# Distributed Systems - Exercise 1

TOC

[Distributed Systems - Exercise 1 1](#_Toc372151028)

[Design and implementation Consideration 1](#_Toc372151029)

[Design Considerations 1](#_Toc372151030)

[High Level Design 2](#_Toc372151031)

[Executables & parameters 3](#_Toc372151032)

[Services detailed description 3](#_Toc372151033)

[FlightsSearchReg 3](#_Toc372151034)

[FlightsSearch (as ClientQueryService) 3](#_Toc372151035)

[TicketsSellingServer 4](#_Toc372151036)

[Services errors summary 5](#_Toc372151037)

## Design and implementation Consideration

### Design Considerations

The design requires 3 entities where one consumes & make transactions through a proxy search server which delegates the requests to all subscribers.

The solution comprised of 2 Servers and a client. The data flow consists of 3 Services.

Client: a command line REST application which is used to book and search for flights. The client receives a URI to a flights search server which serves as a search engine and a flights contractor.

Flights Search Server: a contractor for flight sellers and aggregator of data to clients which look for flights to book (or cancel…). The flights server exposes 2 services, a RESTful service in which client queries/transaction are coming from and a RESTful service for flight sellers to register for search client search events which are in turn being aggregated to the client. The flight server is the first entity which must be executed during startup.

Flights Ticket Seller: this server is the main data supplier for the system, each Flights ticket seller register for search events in the contractor *Flights Search Server* through a RESTful interface by supplying its name & URI and in turn it exposes a SOAP interface in which the search server can submit queries/transactions.

## High Level Design



## Executables & parameters

Executables parameters are with compliance to the Homework spec.

FlightsSearchServer is publishing RESTful services on /Services/FlightsSearch & /Services/FlightsSearchReg for Clients & Sellers respectively.

(see detailed *Design & Considerations* section).

TicketSellingServer is publishing on /Services/ SOAP interface.

## Services detailed description

### FlightsSearchReg

#### Registration

This method allows different Ticket selling servers to register the searching server. The Ticket selling server must supply its Identification name (the way it's exposed to the clients) and its own URI which will allow the Flights search server to open SOAP connection and issue requests to it.

Because the registering server is responsible for the location of the URI the request is a PUT request.

### FlightsSearch (as ClientQueryService)

#### GetFlights

Clients use this method to search for a relevant flight, the client must supply a source, destination and date. Those parameters are supplied as GET parameters because they are filtering options and they don't change anything in the server's state (except for dropped ticket seller connections – which is irrelevant and seamless to the user).

#### MakeReservation

Clients use this method to reserve a flight. The client must supply the seller's name in the URI path in order have a consistent hierarchical URI structure when acting upon a specific seller (canceling a reservation for example). The rest of the parameters are supplied as a reservation request object. This method will change state on seller which this request is acting on (URI).

If the seller is not found a "Seller not found" error will be returned to the client. If the Seller server returns an error, for ex. any of the internal parameters - flight number or date, doesn't exist or inconsistent with an entry in the Seller's database or there are no more seats on the flight a "Flight not found" or "no seats available" error will be returned to the client respectively and a 400 error is returned (Resource not found).

Upon success the client will receive reservation ID which is also used as an approval. Each seller may supply its own reservation ID and a 1:1 correlation to the original flight is made by seller name and reservation ID pair.

POST method is used because the client is not responsible to the final location of the resource created (the reservation ID).

#### CancelReservation

The client may cancel a reservation by accessing this method. Its URI consists of the seller's name (which is used to identify it) and a reservation ID to cancel (see note about correlation in previous method). The reservation ID appears as part of the URI and not as an additional parameter because it represents an existing resource in the system.

Cancelation may throw "Seller not found" if seller really doesn't exists or crashed (we do not maintain a local database in the SearchServer, we just delegate requests). If reservation id is bogus a "Reservation not found" error is returned to the client (and a 400 status).

For cancel we use a DELETE method because it removes a resource from the system.

### TicketsSellingServer

#### GetFlights

SOAP interface which is used by the search server. The search server submits a FlightQuery request which holds the flight information the the user is searching for (Source, destination, date). If the internal database doesn't contain flights which match this criteria the method will return an empty list.

#### MakeReservation

SOAP interface which is used by the search server. The search server submits a FlightSearchReservationRequest which holds the reservation's details (flight number & date).

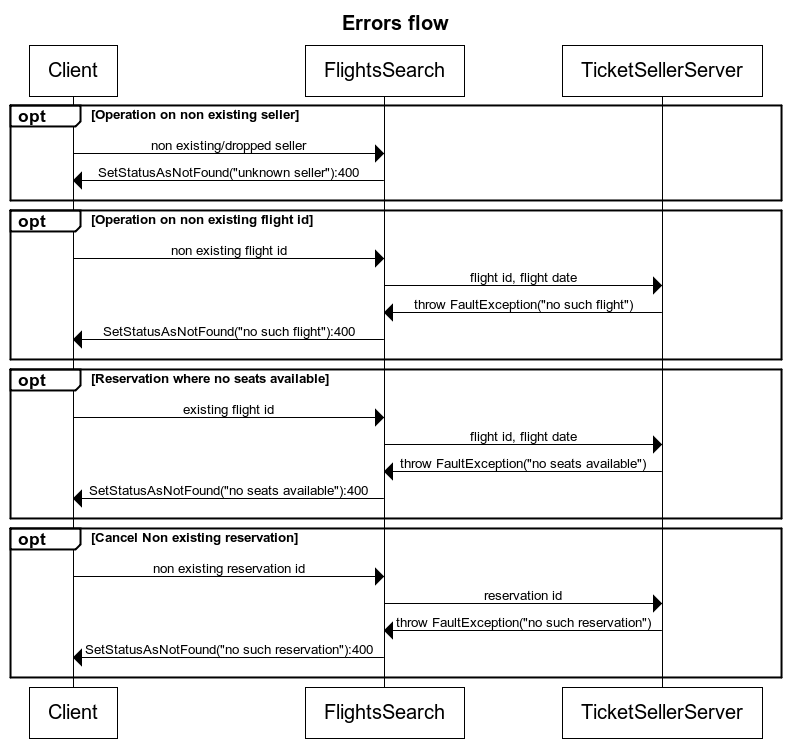
If a valid request is made (flight number & date match an actual entry in the seller's database) a reservation id is returned. The reservation id is used to identify it.

If flight number doesn't exist a "no such flight", if no seats available a "no seats available" error is returned.

#### CancelReservation

SOAP interface which is used by the search server. A previously made reservation may be canceled using this method. If reservation ID doesn't exist a "no such reservation" excetion error is returned to the FlightSearchServer.

## Services errors summary



* The client will print the string inside the error.
* The status code matched the :XXX.