Rest API Design

Oliver Holder, Nik Dijkema, Tai-Ting Chen (Group 18)

February 2019

1   Introduction

1.1   Framework choice
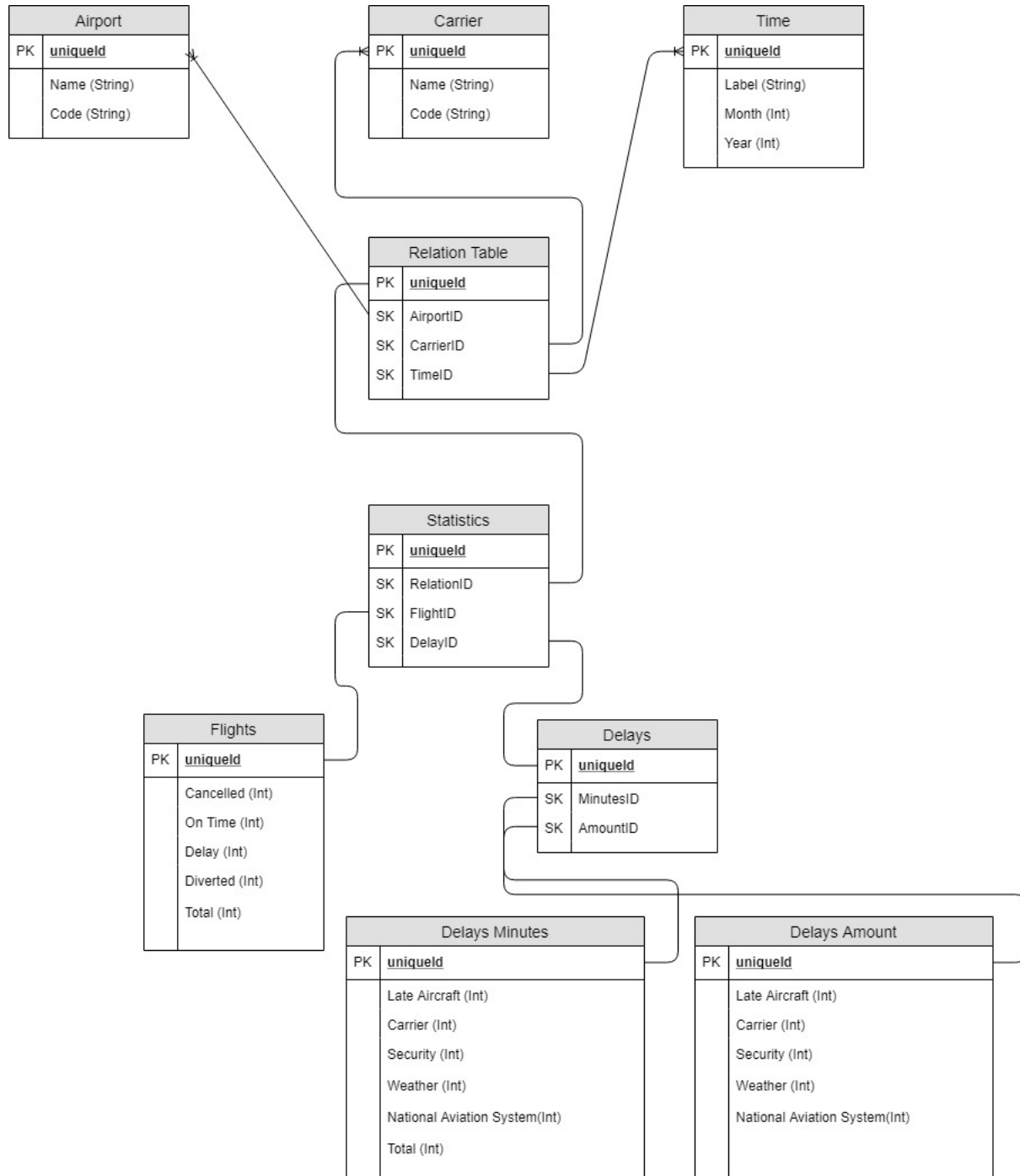
Python will as the main language of the API. The framework flask implements restful design.

1.2   Data storage options

In this project, the initial data we receive is in JSON and CSV format. Either of these formats can be directly parsed and loaded into memory as lists in python. This way of processing data works well for read only data, however, there will be PUSH requests in the API, meaning the data will be subject to change. This causes problems for data loaded in memory, saving the data as text for every PUSH request is very inefficient. Therefore SQL will be used as the main storage method. Because the API will be made in python, the SQL handling will be performed by SQLAlchamy.

2 Data model

2.1 Database structure/relations

Airport

PK	uniqueId
	Name (String)
	Code (String)

Carrier

PK	uniqueId
	Name (String)
	Code (String)

Time

PK	uniqueId
	Label (String)
	Month (Int)
	Year (Int)

Relation Table

PK	uniqueId
SK	AirportID
SK	CarrierID
SK	TimeID

Statistics

PK	uniqueId
SK	RelationID
SK	FlightID
SK	DelayID

Flights

PK	uniqueId
	Cancelled (Int)
	On Time (Int)
	Delay (Int)
	Diverted (Int)
	Total (Int)

Delays

PK	uniqueId
SK	MinutesID
SK	AmountID

Delays Minutes

PK	uniqueId
	Late Aircraft (Int)
	Carrier (Int)
	Security (Int)
	Weather (Int)
	National Aviation System(Int)
	Total (Int)

Delays Amount

PK	uniqueId
	Late Aircraft (Int)
	Carrier (Int)
	Security (Int)
	Weather (Int)
	National Aviation System(Int)

2.2   URI/resource hierachy

The resource and their relations must be defined before an API can be designed. Visible Objects (with URI):

–Airport
–Carrier
    –Statistics
        –Delays
            –Amount
            –Minutes
        –Flights

2.3   URIs

Entity	URI
airports	website/airports/$< code >$
carriers	website/carriers/$< code >$
statistics	website/carriers/$< code >$/statistics
delays(minutes)	website/carriers/$< code >$/statistics/delays/minutes
delays(amount)	website/carriers/$< code >$/statistics/delays/amount
flights	website/carriers/$< code >$/statistics/flights

3   End Points

3.1   Airports

Endpoint	/airports [1]
Type	GET
Return	List of airport URIs

Endpoint	/airports/$< airportcode >$ [3]
Type	GET
return	Airport name + List of carrier URIs related to that airport

3.2   Carriers

3.2.1   /carriers [2]

Type: GET
Return: List of carrier URIs
(- query options: airport)

3.2.2   /carriers/$< carriercode >$

Type: GET
Return: Statistics URI + Carrier Name (string). (- query options: airport)

3.3   Statistics

3.3.1   /carriers/< *carriercode* >/statistics [4]

TYPE: GET
Return: Statistics URI for given carrier returned as delay and flight resources.
(- query options: month, airport)

3.3.2   /carriers/< *carriercode* >/statistics/flights [5]

TYPE: GET
Return: Json Flights information list.
(- query options: month, airport)

3.3.3   /carriers/< *carriercode* >/statistics/delays/minutes [6]

TYPE: GET
Return: Json (minute) delays information list.
(- query options: *delay_type*, month, airport)

3.3.4   airports/< *airportcode* >/carriers/< *carriercode* >/statistics/delays/minutes/averages [7]

(- query options: *delay_type*, airport1, airport2)

4   GET methods summary

- /airports

- /airports/< *airportcode* >

- /carriers?*airport* =< *airportcode* >

- /carriers/< *carriercode* >

- /carriers/< *carriercode* >/statistics
  ?*month* =< *month* >

- /carriers/< *carriercode* >/statistics/flights
  ?*month* =< *month* >

- /carriers/< *carriercode* >/statistics/delays/minutes
  ?*delay_type* =< *type* > &*month* =< *month* >

- /carriers/< *carriercode* >/statistics/delays/minutes/averages
  ?*delay_type* =< *delaytype* > &*airport*1 =< *airportcode* > &*airport*2 =< *airportcode* >

5   PUSH methods summary

- /carriers.< *carriercode* >/statistics

6   Query values

Query variable	type/values
month	Integer from 1-12 indicating january-december
delaytype	$\{carrier, weather, security, national-aviation-system(noa), total\}$
airportcode	String of the airport code.

7 Error codes

The error codes used are inspired by amazons rest API design.

Code	Description
200	Successful request.
201	Created.
400	Bad request (parameter invalid). Will indicate which parameter is invalid and the expected type/format.
404	Page not found.
405	Method invalid.
500	Internal server error (Vague and to be avoided as much as possible).

8 Verification and security

Security for viewing data is unnecessary. (what about push?)
Using HTTPS with a self generated certificate.

9 References

(1) implementation of method 1 from M1 requirements.
(2) implementation of method 2 from M1 requirements.
(3) implementation of method 3 from M1 requirements.
(4) implementation of method 4 from M1 requirements.
(5) implementation of method 5 from M1 requirements.
(6) implementation of method 6 from M1 requirements.
(7) implementation of method 7 from M1 requirements.