Options for the Semester Project – fall 2016

<u>You can chose between these three options for the semester project</u>

1. Implement your own simple "Momondo" Server
2. Implement our business idea, described as "Buy and Sell it All"
3. Come up with your very own project proposal

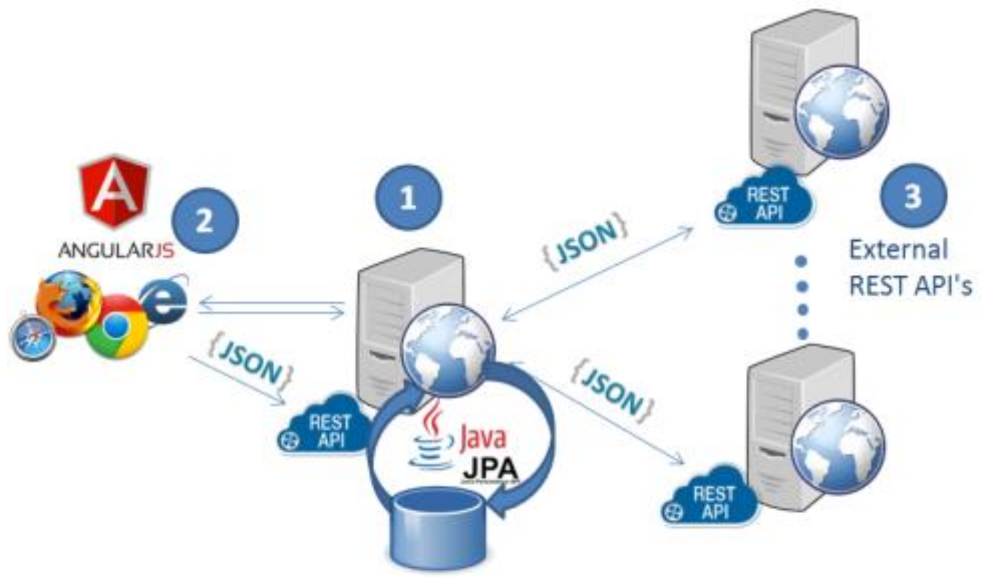
Which one you choose depends on your personal, and group, ambitions and also on your current professional skills. Take is as; 2 + 3 are the most challenging proposals and 1 the more straight forward proposal (but still challenging) with a possibility to bring you through all the technologies we have been around this semester, one more time.

The following will introduce the three proposals (in reverse order).

Come up with your own proposal for a project.

This has to be approved by us, and should *ideally* include an architecture inspired by the one suggested below, but not necessarily all of it - we can accept much, given an exciting real-life project ;-)

Design a web-application (1) which uses JPA for data, provides end user pages with AngularJS that uses REST/JSON to update pages (2) and communicate with external servers in order to serve a request (think Momondo, Hotels.com, etc). The idea could be one you come up with (you might have an idea that could turn into a great business idea), or an idea from an external company.



Buy and Sell it All

This is an idea we came up with, while planning the transition from module3 -> module 4+5. Someone else has probably already had the same idea, but come up with a great solution and put it on the market before them, and you could end up as millionaires ;-)

The idea is that you should create a site, somehow similar to sites like *pricerunner.dk*, but with focus on items sold by private people.

If I for example, would like to buy an old and famous recording with the band "velvet underground" I would enter the request into your site, and your server will contact:

- dba.dk
- http://www.guloggratis.dk/
- ebay.com
- facebook
- …

and make a response, built from the responses from these sites.

The architecture will end up similar to what was explained for proposal 1, but the task of contacting all these remote companies/servers will probably be challenging.

If these companies do not offer and API to programmatically access their data, your only option is to use Screen Scraping[1]. Feel free to ask for hints about this, but Google will be your best friend.

If you find that the task is too challenging for some of the sites, you could consider paying someone else to do the job ;-)

Note: There can be legal implications with this proposal, see the following links before deciding for this option:

English: http://www.out-law.com/en/articles/2015/january/website-operators-can-prohibit-screen-scraping-of-unprotected-data-via-terms-and-conditions-says-eu-court-in-ryanair-case/

Danish: https://www.kromannreumert.com/Nyheder/2015/01/Ny-EU-afgoerelse-begraenser-brugen-af-screen-scraping

[1] https://en.wikipedia.org/wiki/Web_scraping

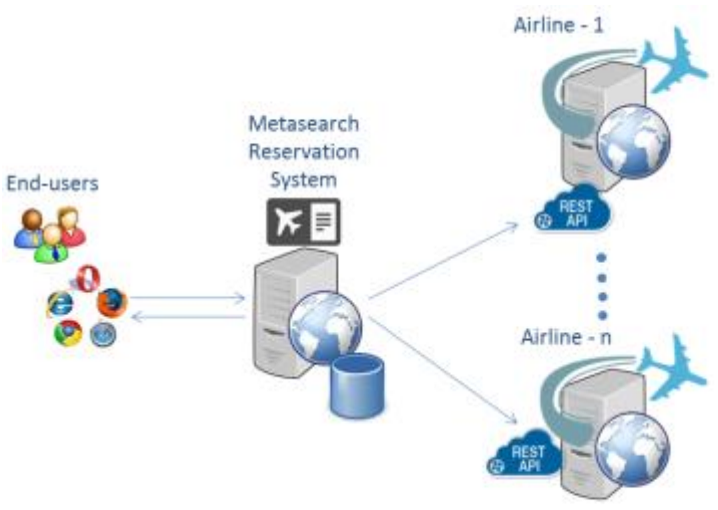
Semester Project – Online Reservation System

Initial Description

The task for this project is to create a Proof of Concept solution for an Online Reservation System, similar to Web sites like Momondo.com, Expedia.com and others. The solution must include the following subsystems:

- A Prototype for an actual airline that exposes its reservation system via a REST API (Airline 1-n, below)
- The Metasearch Reservation System

The overall architecture of the system is sketched in the figure below.



Non-functional Requirements

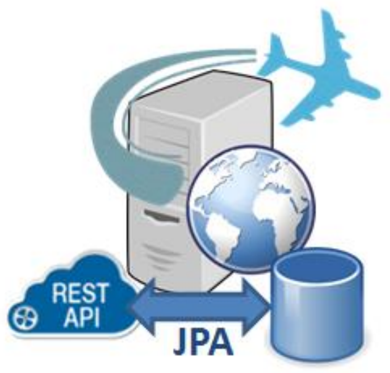
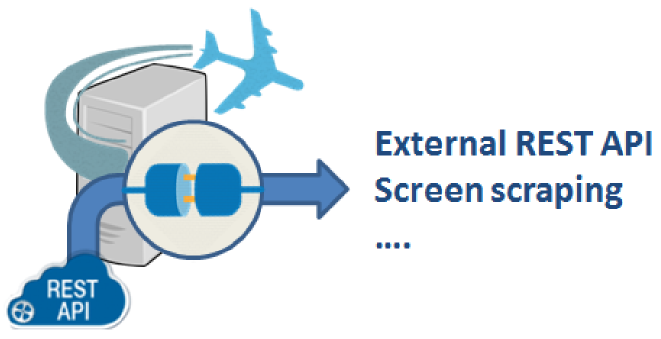
No one expects you to come up with a complete system, even close to something, ready to go to production. In order to simplify Inter Server Communication we will assume that all airline-companies have agreed on Common REST/JSON-Protocol for data exchange between Airlines and Metasearch Reservation Systems.

This protocol is given at the end of this description.

Verify that the protocol is precise enough to let another person/company implement the airline company, while you in parallel develop the Metasearch Reservation System.

There are a number of non-functional requirements you have to obey for this projects as outlined below:

- The Airline System can be implemented using either of the following strategies:

Dummy-Airline using JPA and JAX-RS	**Real data using screen scraping or external APIs**
	
Implement a "dummy airline" using JPA and implement the public REST API using JAX RS. Set up a number of dummy flights in the database and publish your available flights so that other teams knows what to search for	*Fetch real flight information, using screen scraping, or available APIs like QPX Express API*[2] *or similar. Implement an "adapter service" that transforms between our REST API and the external source you are using.*

- Parts of the system must be "outsourced", that is you have to specify and buy this part using sites like *freelancer.com*, *upwork.com* or similar.
  Suggestions for what to "buy":
  - Your Metasearch Site Design, logo etc.
  - All, or parts, of the Airline-company
  - The Screen Scraping Part of the Airline Company
  - The Airline-company including deployment (instructions).
  - Specific Angular-pages
  - AngularJS Consultancy
  - Typing in, Test Data
  - Etc.
- The Metasearch Reservation System must be implemented as a Java Based solution, using a Relational Database and JPA for persistence.
- The Web Client must be implemented as a Single Page Application (SPA), using the AngularJS Framework.
- Code must be thoroughly tested, including JUnit Tests on the Backend(s), and a number of Postman based test of the API

2 https://developers.google.com/qpx-express/

REST API that must be implemented by all "Semester-Airlines"

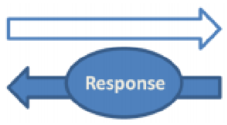
Method	URL	Description	Request-Data (JSON provided with the request)	Response-data (JSON response from the request)	Error
GET	`api/flights/:from/:date/:tickets` <u>Parameters:</u> **from**: Start Airport (as an IATA Code) **date:** Travel date (as a ISO-8601 date) **tickets** : Requested amount of tickets (integer)	Request all available flights that matches the provided search criteria's	None, besides the parameters given in the URL	See ***response-1*** below	**See *Flight-Exception* below**
GET	`api/flights/:from/:to/date/:tickets` <u>Parameters:</u> **from**: Start Airport (as an IATA Code) **to**: Destination Airport (as an IATA **date:** Travel date (as a ISO-8601 date) **tickets** : Requested amount of tickets (integer)	Request all available flights that matches the provided search criteria			**See *Flight-Exception* below**

POST	**`api/reservation/:flightId`**
	
	`flightId:` `id returned by on the two GET requests`
		Make a reservation for flight and persons provided with request	See ***reservationRequest*** below.	See ***reservationResponse*** below	**See *Flight-Exception* below**

JSON Format for Data Exchange between Server and Client

<table>
<tr>
<td>

Response-1



</td>
<td>

JSON – Object

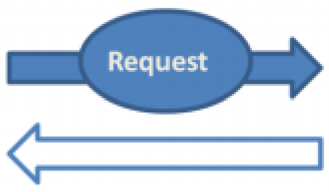
```
{
   "airline":String,
   "flights":[
      {
         "flightID": String,
         "flightNumber" : String
         "date": ISO-8601 String (date+time),
         "numberOfSeats": Integer,
         "totalPrice": Number (Euro),
         "traveltime": Integer (minutes),
         "origin":"IATA-Code (String),
         "destination": IATA-Code (String)
      }
   ],...
}
```

Example

```
{
  "airline": "AngularJS Airline",
  "flights": [
    {
      "flightID": "2257-1457179200000",
      "flightNumber": "COL2257",
      "date": "2016-03-05T13:00:00.000Z",
      "numberOfSeats": 3,
      "totalPrice": 180,
      "traveltime": 120,
      "origin": "CDG",
      "destination": "CPH",
    }
  ]
}
```

</td>
</tr>
<tr>
<td>

reservationRequest



</td>
<td>

JSON – Object

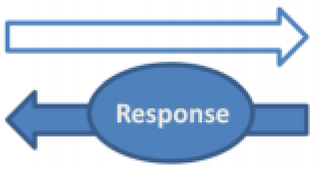
```
{
   "flightID": String,
   "numberOfSeats": Integer,
   "reserveeName": String,
   "reservePhone": String,
   "reserveeEmail": String (valid email),
   "passengers":[
      {
         "firstName":String,
         "lastName": String
      }
   ]
}
```

Example

```
{
   "flightID":"2256-1459929600000",
   "numberOfSeats":2,
   "reserveeName":"Peter Hansen",
   "reservePhone":"12345678",
   "reserveeEmail":"peter@peter.dk",
   "passengers":[
      { "firstName":"Peter","lastName":"Peterson"},
      { "firstName":"Jane","lastName":"Peterson"}
   ]
}
```

</td>
</tr>
</table>

reservationResponse



JSON – Object

```
{
   "flightNumber":"String",
   "origin":"String (Friendly name + IATA)",
   "destination":"String (Friendly name + IATA)",
   "date":"ISO-8601-Date/time",
   "flightTime":"Integer (minutes)",
   "numberOfSeats":"Integer",
   "reserveeName":"String",
   "passengers":[
      {
         "firstName":"String",
         "lastName":"String"
      }
   ]
}
```

Example

```
{
  "flightNumber": "COL2256",
  "origin": "Copenhagen Kastrup(CPH)",
  "destination": "Charles de Gaulle International(CDG)",
  "date": "2016-04-06T10:00:00.000Z",
  "flightTime": 120,
  "numberOfSeats": 2,
  "reserveeName": "Peter Hansen",
  "passengers": [
    {
      "firstName": "Peter",
      "lastName": "Peterson"
    },
    {
      "firstName": "Jane",
      "lastName": "Peterson"
    }
  ]
}
```

JSON Format and HTTP codes for Errors Reported by the Server

All errors responses must be reported using both one of the following HTTP-Status Codes and a supplementary JSON Error Response given below:

HTTP Error Code

- HTTP-400 (all errors relating to the client side, like illegal data etc.)
- HTTP-404 (resource does not exist)
- HTTP-500 (For all Unchecked Exceptions Thrown on the Server)

FlightException - Error Response JSON

```
{
   "httpError": integer (400, 404, 500),
   "errorCode": integer (see below),
   "message": A descriptive message, meant for end users
}
```

How to interpret the *errorCode*:

1: No Flights
2: None or not enough available tickets
3: Illegal Input
4: Unknown error
10 -> You can define your own codes here