Web Engineering Project Documentation

Dimitris Aktsoglou (s2979616), Emanuel Nae (s2931931), Daan Groot (s2958287)

March 29, 2019

1 Introduction

The first goal of this project is to design an API that is able to collect and distribute data on demand for all USA airports and carriers. The second goal is to create the API and the final goal is to create a web application that interacts with the API. The given data includes statistics about flights and delays for carriers that operate at US airports.

The decisions we make in the design of our API are based on RESTful together with HATEOAS design principles. REST design is in general based upon the following principles:

1. Resources, which are any kind of object, data, or service that can be accessed by the client.
2. Resource has an identifier, which is a URI that uniquely identifies that resource.
3. Client interacts with a service by exchanging representation of resources.
4. Use a uniform interface, which helps to decouple the client and service implementations.

Furthermore there are four level of "Maturity" that have been defined in such a design. Our goal is to design an API as close as possible to Maturity level 3.

HATEOAS (Hypermedia as the Engine of Application State) is a constraint of the REST application architecture. A hypermedia-driven site provides information to navigate the site's REST interfaces dynamically by including hypermedia links with the responses.

2 Milestone 1: API Design

2.1 Resources

Resources refer to the information that can be returned by an API. In our case, the various resources that can be returned are the following:

1. **Airport object**: represents an airport located in the USA.
   Fields:

`code:string`	The 3-letter code for this airport.
`name:string`	The full name of this airport.

   JSON example:

```
{
    "airport": {
        "code": "ATL",
        "name": "Atlanta, GA: Hartsfield-Jackson Atlanta International"
    }
}
```

CSV example:

```
airportCode,airportName
ATL,"Atlanta, GA: Hartsfield-Jackson Atlanta International"
```

2. **Carrier object**: represents a carrier operating at US airports.
   Fields:

`code:string`	The 3-letter code for this carrier.
`name:string`	The full name of this carrier.

   JSON example:

```
{
    "carrier": {
        "code": "AA",
        "name": "American Airlines Inc."
    }
}
```

   CSV example:

```
carrierCode,carrierName
AA,American Airlines Inc.
```

3. **FlightData object**: represents statistics about flights.
   Fields:

`cancelledCount:int`	The number of flights that were cancelled.
`onTimeCount:int`	The number of flights that were on time.
`delayedCount:int`	The number of flights that were delayed.
`divertedCount:int`	The number of flights that were diverted.
`totalCount:int`	The total number of flights.

   JSON example:

```
{
    "flightData": {
        "cancelledCount": 5,
        "onTimeCount": 561,
        "delayedCount": 186,
        "divertedCount": 0,
        "totalCount": 752
    }
}
```

CSV example:

```
cancelledCount,onTimeCount,delayedCount,divertedCount,totalCount
5,561,186,0,752
```

4. **DelayData object**: represents statistics about the number of delays for various reasons.
   Fields:
   `lateAircraftCount:int` The number of delays and cancellations caused by a previous flight with the same aircraft arriving late.
   `carrierCount:int` The number of delays and cancellations due to circumstances within the airline's control.
   `weatherCount:int` Number of delays or cancellations caused by significant meteorological conditions.
   `securityCount:int` Number of delays or cancellations caused by evacuation of a terminal or concourse, re-boarding of aircraft because of security breach, inoperative screening equipment and/or long lines in excess of 29 minutes at screening areas.
   `nationalAviationSystemCount:int` The number of delays and cancellations attributable to the national aviation system.

   JSON example:

   ```
   {
       "delayData": {
           "lateAircraftCount": 18,
           "carrierCount": 34,
           "weatherCount": 28,
           "securityCount": 2,
           "nationalAviationSystemCount": 105
       }
   }
   ```

   CSV example:

   ```
   lateAircraftCount,carrierCount,weatherCount,securityCount,
   nationalAviationSystemCount
   18,34,28,2,105
   ```

5. **DelayTimeData object**: represents statistics about the total duration of delays in minutes for various reasons.
   Fields:
   `lateAircraftTime:int` The number of minutes delayed caused by a previous flight with the same aircraft arriving late.
   `carrierTime:int` The number of minutes delayed due to circumstances within the airline's control.
   `weatherTime:int` Number of of minutes delayed caused by significant meteorological conditions.
   `securityTime:int` Number of minutes delayed caused by evacuation of a terminal or concourse, re-boarding of aircraft because of security breach, inoperative screening equipment and/or long lines in excess of 29 minutes at screening areas.
   `nationalAviationSystemTime:int` The number of minutes delayed attributable to the

national aviation system.

`totalTime:int`	The total number of minutes delayed.

JSON example:

```
{
    "delayData": {
        "lateAircraftTime": 1269,
        "carrierTime": 1367,
        "weatherTime": 1722,
        "securityTime": 139,
        "nationalAviationSystemTime": 3817,
        "totalTime": 8314
    }
}
```

CSV example:

```
lateAircraftTime,carrierTime,weatherTime,securityTime,
nationalAviationSystemTime,totalTime
1269,1367,1722,139,3817,8314
```

6. **ExtraStatisticData object**: represents the additional statistics: mean, median, and standard deviation, for carrier-specific delay durations.
Fields:

`lateAircraftTimeMean:float`	The mean of the number of minutes delayed for the late aircraft reason.
`lateAircraftTimeMedian:float`	The median of the number of minutes delayed for the late aircraft reason.
`lateAircraftTimeSd:float`	The standard deviation of the number of minutes delayed for the late aircraft reason.
`carrierTimeMean:float`	The mean of the number of minutes delayed for the carrier reason.
`carrierTimeMedian:float`	The median of the number of minutes delayed for the carrier reason.
`carrierTimeSd:float`	The standard deviation of the number of minutes delayed for the carrier reason.

JSON example:

```
{
    "lateAircraftTimeMean": 3544.79,
    "lateAircraftTimeMedian": 3183,
    "lateAircraftTimeSd": 2164.63,
    "carrierTimeMean": 3313.05,
    "carrierTimeMedian": 2912,
    "carrierTimeSd": 1741.66
}
```

CSV example:

```
lateAircraftTimeMean,lateAircraftTimeMedian,lateAircraftTimeSd,
carrierTimeMean,carrierTimeMedian,carrierTimeSd
3544.79,3183.0,2164.63,3313.05,2912.0,1741.66
```

7. **Link object**: represents a hypermedia link to another endpoint.
   Fields:
   `href:string` URL of another endpoint.
   `rel:string` The relationship between the data and link.
   `type:string` The type of requests (GET,POST,PUT,DELETE) that are support by the endpoint.

   JSON example:

   ```
   {
       "href": "http://94.212.164.28:8080/airports/ATL/",
       "rel": "airport",
       "type": "GET"
   }
   ```

   Links only occur in JSON responses.

2.2 Endpoints

Endpoints represent the entries that give access to the resources. The endpoints provide support for both JSON and CSV.
JSON responses are documented using objects. Objects are of the form *objectName* : *objectType*, where *objectName* is the name of the object and *objectType* is the data type of the object, e.g. *string*, *int* or *Airport*.JSON responses contain HATEOAS links to other endpoints when applicable.

Some endpoints support request parameters. A request parameter can be used to filter data. Example of request parameters:
`/airports/ATL/carriers/AA/stats?year=2003&month=6`
Here *year* and *month* are request parameters. With a GET request on this endpoint only statistics for the year 2003 and month june will be returned.

Below we define the endpoints with requests and respones. The asterisk (*) indicates that an object or field in a response can be optional.

1. **/airports**
   GET request:
   Returns all airports available in the US.

   Request headers:

   ```
   Accept: application/json
   ```
   or
   ```
   Accept: text/csv
   ```

   Request body:

   ```
   empty
   ```

Response headers:

`Content-Type: application/json`

or

`Content-Type: text/csv`

Responses with content:

- 200 OK
  JSON response body:
  ```
  content: array[Object]
      airport: Airport
      links: array[Link]
  links: array[Link]
  ```
  CSV response body:
  ```
  airportCode,airportName
  string,string
  ```
- 500 Internal Server Error
  `Something went wrong!`

2. **/carriers**
   GET request:
   Returns all carriers operating in US airports.

   Request headers:

   `Accept: application/json`

   or

   `Accept: text/csv`

   Request body:

   `empty`

   Response headers:

   `Content-Type: application/json`

   or

   `Content-Type: text/csv`

   Responses with content:

   - 200 OK
     JSON response body:
     `content: array[Carrier]`

     CSV response body:

```
airportCode,airportName
string,string
```

- 500 Internal Server Error

  `Something went wrong!`

3. **/airports/{airport_code}/carriers**
   GET request:
   Returns all carriers operating at a specific US airport.

   Endpoint path parameters:

   ```
   airport_code    The 3-letter code of this airport.
   ```

   Request headers:

   `Accept: application/json`

   or

   `Accept: text/csv`

   Request body:

   `empty`

   Response headers:

   `Content-Type: application/json`

   or

   `Content-Type: text/csv`

   Responses with content:

   - 200 OK
     JSON response body:

     ```
     content: array[Object]
         carrier: Carrier
         links: array[Link]
     ```

     CSV response body:

     ```
     carrierCode,carrierName
     string,string
     ```

   - 404 Not Found

     `Airport with code {airport_code} does not exist!`

   - 500 Internal Server Error

     `Something went wrong!`

4. **/airports/carriers/stats**
   POST request:
   Adds statistics.

   Request headers:

```
Content-Type: application/json
```

or

```
Content-Type: text/csv
```

Request body:

JSON request body:

```
airport: Airport
carrier: Carrier
year: int
month: int
flightData: FlightData
delayData: DelayData
delayTimeData: DelayTimeData
```

CSV request body:

```
airportCode,carrierCode,year,month,cancelledCount,onTimeCout,delayedCount,divertedCount,
totalCount,lateAircraftCount,carrierCount,weatherCount,securityCount,
nationalAviationSystemCount,lateAircraftTime,carrierTime,weatherTime,securityTime,
nationalAviationSystemTime,totalTime
string,string,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int
```

Response headers:

```
empty
```

Responses with content:

- 200 OK
  `empty`
- 400 Bad Request
  `Syntax error in JSON string.`
  or
  `Syntax error in CSV string.`
- 500 Internal Server Error
  `Something went wrong!`

5. **/airports/{airport_code}/carriers/{carrier_code}/stats**

   5.1. **GET** request:
   Returns statistics about flights of a carrier from/to an US airport for an optional year and or optional month.

   Endpoint path parameters:

   ```
   airport_code    The 3-letter code of the airport.
   carrier_code    The 3-letter code of the carrier.
   ```

   Request parameters:

`year: int`	`The year for which data should be returned. Is optional.`
`month: int`	`The month for which data should be returned. Is optional.`

Request headers:

`Accept: application/json`
or
`Accept: text/csv`

Request body:

`empty`

Response headers:

`Content-Type: application/json`

`Content-Type: text/csv`

Responses with content:

- 200 OK
  JSON response body:
  ```
  content: array[Object]
      year*: int
      month*: int
      flightData: FlightData
      delayData: DelayData
      delayTimeData: DelayTimeData
  links: array[Link]
  ```
  CSV response body:
  ```
  year*,month*,cancelledCount,onTimeCout,delayedCount,divertedCount,totalCount,
  lateAircraftCount,carrierCount,weatherCount,securityCount,
  nationalAviationSystemCount,lateAircraftTime,carrierTime,weatherTime,
  securityTime,nationalAviationSystemTime,totalTime
  int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int
  ```
- 400 Bad Request
  `Year can not be smaller than 1901.`
  or
  `Year can not be greater than the current year.`
  or
  `Month can not be smaller than 1.`
  or
  `Month can not be greater than 12.`
  or
  `Month must be smaller than the current month.`
- 404 Not Found
  `Airport with code {airport_code} does not exist!`
  or
  `Carrier with code {carrier_code} does not exist!`

- 500 Internal Server Error
  `Something went wrong!`

5.2. **PUT** request:
Updates statistics, or adds statistics if they don't yet exist, about flights of a carrier from/to an US airport for an optional year and or optional month.

Endpoint path parameters:

`airport_code`	`The 3-letter code of the airport.`
`carrier_code`	`The 3-letter code of the carrier.`

Request parameters:

`year: int`	`The year for which data should be updated/added. Is optional.`
`month: int`	`The month for which data should be updated/added. Is optional.`

Request headers:

`Content-Type: application/json`
or
`Content-Type: text/csv`

Request body:

```
year*: int
month*: int
flightData: FlightData
delayData: DelayData
delayTimeData: DelayTimeData
```

CSV request body:

```
year*,month*,cancelledCount,onTimeCout,delayedCount,divertedCount,
totalCount,lateAircraftCount,carrierCount,weatherCount,
securityCount,nationalAviationSystemCount,lateAircraftTime,
carrierTime,weatherTime,securityTime,nationalAviationSystemTime,
totalTime
int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,int,
int
```

Response headers:

`empty`

Responses with content:

- 200 OK
  `empty`
- 400 Bad Request
  `Year can not be smaller than 1901.`
  or
  `Year can not be greater than the current year.`
  or

`Month can not be smaller than 1.`
or
`Month can not be greater than 12.`
or
`Month must be smaller than the current month.`
- 404 Not Found
  `Airport with code {airport_code} does not exist!`
  or
  `Carrier with code {carrier_code} does not exist!`
- 500 Internal Server Error
  `Something went wrong!`

5.3. **DELETE** request:
Deletes statistics about flights of a carrier from/to an US airport for an optional year and or optional month.

Endpoint path parameters:

`airport_code`	`The 3-letter code of the airport.`
`carrier_code`	`The 3-letter code of the carrier.`

Request parameters:

`year: int`	`The year for which data should be deleted. Is optional.`
`month: int`	`The month for which data should be deleted. Is optional.`

Request headers:
`empty`

Request body:
`empty`

Response headers:
`empty`

Responses with content:
- 200 OK
  `empty`
- 400 Bad Request
  `Year can not be smaller than 1901.`
  or
  `Year can not be greater than the current year.`
  or
  `Month can not be smaller than 1.`
  or
  `Month can not be greater than 12.`
  or
  `Month must be smaller than the current month.`
- 404 Not Found

  `Airport with code {airport_code} does not exist!`
  or
  `Carrier with code {carrier_code} does not exist!`
- 500 Internal Server Error
  `Something went wrong!`

6. **/airports/{airport_code}/carriers/{carrier_code}/stats/flight**
   GET request:
   Returns the number of on-time, delayed, and cancelled flights of a carrier from/to an US airport for an optional year and or optional month.

   Endpoint path parameters:

`airport_code`	`The 3-letter code of the airport.`
`carrier_code`	`The 3-letter code of the carrier.`

   Request parameters:

`year: int`	`The year for which data should be deleted. Is optional.`
`month: int`	`The month for which data should be deleted. Is optional.`

   Request headers:

   `Accept: application/json`

   or

   `Accept: text/csv`

   Request body:

   `empty`

   Response headers:

   `Content-Type: application/json`

   or

   `Content-Type: text/csv`

   Responses with content:

   - 200 OK
     JSON response body:
     ```
     content: array[Object]
         year*: int
         month*: int
         flightData: FlightData
             cancelledCount: int
             onTimeCount: int
             delayedCount: int
     ```
     CSV response body:

```
year*,month*,cancelledCount,onTimeCount,delayedCount
int,int,int,int,int
```

- 400 Bad Request

  `Year can not be smaller than 1901.`

  or

  `Year can not be greater than the current year.`

  or

  `Month can not be smaller than 1.`

  or

  `Month can not be greater than 12.`

  or

  `Month must be smaller than the current month.`
- 404 Not Found

  `Airport with code {airport_code} does not exist!`

  or

  `Carrier with code {carrier_code} does not exist!`
- 500 Internal Server Error

  `Something went wrong!`

7. **/airports/carriers/stats/delay-time**
   GET request:
   Returns the number of minutes of delay per carrier attributed to carrier-specific reasons for all US airports and for an optional year and or optional month.

   Request parameters:

   ```
   type: array[string] The delay types for which the minutes of delay should be returned.
                       Accepted values are: late-aircraft, carrier, weather, security,
                       national-aviation-system, total.
   year: int       The year for which data should be deleted. Is optional.
   month: int      The month for which data should be deleted. Is optional.
   ```

   Request headers:

   `Accept: application/json`

   or

   `Accept: text/csv`

   Request body:

   `empty`

   Response headers:

   `Content-Type: application/json`

   or

   `Content-Type: text/csv`

Responses with content:

- 200 OK
  JSON response body:
  ```
  content: array[Object]
      airport: Airport
      carrier: Carrier
      year*: int
      month*: int
      delayTimeData: DelayTimeData
          lateAircraftTime*: int
          carrierTime*: int
          weatherTime*: int
          securityTime*: int
          nationalAviationSystemTime*: int
          totalTime*: int
  ```
  CSV response body:
  ```
  airportCode,airportName,carrierCode,carrierName,year*,month*,lateAircraftTime*,
  carrierTime*,weatherTime*,securityTime*,nationalAviationSystemTime*,totalTime*
  string,string,string,string,int,int,int,int,int,int,int,int
  ```
- 400 Bad Request
  `Year can not be smaller than 1901.`
  or
  `Year can not be greater than the current year.`
  or
  `Month can not be smaller than 1.`
  or
  `Month can not be greater than 12.`
  or
  `Month must be smaller than the current month.`
- 500 Internal Server Error
  `Something went wrong!`

8. **/airports/{airport_code}/carriers/stats/delay-time**
   GET request:
   Returns the number of minutes of delay per carrier attributed to carrier-specific reasons for a specific airport and for an optional year and or optional month.

   Endpoint path parameters:

`airport_code`	`The 3-letter code of the airport.`

   Request parameters:

`type: array[string]`	`The delay types for which the minutes of delay should be returned. Accepted values are: late-aircraft, carrier, weather, security, national-aviation-system, total.`
`year: int`	`The year for which data should be deleted. Is optional.`
`month: int`	`The month for which data should be deleted. Is optional.`

Request headers:

```
Accept: application/json
```

or

```
Accept: text/csv
```

Request body:

```
empty
```

Response headers:

```
Content-Type: application/json
```

or

```
Content-Type: text/csv
```

Responses with content:

- 200 OK
  JSON response body:
  ```
  content: array[Object]
      carrier: Carrier
      year*: int
      month*: int
      delayTimeData: DelayTimeData
          lateAircraftTime*: int
          carrierTime*: int
          weatherTime*: int
          securityTime*: int
          nationalAviationSystemTime*: int
          totalTime*: int
  ```
  CSV response body:
  ```
  carrierCode,carrierName,year*,month*,lateAircraftTime*,
  carrierTime*,weatherTime*,securityTime*,nationalAviationSystemTime*,totalTime*
  string,string,int,int,int,int,int,int,int,int
  ```
- 400 Bad Request
  `Year can not be smaller than 1901.`
  or
  `Year can not be greater than the current year.`
  or
  `Month can not be smaller than 1.`
  or
  `Month can not be greater than 12.`
  or
  `Month must be smaller than the current month.`
- 404 Not Found

`Airport with code {airport_code} does not exist!`

- 500 Internal Server Error
  `Something went wrong!`

9. **/airports/{airport_code_1}/{airport_code_2}/carriers/extra-stats/delay-times**
   GET request:
   Returns the descriptive statistics: mean, median, and standard deviation, for carrier-specific delays for a flight between two airports in the USA for all carriers serving this route.

   Endpoint path parameters:

   ```
   airport_code_1    The 3-letter code of the first airport.
   airport_code_2    The 3-letter code of the second airport.
   ```

   Request headers:

   `Accept: application/json`

   or

   `Accept: text/csv`

   Request body:

   `empty`

   Response headers:

   `Content-Type: application/json`

   or

   `Content-Type: text/csv`

   Responses with content:

   - 200 OK
     JSON response body:
     ```
     content: array[Object]
         carrier: Carrier
         lateAircraftTimeMean: int
         lateAircraftTimeMedian: int
         lateAircraftTimeSd: int
         carrierTimeMean: int
         carrierTimeMedian: int
         carrierTimeSd: int
     ```
     CSV response body:
     ```
     carrierCode,carrierName,lateAircraftTimeMean,lateAircraftTimeMedian,
     lateAircraftTimeSd,carrierTimeMean,carrierTimeMedian,carrierTimeSd
     string,string,int,int,int,int,int,int
     ```
   - 404 Not Found
     `Airport with code {airport_code_1} does not exist!`

or

`Airport with code {airport_code_2} does not exist!`

- 500 Internal Server Error

  `Something went wrong!`

10. **/airports/{airport_code_1}/{airport_code_2}/carriers/{carrier_code}/extra-stats/delay-times**
    Returns the descriptive statistics: mean, median, and standard deviation, for carrier-specific delays for a flight between two airports in the USA for a specific carrier that serves this route.

    Endpoint path parameters:

`airport_code_1`	`The 3-letter code of the first airport.`
`airport_code_2`	`The 3-letter code of the second airport.`
`carrier_code`	`The 3-letter code of the carrier.`

    Request headers:

    `Accept: application/json`

    or

    `Accept: text/csv`

    Request body:

    `empty`

    Response headers:

    `Content-Type: application/json`

    or

    `Content-Type: text/csv`

    Responses with content:

    - 200 OK
      JSON response body:

      ```
      content: array[Object]
          lateAircraftTimeMean: int
          lateAircraftTimeMedian: int
          lateAircraftTimeSd: int
          carrierTimeMean: int
          carrierTimeMedian: int
          carrierTimeSd: int
      ```

      CSV response body:

      ```
      lateAircraftTimeMean,lateAircraftTimeMedian,lateAircraftTimeSd,
      carrierTimeMean,carrierTimeMedian,carrierTimeSd
      int,int,int,int,int,int
      ```

    - 404 Not Found

`Airport with code {airport_code_1} does not exist!`

or

`Airport with code {airport_code_2} does not exist!`

or

`Carrier with code {carrier_code} does not exist!`

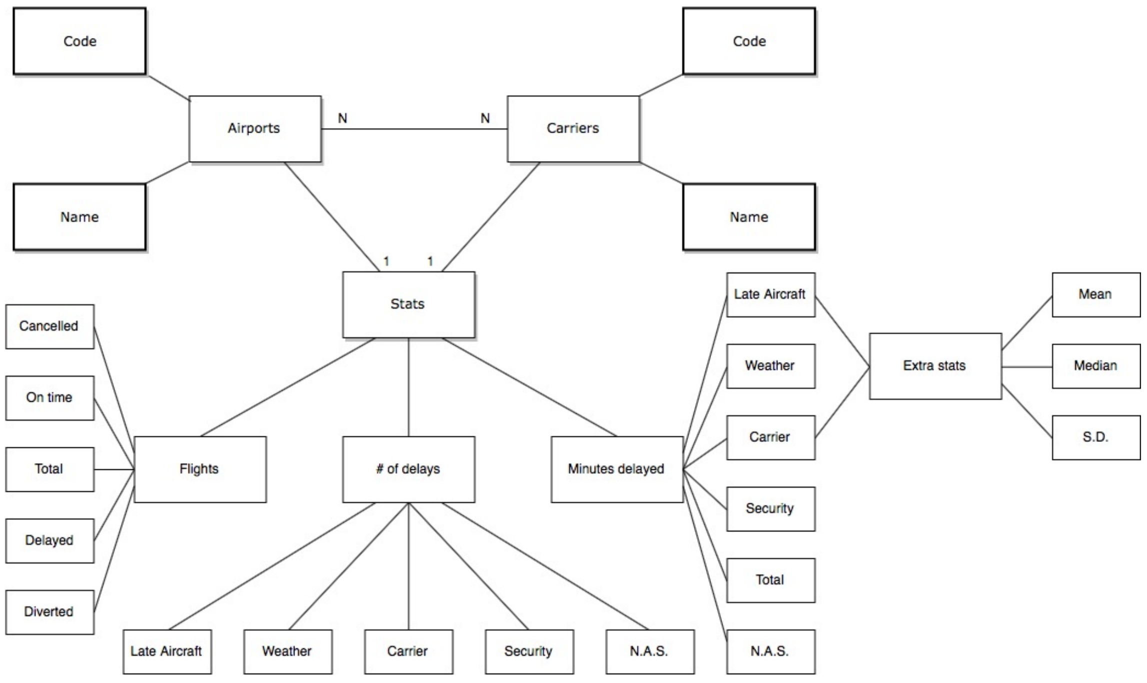
- 500 Internal Server Error

  `Something went wrong!`

2.3 ER Model

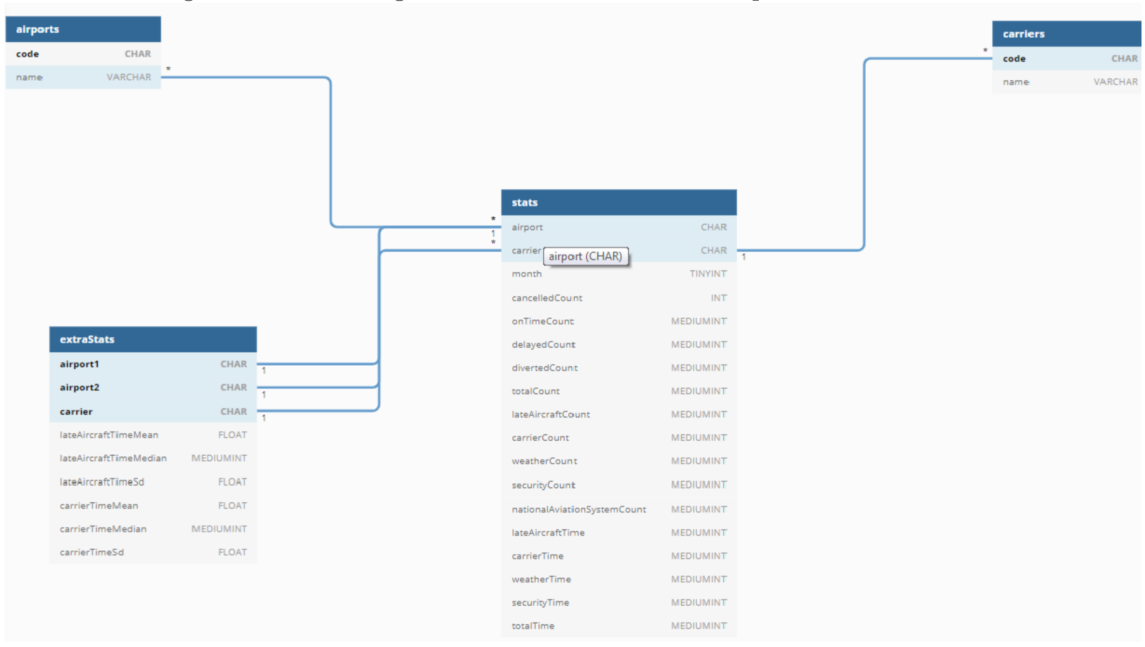
This is an alpha version of our ER model. We identify three main entities in our model: **Airports, Carries and Flights**. The first two entities are pretty straight forward. Each of contains a name and a code as their identifiers. For the third entity things are a bit more complicated. Flights are mainly identified by the flight code. However we also need to keep track of the statistics of all flights. To do that we decided to split the Flight entity into 4 sub-entities. **On-time, Cancelled Delayed and Diverted.** For each of those sub categories we keep track of the statistics. We have a large number of statistics. First of all we need to keep track the reason a flight has been delayed due to different reasons(N.A.S,aircraft,weather,carrier),which are provided to us in both json and csv files. Secondly we also need to keep track of the total time(in minutes) of each of those reason separately and the combined time of all those reasons.

Since this is an Alpha version of our ER model, some of the entities and identifiers might change in the future. However we feel its important to provide a starting point for a visual representation of our thought process.



2.4 Database Diagram

Since we have created tables an connections between those table we decided to also demonstrate the relations through a Database diagram for a more clear visual representations.



3 Milestone 2: Back-end

3.1 Technology Stack

1. **Java**. The back-end is written in Java. We chose Java because we are all familiar with the language.

2. **RESTful API**. Services provide interoperability between computer systems on the Internet. REST-compliant web services allow the requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of stateless operations. The framework we use is called **Spring**.

3. **JSON** (JavaScript Object Notation). JSON is a syntax for storing and exchanging data. JSON is text, written with JavaScript object notation.

4. **CSV** (Comma-seperated Values). A CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas.

5. **Maven**. Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

6. **MySQL** is an open source relational database management system. It was an obvious choice for us as it is easy to use and can manage large quantities of data.

3.2 Code Structure

The back-end code is divided into four parts (packages): models, converters, database, rest, and tools.

1. **Models**

   The `models` package contains classes for modelling the resources. There are the following models:

   `Airport, Carrier, FlightData, DelayData, DelayTimeData, Statistic, ExtraStatistic, Link`

   All of the models, except `Statistic` and `ExtraStatistic`, have already been discussed in section 1.2, so we will only explain `Statistic` and `ExtraStatistic` now.

   `Statistic` represents all statistics (`FlightData`, `DelayData`, and `DelayTimeData`) together with the reported year and month, and the corresponding airport and carrier.

   `ExtraStatistic` represents the more detailed statistics: mean, median, and standard deviation, for carrier-specific delays. `ExtraStatistic` has two Airport fields since the detailed statistics are computed for flights between two airports. It also contains the carrier of which the data has been acquired. The detailed statistics are computed from all data taken together, therefore `ExtraStatistic` does not contain a year and month.

2. **Converters**

   Since we need the ability to create and exhange data in both JSON and CSV formats we created two java classes, JsonConverter and CSVconverter. Each of those is able to handle serialization and deserialization of their corresponding formats. In the case of JSON we decided to use Googles **Gson** library as it is reliable and easy to use, since it uses reflection so it does not require additional modifications to classes of deserialized or deserialized objects. For the CSVconverter we use the Apache Commons CSV library, because it was the only CSV library we could found, that would do CSV conversion without the need of CSV files.

   Both classes implement the interface `DataConverter`. Apart from the default interface methods, the `JsonConverter` also contains some method to merge hypermedia links with data.

3. **Rest**

   This is the driver application of our project. From here we initiate all the functionality of our code by using the spring framework and libraries. For our application to navigate through all the data and parameters we made use of URI and HTTP. In more detail:

   3.1. **Application.java**. Is the main driver function of the back-end. Executing it will start the application.

   3.2. **MainRestController.java**. Here we map the endpoints to Java methods. Once a HTTP request comes in, the mapped method is called. Next data is retrieved, added, updated, or deleted from the database and a response is send to the client. The endpoints can be found on section 2.3 of the report. In order to achieve the desired result we made use of Spring framework libraries. **HttpHeaders** is a data structure representing HTTP request or response headers, mapping String header names to a list of String values. **HttpStatus**is used for enumeration of HTTP status codes. **GetMapping** is annotation for mapping HTTP GET requests onto specific handler methods. There are also **PostMapping**, **PutMapping**, and **DeleteMapping** annotations. **PathVariable** make it possible to have annotation which indicates that a method parameter should be bound to a URI template variable. Last but not least **RequestParam** indicates that a method parameter should be bound to a web request parameter.

4. **Database**

   Lastly the `DatabaseConnector` class is used for connecting to the MySQL database. It can be used to create the appropriate tables, insert or delete data as well as getting the data we require on demand. To achieve the we make use of SQL statements and JDBC. Java Database Connectivity (JDBC) is an API for Java, which allows a client to access a database.

   The database has four tables: airports, carriers, stats, and extraStats. The airports table has two columns, one for the code and one for the name. The carriers table has the same columns as the airports table. The stats table contains the twelve different statistics, the airport, the carrier, the year, and the month. The extraStats table serves for holding the additional statistic data (mean, median and standard deviation). It has a carrier column, two airports columns and six columns for the statistics data. The tables created are compliant with the CSV tables we have received for this project. The `DatabaseConnector` starts by requesting the credentials of the user in order to connect to the database. Once it's connected it will created the four tables if they don't yet exist. After that the `DatabaseConnector` is ready to execute SQL statements.

4 Milestone 3: Web application

4.1 Introductions

In designing the front end, we decided to also use a framework called Angular6, since it would make it easy to connect software components. Furthermore Angular provides client-side model view controlers and model-view-viewmodel architectures, which makes for easy testing. It is important to note hear that our project will work for any Angular version after version 6.0. Of course on top of Angular6 we also made use of Angular CLI, as it make it easy to create and delete files and components of models that we created.

Another component of the front-end is Node JS. Node is easy to use and is designed to build scalable network applications. Furthermore Node can support many connections at the same time. Upon each connection the callback is fired, but if there is no work to be done, Node will sleep

4.2 Code Structure

For each of the models that exist in the back end we created files with the appropriate names, for easy association. Each of these files consists of 4 componenets that help communicate information to a web application and keep track of the airport data as visual representation. Those four components are ***.component.css**, ***.component.spec.ts**, ***.component.ts** , ***.component.html**. These files are created by using Angular CLi commands. In the following section we will go into details on what each of these files do how they are structured. Since all of the models contain the same type of structure, we will use the aiport model as an example to analyze the functions.

1. **airport.component.ts** : Components are the fundamental blocks of Angular applications. They display data on the screen, listen for user input, and take action based on that input.

After the components have been defined we specify which class we want to export and implementing hooks to get that data. This is done by the function **ngOnInit**. Of course we also need to import the appropriate routing and restservice to do that.

2. **airport.component.spec.ts** : The ***.spec** files are unit tests for your source files. The convention for Angular applications is to have a .spec.ts file for each .ts file. They are run using the Jasmine javascript test framework through the Karma task runner when you use the **ng test** command.

3. **airport.component.css** : Here we create web pages to help keep information in the proper display format. CSS files can help define font, size, color, spacing, border and location of HTML information on a web page.

4. **airport.component.html** : In this file we specify the actual structure of the web page.

Furthermore there are some additional files outside the structure names above namely **app.module.ts** and **rest.service.ts**. On **app.module.ts** we describe the paths, so that our application finds the appropriate data , and also declaring all the modules that need to exist in order to follows these paths.The **@NgModule** decorator identifies AppModule as an Angular module class and tells Angular how to compile and launch the application. On the **rest.service.ts** the **@Injectable()** decorator accepts a metadata object for the service, the same way the **@Component()** decorator did for your component classes. Also here is where we include all the methods for get,update and delete data functions.