Web-ENG

Zamir Amiri s2780542
Marco Zanghieri S3838676
Arturo Lopez-Damas s3839451

March 2019

1 Introduction

This is the documentation of the web app called WebAPI. This web app was created for the course web engineering. This web app is supposed to heal users get information about airports, carriers and the flights of carrier airplanes on a specific airport. The project was split in three parts, M1 API-design, M2 back-end implementation and M3 front-end implementation. In this report we will talk about all three of these parts in the following way:

- Introduction
- API
- API-Design decisions
- Technology used
- Databse
- Back-end
- Front-end
- Member contribution
- Conclusion

The project can be found at `www.github.com/ZamirAmiri/web-eng`. The branch master contains the final code of the project. The web app should be able to deliver the following data:

R1 all airports available in the USA,

R2 all carriers operating in US airports,

R3 all carriers operating at a specific US airport,

R4 all statistics about flights of a carrier from/to a US airport for a given month/all months available (‡),

R5 number of on-time, delayed, and cancelled flights of a carrier Update Feb 26: from/to a US airport for a given month/all months available,

R6 number of minutes of delay per carrier attributed to carrier-specific reasons (i.e. attributes carrier and late aircraft in the dataset)/all reasons, for a given month/all months available and for a specific airport/across all US airports,

R7 descriptive statistics (mean, median, standard deviation) for carrier-specific delays (as above) for a flight between any two airports in the USA for a specific carrier/all carriers serving this route.

Entries entries marked with (‡) require support for both retrieval and manipulation (addition, update, deletion) of data through the API; otherwise only retrieval is to be supported.

2 API

Our API-design when first created, was created without us having to think about the actual implementation. Once we started actually coding the back-end, our API design changed. The new version of the API design is what you can find below. All data is sent back to the user in a JSON format.

R1

A list of all the available USA airports.

- URL
  /airports
- Method
  GET
- URL Parameters
  NONE
- Data Parameters
  NONE
- Header Field:
  Content-Type: application/json or text/csv
- Success Response
  CODE: 200
  The request is successfully received, understood, and accepted.
  Content: JSON

```
airports : {
    {
        "code": String
        "name": String
    },
    {
        "code": String
        "name": String
    },
..}
```

- Error Response
  CODE: 400
  The request contains bad syntax.
  CODE: 404
  The information you have requested is currently unavailable.

R2

Returns a list of all the operating carriers US airports.

- URL
  /carriers
- Method
  GET
- URL Parameters
  NONE
- Data Parameters
  NONE
- Header Field:
  Content-Type: application/json or text/csv
- Success Response
  CODE: 200
  The request is successfully received, understood, and accepted.
  Content: JSON

```
"carriers" : {
    {
        "code": String
        "name": String
    },
```

```
        {
            "code": String
            "name": String
        },
    ..}
```

- Error Response
  CODE: 400
  The request contains bad syntax.
  CODE: 404
  The information you have requested is currently unavailable.

R3

Returns a list of all the operating carriers in (a specific) US airport(s).

- URL
  airport/carriers/{airportCode}
- Method
  GET
- URL Parameters
  Char [3] airportCode: code of the airport
- Data Parameters
  None
- Header Field:
  Content-Type: application/json or text/csv
- Success Response
  CODE: 200
  The request is successfully received, understood, and accepted.
  Content: As JSON

```
{
    "airport" : {
        "code": String
        "name": String
     }

    "carriers" :
    {{
        "code": String
        "name": String
     },
```

```
        {
            "code": String
            "name": String
        },
        ..}
}
```

- Error Response
  CODE: 400
  The request contains bad syntax.
  CODE: 404
  The information you have requested is currently unavailable.

- Sample Call
  Request to get all carriers of airport with code "ATL".
  GET airports/carriers/ATL",

R4

Get, delete and update all statistics about flights of a carrier from/to a US airport for a given month/all months available

TODO: MARCO

R5

Number of on-time, delayed, and cancelled flights of a carrier from/to a US airport for a given month/all months available

- URL
  /airports/carriers/{airportcode}/{carriercode}/date/{month}/{year}

- Method
  GET

- URL Parameters
  Char[2] carrierCode : Code of the carrier
  Char[3] airportCode : Code of the airport
  String month : month (in numbers) chosen by user
  String year : year chosen by user

- Data Parameters
  None

- Header Field:
  Content-Type: application/json or text/csv

- Success Response
  CODE: 200

```
{
    'total_cancelled':   int,
    'total_delayed':     int,
    'total_on_time':     int,
    flight:              {
        on_time: int,
        delayed: int,
        cancelled: int
    }
}
```

- Error Response
  CODE: 400
  The request contains bad syntax.
  CODE: 404
  The information you have requested is currently unavailable.

- Sample Call
  /airports/carriers/ATL/AA/date/1/2016/

R6

Number of minutes of delay per carrier attributed to carrier-specific reasons/all reasons, for a given month/all months available and for a specific airport/across all US airports.

- URL
  /airports/carriers/{airportcode}/{carriercode}/date/{month}/{year}

- Method
  GET

- URL Parameters
  Char[3] airportcode : airport code
  Char[2] carriercode : carrier code
  Integer month : Value in the interval [1,12].
  Integer year : any 4 digit value between now and the year 0.

- Data Parameters
  None

- Header Field:
  Content-Type: application/json or text/csv

- Success Response
  CODE: 200

```
{
    "cdasm": {
        carrier: int,
        late_aircraft: int
    },
    "total_cdasc":  int,
    "total_cdasla": int,
    "total_cdas_weather": int,
    "total_cdas_security": int,
    "total_cdas_nas": int,
    "total_cdas_total": int,
    "total_md_la": int,
    "total_md_cd": int,
    "total_md_weather": int,
    "total_md_security": int,
    "total_md_nas": int,
    "total_md_total": int,
    "md_cd": int,
    "md_la": int,
    "md_weather": int,
    "md_security": int,
    "md_nas": int,
    "md_total": int
}
```

  cdasm stands for Carrier minutes Delay Airport Specific Monthly.
  total_cdas* contains the sum of all attributes of "cdasm" for all months available.
  md_* are the attributes of minutes delayed of a carrier for all airports on a specific date
  total_md_* the sum of each attribute of md_* for all months available

- Error Response
  CODE: 400
  The request contains bad syntax.
  CODE: 404
  The information you have requested is currently unavailable.

- Sample Call
  /airports/carriers/ATL/AA/date/1/2016/

R7

Descriptive statistics (mean, median, standard deviation) for carrier-specific delays for a flight between any two airports in the USA for a specific carrier/all carriers serving this route.

- URL
  /airports/carriers/{airportcode}/
- Method
  GET
- URL Params
  Char[3] airportcode : code of the airport
- Data Params
  None
- Success Response
  CODE: 200

```
As JSON
{
    "carriers": [
        {
            "carrier": carrierObject,
            "avg_la": int,
            "avg_cd": int,
            "median_cd": int,
            "median_la": int,
            "std_la": int,
            "std_cd": int
        }
    ],
    "avg_la": int,
    "avg_cd": int,
    "median_cd": int,
    "median_la": int,
    "std_la": int,
    "std_cd": int
}
```

  Note: The objects inside the objects array

- Error Response
  CODE: 400
  The request contains bad syntax.

CODE: 404
The information you have requested is currently unavailable.

- Sample Call
  /airports/carriers/ATL

3 API-Design decisions

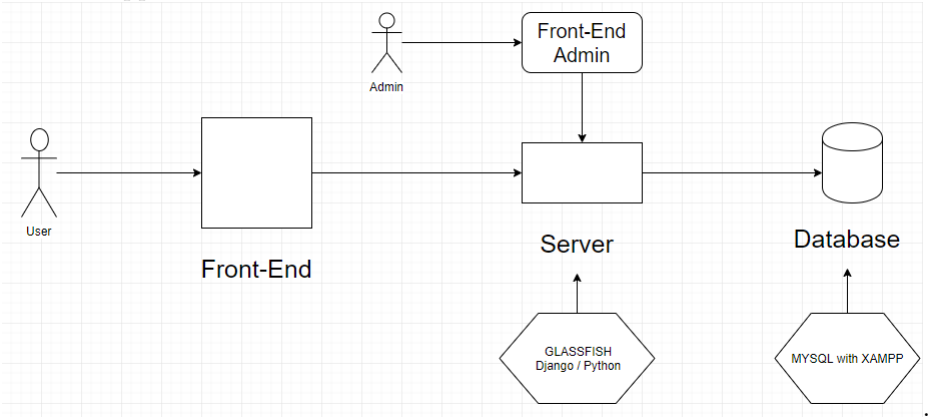
In these section we will explain why each ULR looks the way it does. First of all to print a list of all the carriers, we use the ULR: "domain:port/carriers/". Since it is our job to print all the carriers no matter what, it is obvious that we do not need any parameters to clarify anything. The same reason also implies to printing all the airports at "domain/airports/".
For **R3** we use "domain/airports/carriers/{airportcode}. Our job is to print all carriers which work at a certain airport. This would be equal to the following SQL query:

```
"SELECT * from Carriers WHERE code IN
(SELECT carrierCode FROM airportCarrier WHERE airport = \{airportcode\})
```

We can clearly see that only one value is required to get this data. Hence we only ask for airportcode.
For **R5** we use "/airports/carriers/airportcode/carriercode/date/month/year" because each Flight object in our database can be identified with the airportcode, carriercode and the date. This is also true for every other statistical item (e.g. minutes delay, number of delays). Because of that we use for both **R5** and **R6** the same URL. This made sense to us because both **R5** and **R6** were requesting data about the same instance of the same object. And so now this page contains all the "important" data of this object for the specific date. For **R7** we use "/airports/carriers/airportcode/". The reason for that is that we realized that it would be nice for the user to be able to see the mean, median and standard deviation (std) of carrier specific delay for a certain airport right at the page where they can select a carrier. The idea was to let the user also order the list of carriers by minimum average "late aircraft". This way the data becomes less tedious for the user, and it could help users to make their carrier choice easier. We also print the average, median and std of all the carriers working on this airport on this page. We later realized that it would have been nicer to print in on the "domain/airports" page. Because then the user would have been able to order those as well. Sadly because of some time issues, we did not change this.

4 Technology used

At the start of the project, we had a difference in opinion on what language and frame work we should use for the server. Zamir wanted to use JAVA with a java framework for HTTP requests, while Arturo and Marco wanted to use Python with Django framework. The majority won and we chose Django for the back end. Django is extremely easy to use and beginner friendly (it has tons of online tutorials), in case of java this is different. Since only Zamir knew how to write back end in JAVA, it would have taken Marco and Arturo too much time to use it. Since we had already some time issues it was very important for us to develop fast! And so we can conclude that using Django was indeed the right decision. Django also gave us an admin page which works really nicely. For the back end we use Glassfish as server, this is mainly because the tutorials we followed were using Glassfish. For development we use XAMPP. This is a third party app which allows programmers to view their database in a nice GUI and it has some useful functions.
Our front end was built with HTML, CSS and javascript. Initially we wanted to use React for it, but again because of time and communication issues this did not happen.



5 Database

Using a database was a given to us. Since the JSON file was way to big to search through for data. And so in order to make our web app fast and efficient, we chose to build a database. Our database exists out of six tables (excluding all the default Django tables). Our tables are:

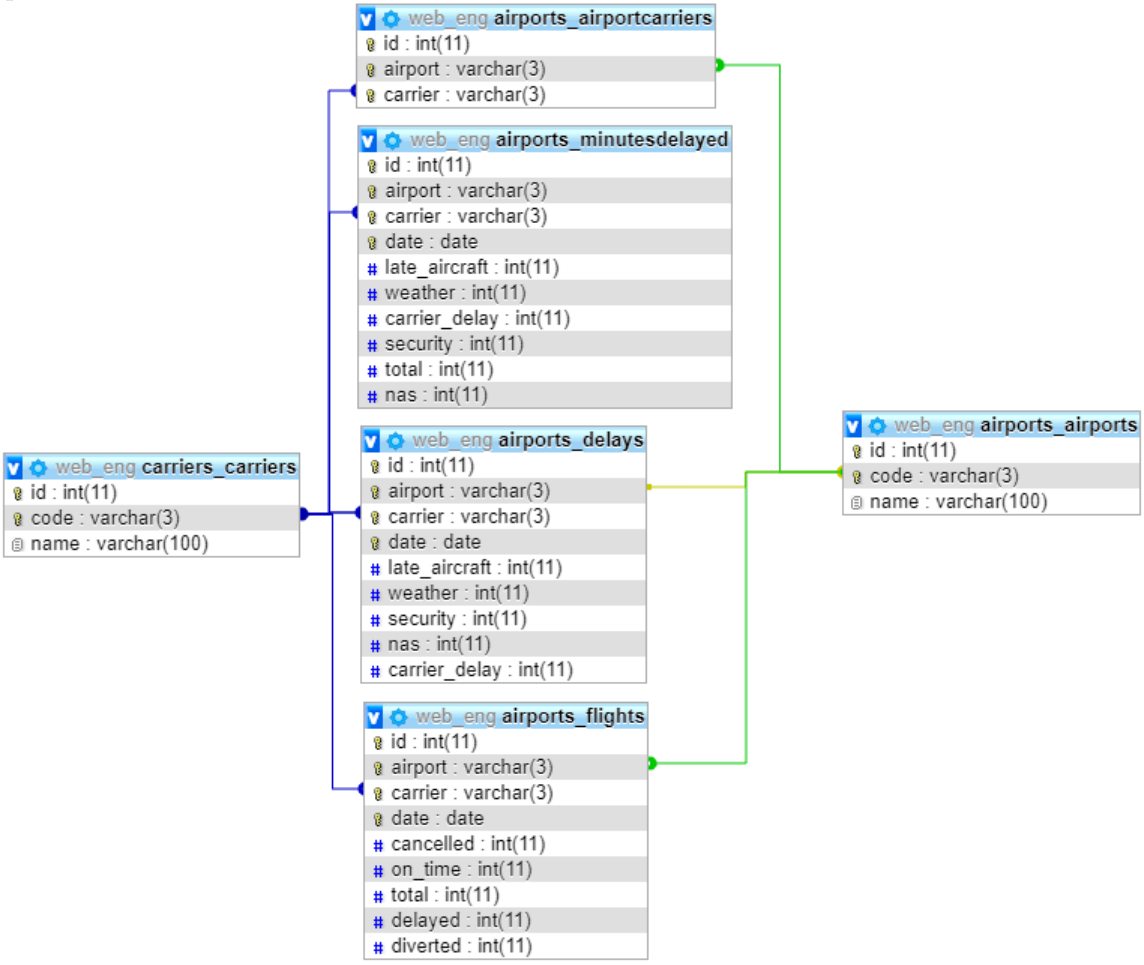
1. **airports_airportcarriers(id,airport,carrier)**
2. **airports_airports(id, code, name)**
3. **carriers_carriers(id, code, name)**

4. **airports_delays(id,airport,carrier,date,late_aircraft, weather, security, nas, carrier_delay)**

5. **airports_minutesdelayed(id,airport,carrier,date,late_aircraft, weather, carrier_delay, security, total, nas)**

6. **airports_flights(id,airport,carrier,date,cancelled, on_time, total, delayed, diverted)**

With this database we can store all the information in the JSON file. To do that we have created a program called **update_database.py**. This file will read the JSON file in the same directory and will ADD/UPDATE the database. So if a new JSON file is released by the service, we can immediately update our database accordingly. We have then used 'phpmyadmin' (xampp function) to add some uniqueness constrains in order to make sure that we do not have duplication's. Overall the relational database looks as followed.



6 Back-end

It is quite hard to explain how the back end works without explaining Django as well. Therefore we will talk about Django a little. When creating a Django project, the program will create some files on it's own. The important files will be inside the folder named after your project, in this case that is WebAPI.

6.1 WebAPI

You have to add your 'apps' to **settings.py**. In our case that is **carriers** and **airports**. Inside **urls.py** we can find the URLs which the python project will try to find if an http request is made. For example if an http request like Get "/localhost:8000/airports/etc" is made, then the program will try to match it with one of the paths added inside urlpatterns in **urls.py**. In this case the program will find *path('airports/', include('airports.urls'))*. It will then split the URL exactly at airports/ and will send the rest of the url (in this 'etc') to **airports.urls**. The rest of the URLs is then handled there. Since our code has two starting URLs namely /airports/ and /carriers/, we require to add these two to the urlpatterns of **urls.py**. With this made clear we can move on to the main part of the code.

6.2 Carriers

Inside the carriers folder you can find everything about each http request that starts with /carriers/. First when WebAPI/urls.py splits the URL, and goes to carriers/URLs.py, it will try to find the right path for the left-over string. Here we have to possibilities. Either the string is empty or we have something like: "airports/{carriercode}". When the URL matches the program will go to the **views.py** and call the corresponding function. The real computation will happen inside views. Each function inside **views.py** returns a rendering of an HTML file which get a JSON object (in python a 'dict') as input and this HTML file is shown to the person who made the request. Inside carries/views we have only two functions, that is index and airports. Index first gets all carrier objects by calling 'Carriers.object.all()' and places it in the context JSON object. Then it renders index.html. If we go to Index.html we find:

```
{% for carrier in carriers %}
  <li><a href="/carriers/airports/{{carrier.code}}">{{ carrier.name }}</li>
{% endfor %}
```

In this part each object inside the carriers array property of the JSON object, is iterated through and each of them is then printed as a list item. This is of course not meant as an end product. This was made by Zamir as a way of showing to his lab partners how the code works such that they would have an easier time to style the HTML. Inside the function airports we are doing pretty much the same thing. But this time we only need one carriers object which can be acquired using *Carriers.object.get*() and we get all the airports where this

carrier works. We do this by first getting all the airport_carriers objects where carrier equals carrier code. Then we append all airport objects to the array **airport** which then is added to the JSON object.

```
carrier = Carriers.objects.get(code = code)
airport_carriers = AirportCarriers.objects.filter(carrier = code)
print("print: " , len(airport_carriers))
airport = []

for item in airport_carriers:
    airport.append(Airports.objects.get(code = item.airport))
```

After that the airports.html is rendered.

6.3 Airports

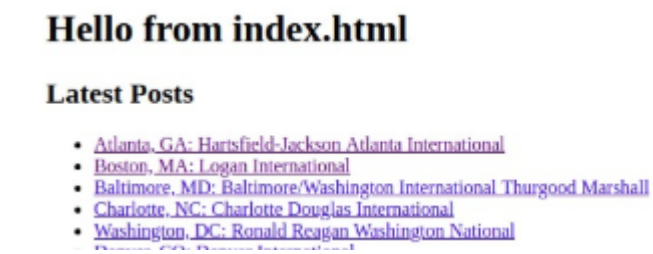
In airports we provide the information to R4,5,6 and 7. Airports.views has a function a called $calcAvgMedStd(delays)$ this function is used to calculate the average,median and standard deviation. It does this for carrier_delay and late_aircraft. By using this function once with all the delays for a specific carrier and once for all carriers on the specific airport, we can display all the information that is needed for $R7$. $R5$ and $R6$ are pretty similar to each other. It seems redundant to explain each line of the code, so instead I will give a small summary of the code. The function for $R5 and R6$ is written in the same function which is called $monthly(request, a\_code, c\_code, month, year)$ with a_code being airport code, c_code being carrier_code, month = month chosen by the user and year = year chosen by the user. We use these parameters to then get all the required data. (remmeber how arport_code,carrier_code and date are together the unique key of the database tables flights,delays and minutes_delayed).

7 Front-end

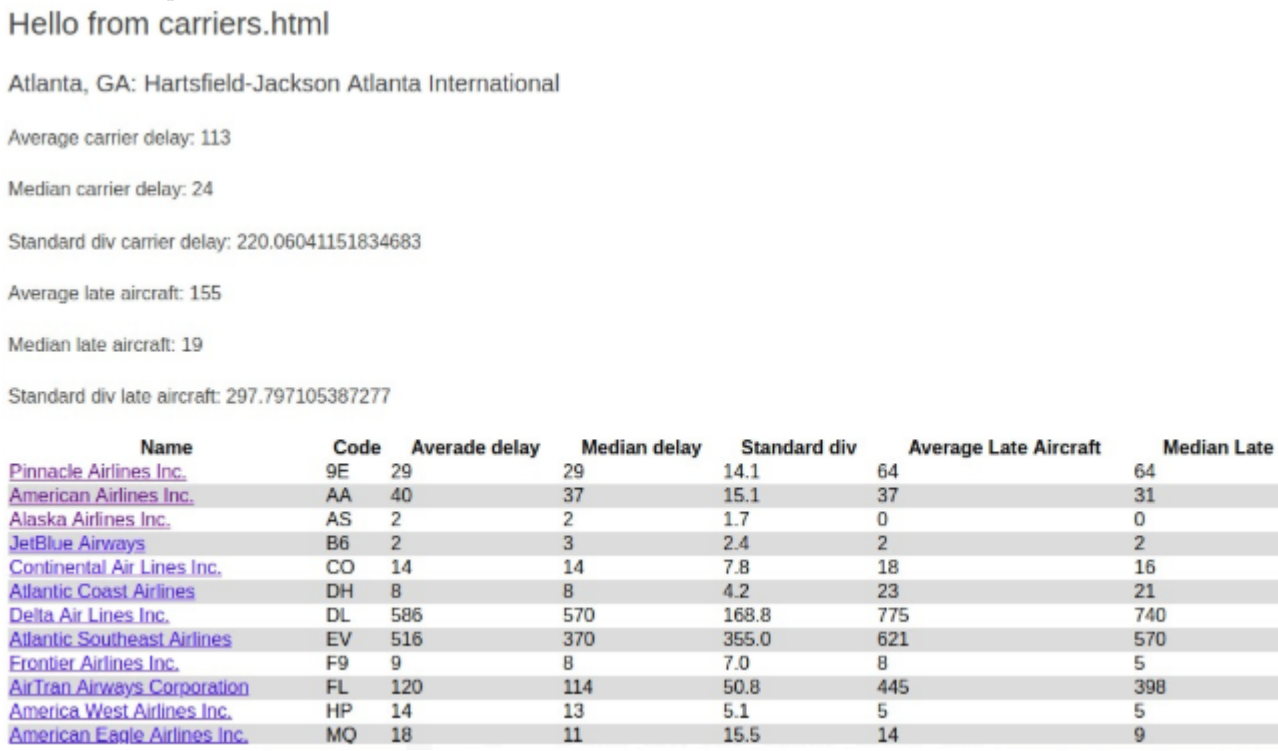
In this section it is going to be discuss the front - end process. Front - end for web development is the practice of converting data to graphical interface for user to view and interact with data. The front end refers to the presentation layer. In the client - server model, the client is usually considered the front end. A user-friendly interface will be provided. Our presentation layer exists out of three different views:

- Airports: Shows all the different airport available in the system.
- Carriers in airports: Shows info(Average carrier delay, Median carrier delay, etc...) about a specific airport and also the info related to all the carriers operating at this airport.
- Statistics: Shows the minutes of delay per carrier attributed to carrier , for a given month available and for a specific airport/across all US airports.

Airports interface



In this interface all the available airports in the US are shown. In this list all the airports names and codes can be found.URL: /airports
Carriers in airports



Name	Code	Averade delay	Median delay	Standard div	Average Late Aircraft	Median Late
Pinnacle Airlines Inc.	9E	29	29	14.1	64	64
American Airlines Inc.	AA	40	37	15.1	37	31
Alaska Airlines Inc.	AS	2	2	1.7	0	0
JetBlue Airways	B6	2	3	2.4	2	2
Continental Air Lines Inc.	CO	14	14	7.8	18	16
Atlantic Coast Airlines	DH	8	8	4.2	23	21
Delta Air Lines Inc.	DL	586	570	168.8	775	740
Atlantic Southeast Airlines	EV	516	370	355.0	621	570
Frontier Airlines Inc.	F9	9	8	7.0	8	5
AirTran Airways Corporation	FL	120	114	50.8	445	398
America West Airlines Inc.	HP	14	13	5.1	5	5
American Eagle Airlines Inc.	MQ	18	11	15.5	14	9

This interface can be split in halves. In the beginning the stats from the airport are shown. To do this different data was introduced as: average carrier delay, median carrier delay, standard div carrier delay, average late aircraft, median late aircraft, standard div lat aircraft. The style of the headers was changed to make it look better.
After it, a table is displayed showing name, code and descriptive statistics for all the carriers in this airport.

On the header of the table all the data names are displayed.

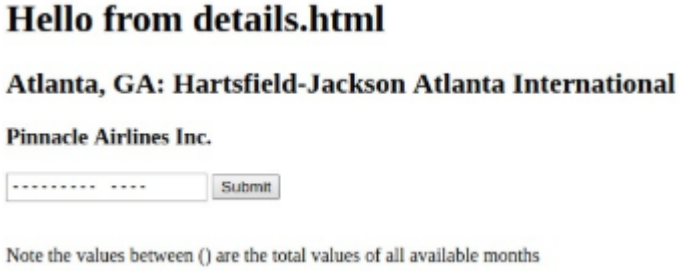
```
{% block content %}
<table id=carriers>
<table>
  <thead>
  <tr><th>Name</th><th>Code</th><th>Averade delay</th><th>Median delay</th><th>Standard div</th><th>Average Late Aircraft</th>
  </thead>
```

On the body, for each carrier all the statistics from this code.

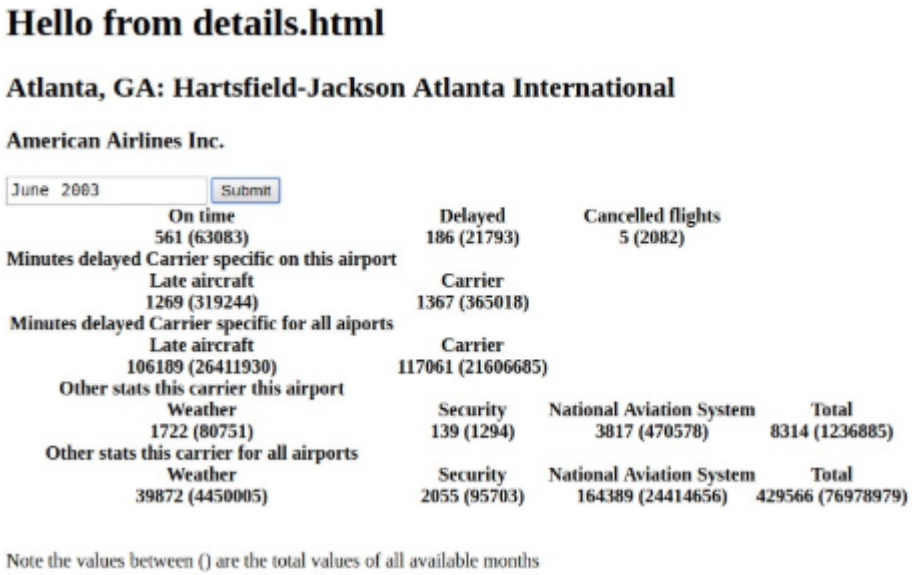
```
  <tbody>
  {% for carrier in carriers %}
      <tr><td><a href="/airports/carriers/{{airport.code}}/{{ carrier.carrier.code }}">
          {{ carrier.carrier.name }}</a></td> <td>{{ carrier.carrier.code }}</td><td> {{ carrier.avg_cd }}</td><td> {{ carrier.me
          <td>{{ carrier.avg_la }}</td><td> {{ carrier.median_la }}</td><td> {{ carrier.std_la }}</td>
      </tr>
  {% endfor %}
</tbody>
</table>
{% endblock %}
```

URL: airports/carriers
Statistics

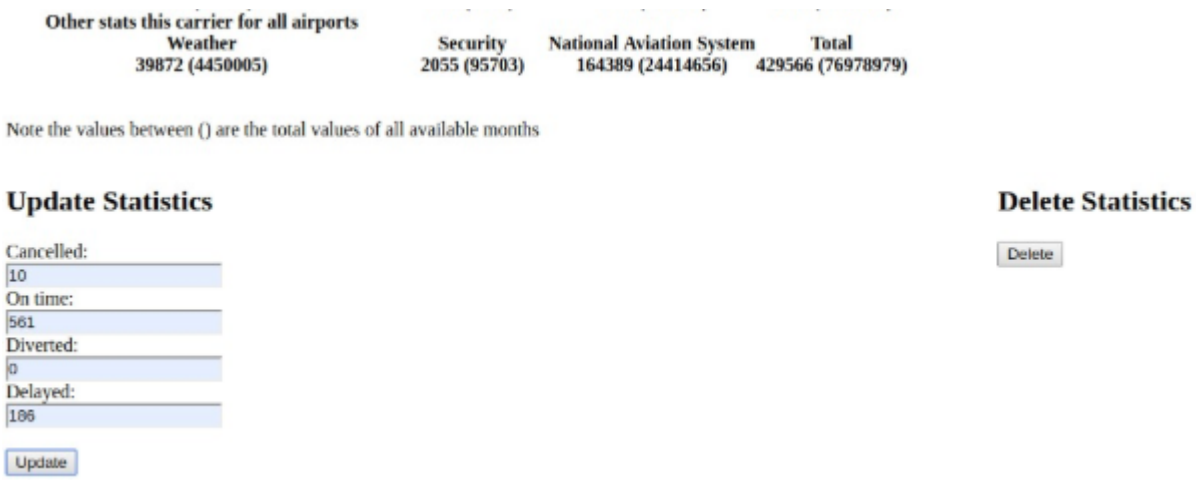
With this interface the user is able to look up for the minutes of delay for an specific carrier and for an airport available in the US in a specific time



The button 'Submit' lets the user choose the month and year of his/her request.
ADD PHOTO SUBMIT BUTTON



Once this has been done, all the data refereeing to the delays its shown, such as (minutes delayed on this/ for all airports, etc...)



There are two functions that the user can do in this interface:

- Update: The information from the carrier can be updated with this function. ADD PHOTO BUTTON UPDATE
- Delete: The information from the carrier can be deleted with this function. ADD PHOTO BUTTON DELETE

8 Member contribution

Here you can find a table with our names and all the tasks. The people who worked on a task have an x marked under their name of a tasks row.

Tasks	Arturo	Marco	Zamir
M1 report		x	x
M1 Applying comments of TA		x	
M2 Requirement 1 -> code		x	x
M2 Requirement 2 -> code		x	x
M2 Requirement 3 -> code		x	x
M2 Requirement 4 -> code		x	
M2 Requirement 5 -> code			x
M2 Requirement 6 -> code			x
M2 Requirement 7 -> code			x
Json -> Database		x	x
Database Design			x
Html delivering data			x
Doc: Intro			x
Doc: API		x	x
Doc: Design decisions			x
Doc: Technology used			x
Doc: Database			x
Doc: Back-end			x
Doc: Front-end	x		
Doc: Member contribution			x
Doc: Conclusion			x
Html design	x	x	x

9 Conclusion

This project was living hell. As you can see our contributions deffer severely and it is hard to blame all of it on miss-communication. I (Zamir) have been under stress because of this project, ever since the second week of this semester. I am not happy at all with the behaviour shown by some group members, and it I was exhausted by the idea that we would be graded together for this project. Luckily i heard that the professor would grade us separately depending on our

contribution, which has given me a glimpse of hope on passing this course. I have experienced that our skill level in coding differs by a large margin as well. Because of that it seemed like I had to solve every problem we had. In the end I wrote around 86% of the code and not 100%.

In terms of the code, it does not look nice. This was mainly because of how are work was distributed.