_id", string)
2. Flight number ("flight_num" , string)
3. Departure airport ("dep_airport", string)
4. Destination airport ("dest_airport", string)
5. Departure date and time ("dep_datetime", string)
6. Arrival date and time ("arr_datetime", string)
7. Flight duration ("duration" , string)
8. Seat price for one passenger ("price" , number)

If no flights are found the server should respond with 503 Service Unavailable with text/plain payload giving reason.

Client sends a request to book a flight and server responds with booking number

Service Aim: Book a flight given by its unique identifier.

Service Details:

The client sends a POST request to /api/bookflight with the following data in a JSON payload:

1. Flight unique identifier ("flight_id", string)
2. A list of passengers ("passengers" , array) with the following details about each passenger:
   - First name ("first_name", string)
   - Surname ("surname" , string)
   - email ("email", string)
   - phone number ("phone", string)

If the request is processed successfully, the server responds with 201 CREATED and a JSON payload containing the following data:

1. Booking number ("booking_num", string)
2. Booking status ("booking_status", string) = "ON_HOLD"
3. Total price for this booking ("tot_price", number)

If a booking cannot be made for any reason (e.g. no seats are available), the server should respond with 503 Service Unavailable with text/plain payload giving reason

Client sends a request for available payment methods and server responds with a list of payment providers

Service Aim: Request Payment Methods

Service Details:

The client sends a GET request to /api/paymentmethods :

If the request is processed successfully, the server responds with 200 OK and a list of payment service providers ("pay_providers" , array) in a JSON payload. For each provider in the list, the following data must be given:

1. The provider unique identifier in the airline's database ("pay_provider_id", string)
2. Provider name ("pay_provider_name", string)

If no providers are available, the server should respond with 503 Service Unavailable with text/plain payload giving reason.

Client sends a request to pay for a booking with a certain provider and server responds with electronic invoice

Service Aim: Pay for a Booking

Service Details:

The client sends a POST request to /api/payforbooking with the following data in a JSON payload:

1. Booking number ("booking_num", string)
2. Payment provider identifier ("pay_provider_id", string)

If the request is processed successfully, the server responds with 201 CREATED and a JSON payload with the following data:

1. Payment provider identifier ("pay_provider_id", string)
2. Payment provider invoice unique id ("invoice_id", string)
3. Booking number ("booking_num", string)

If the server is unable to process the request for any reason (e.g. could not establish connection to the server of the payment service provider), the server should respond with a 503 Service Unavailable with text/plain payload giving reason.

Client sends a request to finalize (confirm) a booking and server responds with the confirmed booking details

Service Aim: Finalize a Booking

Service Details:

The client sends a POST request to /api/finalizebooking with the following data in a JSON payload:

1. Booking number ("booking_num", string)
2. Payment provider identifier ("pay_provider_id", string)
3. Payment provider receipt id ("receipt_id", string)

If the request is processed successfully, the server responds with 201 CREATED and a JSON payload with the following data:

1. Booking number ("booking_num", string)
2. Booking status ("booking_status", string) = "CONFIRMED"

If the server is unable to process the request for any reason, the server should respond with a 503 Service Unavailable with text/plain payload giving reason.

Client sends a request to know the status of a booking and server responds with the booking details and status

Service Aim: Provide Booking Status

Service Details:

The client sends a GET request to /api/bookingstatus with the following data:

1. Booking number ("booking_num", string)

If the request is processed successfully, the server responds with 200 OK and a JSON payload with the following data:

1. Booking number ("booking_num", string)
2. Booking status ("booking_status", string)
3. Flight number ("flight_num" , string)
4. Departure airport ("dep_airport", string)
5. Destination airport ("dest_airport", string)
6. Departure date and time ("dep_datetime", string)
7. Arrival date and time ("arr_datetime", string)
8. Flight duration ("duration" , string)

If the server is unable to process the request for any reason, the server should respond with a 503 Service Unavailable with text/plain payload giving reason.

Client sends a request to cancel a booking and server responds with the booking status

Service Aim: Cancel a Booking

Service Details:

The client sends a POST request to /api/cancelbooking with the following data in a JSON payload:

1. Booking number ("booking_num", string)

If the request is processed successfully, the server responds with 200 OK and a JSON payload with the following data:

1. Booking number ("booking_num", string)
2. Booking status ("booking_status", string) "CANCELLED"

If the server is unable to process the request for any reason, the server should respond with a 503 Service Unavailable with text/plain payload giving reason.