

Portfolio Part 2: MongoDB

Aoife Sayers

Advanced Database Programming

Due 10th November 2017

Contents

[Introduction 3](#_Toc497178219)

[1. Implement your own MongoDB database for a possible big data application (showing examples of CRUD) (40%) (JSON document) 4](#_Toc497178220)

[RESTful Interface CRUD - Create collection & insert 4](#_Toc497178221)

[RESTful Interface CRUD – Read 5](#_Toc497178222)

[RESTful Interface CRUD – Update 7](#_Toc497178223)

[RESTful Interface CRUD – Delete 8](#_Toc497178224)

[2. Evaluate and Download a driver for a programming language of your choice. Build a UIF in Java/C# etc and show the CRUD operations working through the UIF. Show a call to a mapreduce operation from within the programming language. Comment your code and show screen pictures of it working. (60%) 9](#_Toc497178225)

[NodeJS 9](#_Toc497178226)

[Node Modules 13](#_Toc497178227)

[Create UIF 14](#_Toc497178228)

[Read UIF 16](#_Toc497178229)

[Update UIF 18](#_Toc497178230)

[Delete UIF 19](#_Toc497178231)

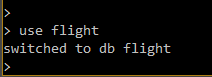
# Introduction

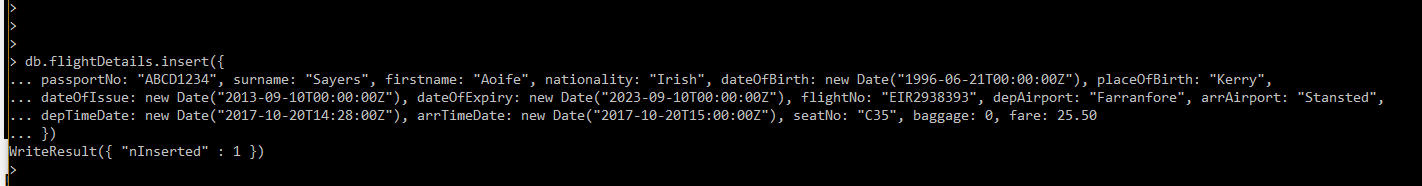
MongoDB is free and open-source NoSQL database particularly good for dealing with Big data. MongoDB stores data in JSON documents (technically BSON), meaning fields can vary from document to document and data structure can be changed over time. MongoDB is a very readable and comprehensible database to learn for a programmer as it’s functions are in a JavaScript format. The flexible schema-less structure allows or faster, iterative development with regards scope creep. MongoDB allows Ad hoc querying and indexing for analyzing data. MongoDB is high availability, has horizontal scaling, and geographic distribution using GridFS are built in and these functions are easy to use. (MongoDB, 2017)

Explain vision behind database design and explain why couchdb/mongo a good fit

# 1. Implement your own MongoDB database for a possible big data application (showing examples of CRUD) (40%) (JSON document)

### RESTful Interface CRUD - Create collection & insert

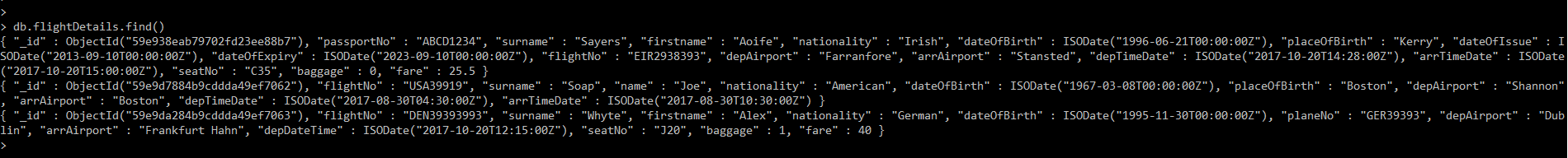


I created a collection called flightDetails and inserted one document via the command line interface. I started the MongoDB server on the path C:\Program Files\MongoDB\Server\3.4\bin>mongod. I opened another CMD prompt on the path C:\Program Files\MongoDB\Server\3.4\bin>mongo to access the CLI. I then issued the command in the figure below to create the database and insert a document to the flightDetails collection. I chose to model a flight booking systems as….

*Figure 1: Creating a database called flightDetails and inserting a document to the flightDetails collection*

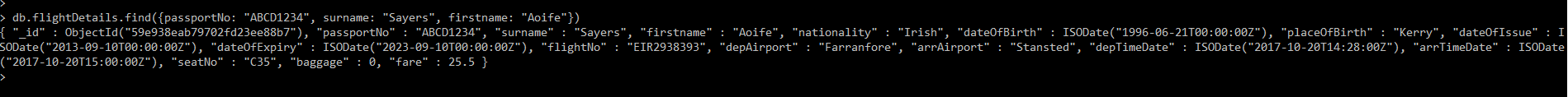
### RESTful Interface CRUD – Read

Finding all documents in the collection using db.flightDetails.find()

I currently have 3 documents in flightDetails

*Figure 2.0: Reading all documents from the flightDetails collection with the find() method with no parameters*

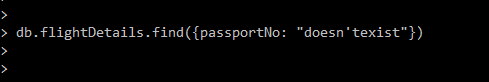
Finding a specific document *db.flightDetails.find({key, value})*

I want to find the document with specific details of passportNo ABCD1234, surname Sayers and first name Aoife

*Figure 2.1: Reading specific documents from the flightDetails collection with the find() method with specified key and value parameters*

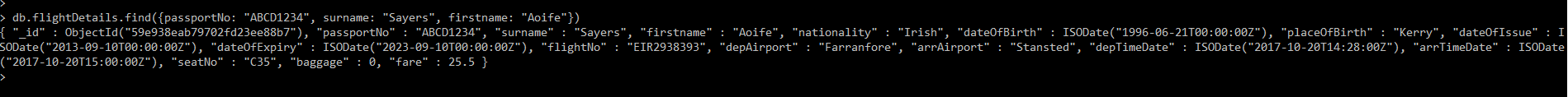
Finding a document with no relevant fields in the collection

As expected it returns no document



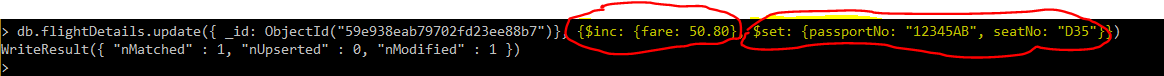
*Figure 2.1: Reading a document with specified key value fields that does not exist in the flightDetails collection*

### RESTful Interface CRUD – Update

Updating details from this

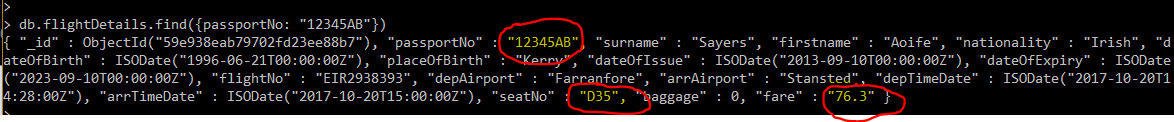
*Figure 3.0: Original details not updated*

Updating using $set & $inc

I updated the details using the db.*collectionName.*update() method passing the ObjectID and $set and $inc fields as the fields to be updated. The $set updates the specified field names of the document and the $inc adds the updated value onto the existing value in the database.

*Figure 3.1: Updating fields using $set and $inc using the specified ObjectID*

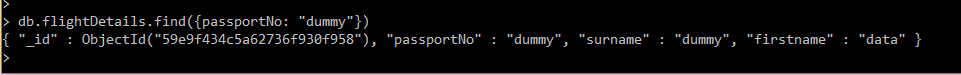
Updated details



*Figure 3.2 Result of updating details*

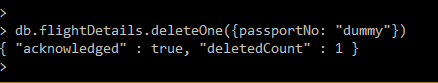
### RESTful Interface CRUD – Delete

Deleting this data



*Figure 4.0: document to be deleted*

Deleting with passportNo dummy



*Figure 4.1: Deleting a document with a passportNo of “dummy”*

Proof

Proof the document with passportNo dummy is deleted



*Figure 4.2: Document with a passportNo of “dummy” no longer exists within the flightDetails collection*

# 2. Evaluate and Download a driver for a programming language of your choice. Build a UIF in Java/C# etc and show the CRUD operations working through the UIF. Show a call to a mapreduce operation from within the programming language. Comment your code and show screen pictures of it working. (60%)

# NodeJS

The driver I started out originally was NodeJS.

Node.js is an open source server framework that allows you to run JavaScript on the server. I downloaded the MongoDB driver for NodeJS via npm (Node Package Manager).

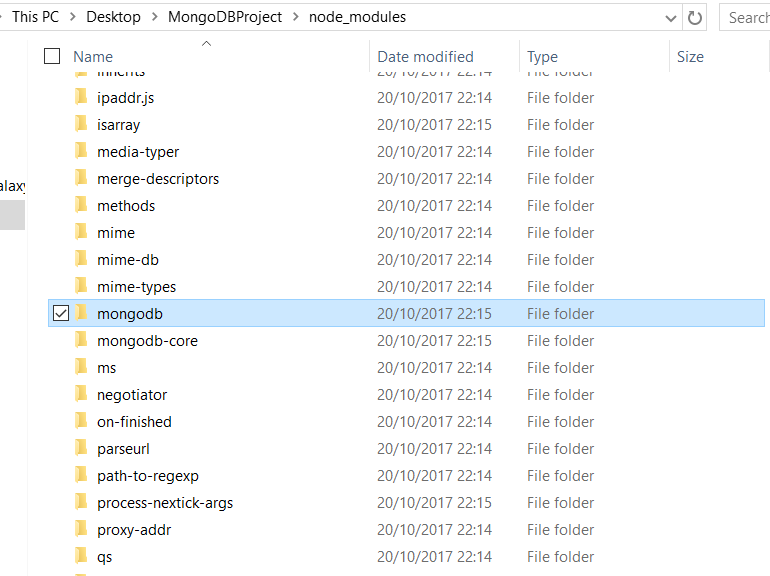
To start I created a *MongoDBProject* directory. On the command line I changed directory to the *MongoDBProject* directory. I issued the command npm init to create the package.json file. It serves as documentation for what packages your project depends on – what drivers and their versions are used etc. Then I installed the mongodb driver. The mongodb driver is saved into the node\_modules folder in the root of your project.

cd MongoDBProject

npm init

npm install mongodb --save

*Figure 5: Installing mongodb driver via npm*



*Figure 5.1 MongoDB in projects node\_modules*

To learn more about Node I completed Create or Insert without a UI

*//Insert*

**var** MongoClient = require('mongodb').MongoClient;

*// Connect to the db*

MongoClient.connect("mongodb://localhost:27017/exampleDb", **function**(err, db) {

**if**(err) { **return** console.dir(err); }

**var** collection = db.collection('test');

**var** doc1 = {'hello':'doc1'};

**var** doc2 = {'hello':'doc2'};

**var** lotsOfDocs = [{'hello':'doc3'}, {'hello':'doc4'}];

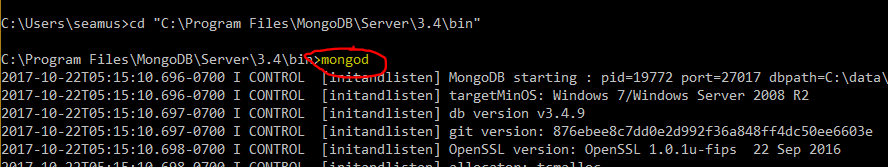
collection.insert(doc1);

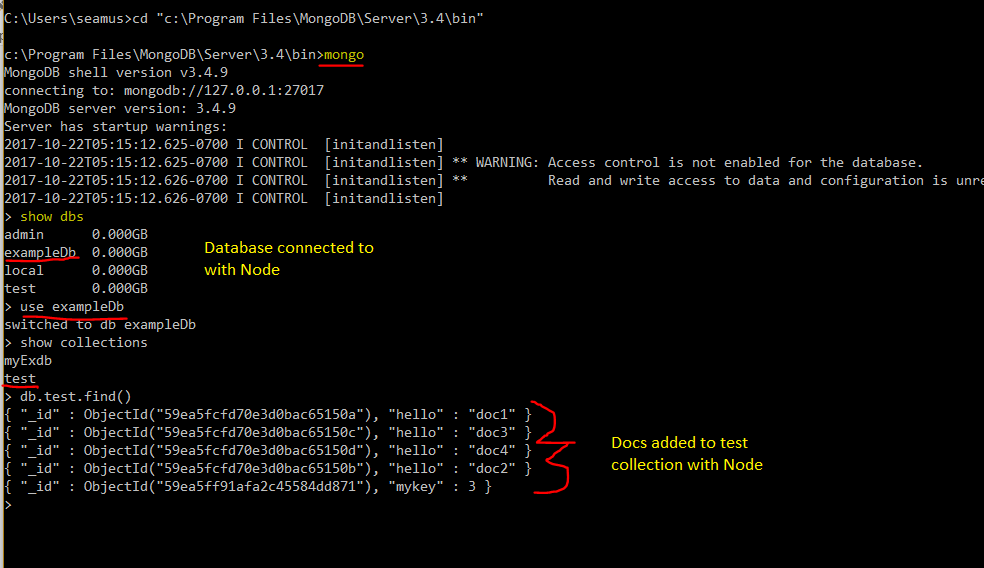
collection.insert(doc2, {w:1}, **function**(err, result) {});

collection.insert(lotsOfDocs, {w:1}, **function**(err, result) {});

});

Starting MongoDB





## Node Modules

Equivalent to the packages in Java. Node modules are a set of functions/libraries to be included in your application. There are some built in modules included by default like: assert, dgram, dns, http etc. You can include/import modules using the require() function. var http = require('http');

Mongodb

The officially supported Node.js driver for MongoDB that handles connections to a MongoDB server, a replicaset or a set of Mongos proxies in a sharded system. It let’s your application interact with the server and lets it perform operations against the database. Installing Mongodb driver node on npm: npm install mongodb --save

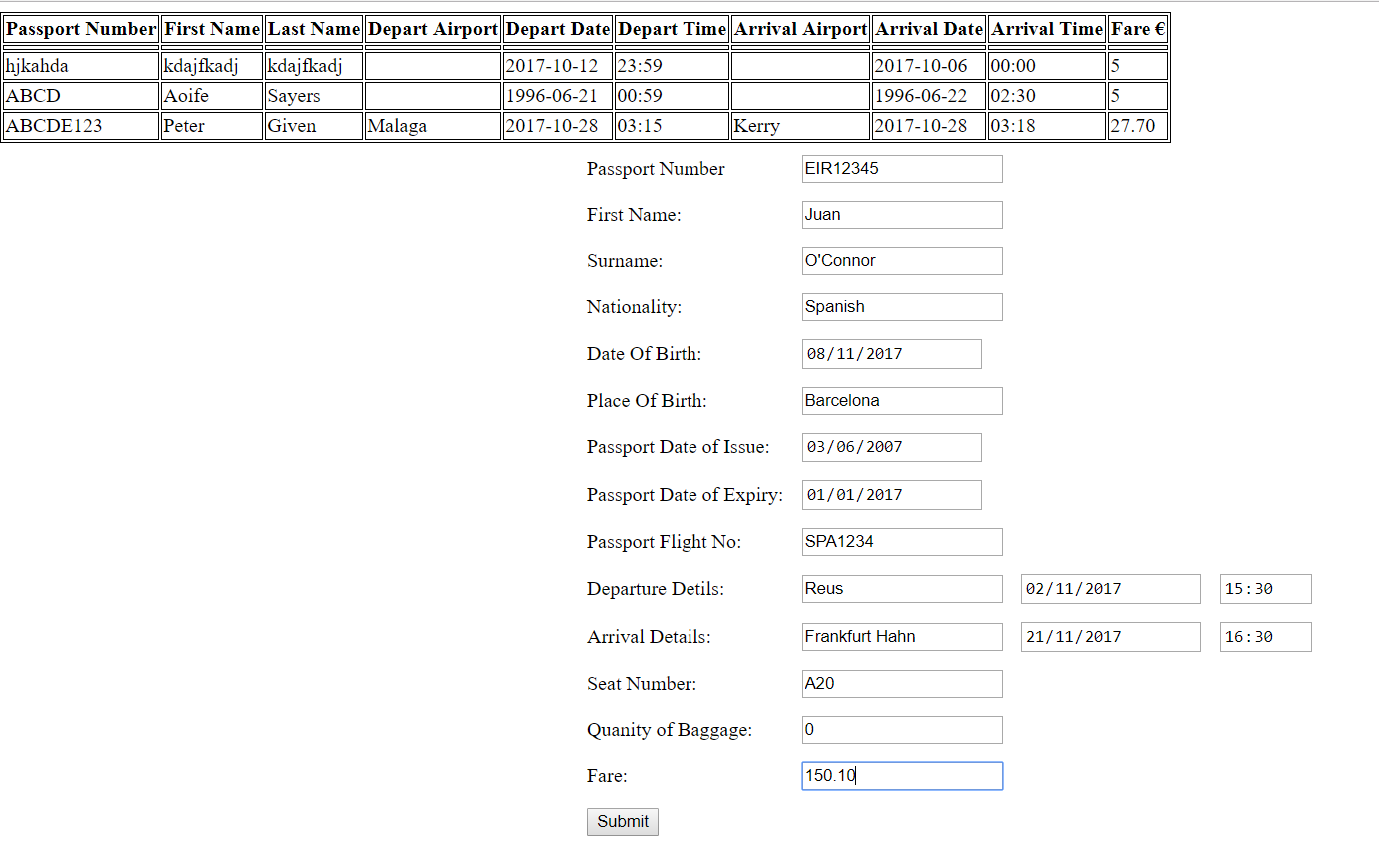
Express

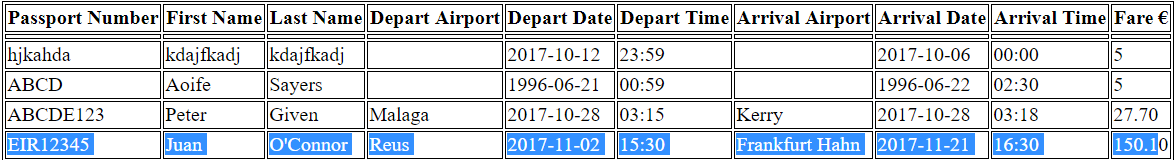
Express is lightweight Node.js web application framework that provides an efficient set of features for web and mobile applications. Respond to HTTP Requests. Defines a routing table which is used to perform different actions based on HTTP Method and URL. Allows you to dynamically render HTML Pages based on passing arguments to templates.

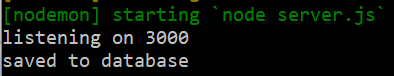
Body-parser

This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.

## Create UIF









Server.js code snippet

res.render('index.ejs', {flightDetails: result}) – takes the name value from the html form in index.ejs.

For Example:

<input type="text" name="passportNo" required> - value in json is passportNo

Passport number and other entries are passed into the method and inserted into the flightDetails collection. On successful insertion, a console.log is printed out and the res.redirect(‘/’) reloads the page

app.post('/flightDetails', (req, res) => {

db.collection('flightDetails').save(req.body, (err, result) => {

**if** (err) **return** console.log(err)

console.log('saved to database')

res.redirect('/')

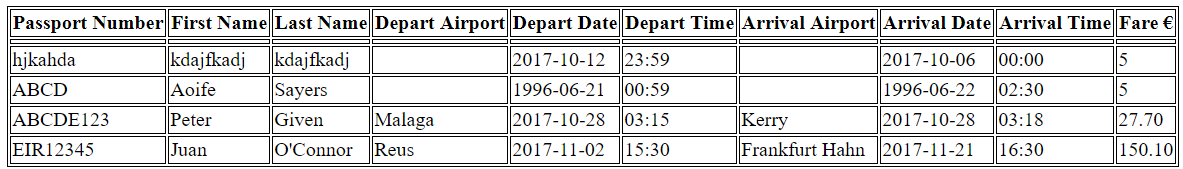
})

})

## Read UIF

Read selected attributes of the JSON document into a HTML table with NodeJS and Express. The result of the inserted document above appears in the table. The html file below is in a ejs file.

EJS

"E" is for "effective." EJS is a simple templating language that lets you generate HTML markup with plain JavaScript

Server.js Code Snippet

The server reads in all the details the flightDetails collection into an array using the GET method.

app.get('/', (req, res) => {

db.collection('flightDetails').find().toArray((err, result) => {

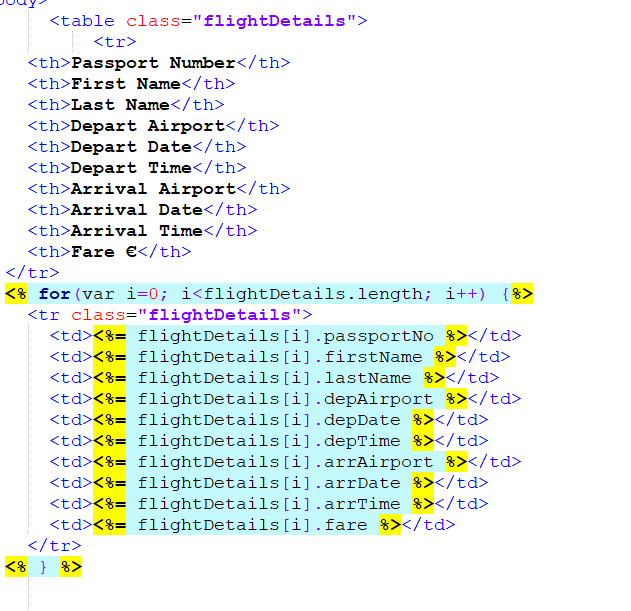
**if** (err) **return** console.log(err)

res.render('index.ejs', {flightDetails: result})

})

})

Index.ejs Code Snippet

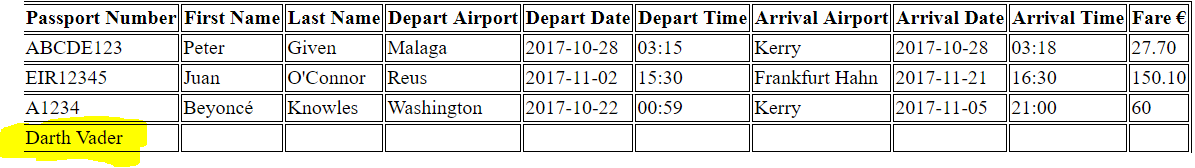
EJS is a simple templating language that lets you generate HTML markup with plain JavaScript by placing JS inside ***<%= … %>***. Index.ejs file which reads in the flightDetails array from the server.js using embedded JavaScript. Specifies which fields it places inside in a HTML table

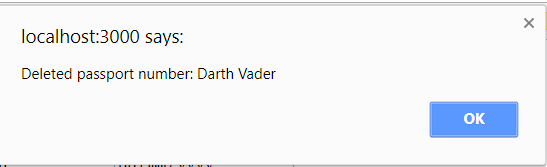
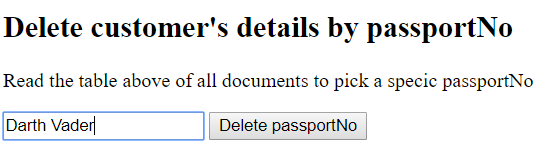
## Update UIF

Gone to shit!💩💩💩💩💩💩💩💩💩

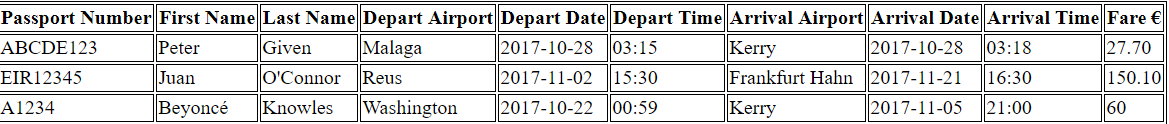
## Delete UIF

I want to delete the document with a passport no of Darth Vader. The HTML table below shows the current list of documents in the collection.





The flightDetails collection after deleting passportNo: Darth Vader. I used the findOneAndDelete()



Delete Code Snippet

Index.ejs

Regular HTML text box and button

**<div>**

**<h2>**Delete customer's details by passportNo**</h2>**

**<p>** Read the table above of all documents to pick a specic passportNo**</p>**

**<input** **type=**"text" **id=**"nameToDelete" **placeholder=**"Passport No" **required>**

**<button** **id=**"delete"**>** Delete passportNo **</button>**

**</div>**

Main.js

Javascript event handlers to handle click event & set up delete

del.addEventListener('click', **function** () {

fetch('flightDetails', {

method: 'delete',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({

'passportNo': passportDel.value

})

}).then(**function** (response) {

window.alert("Deleted passport number: " + passportDel.value)

window.location.reload()

})

})

Server.js

app.**delete**('/flightDetails', (req, res) => {

db.collection('flightDetails').findOneAndDelete({passportNo: req.body.passportNo}, (err, result) => {

**if** (err) **return** res.send(**500**, err)

res.send('A customers flight details were deleted')

})

})

# Conclusion

I chose to model a flight details database as …

I really like MongoDB as a NoSQL database as it’s a very robust and efficient database with it’s sensical methods.

Advantages of MongoDB

* MongoDB enables horizontal scalability by using a technique called sharding. Sharding distributes the data across physical partitions to The data is automatically balanced in the clusters.
* ACID properties at the document level as in the case of relational databases.
* Replica Sets; in other words, a failover mechanism is automatically handled. If the primary server goes down, the secondary server becomes the primary automatically, without any human intervention.
* It supports the common authentication mechanisms, such as LDAP, AD, and certificates. Users can connect to MongoDB over SSL and the data can be encrypted.
* MongoDB can be a cost effective solution because improves flexibility and reduces cost on hardware and storage.

Disadvantages of MongoDB

* NoSQL DBs are relatively new, so there are not that many established patterns which help you structure your DB efficiently. This issue might result in performance penalties.
* MongoDB is very memory hungry. Your server will need a lot of RAM
* Some operations might lock your entire database
* You have to update many things manually

I am happy with how this project turned out. This was my first time using NodeJS as a driver and this project was a great learning curve for learning about NodeJS as well as MongoDB. NodeJS is also a very powerful JavaScript Library. I think Node was the right choice for a MongoDB driver. I decided to go with NodeJS as MongoDb’s various methods are written using JavaScript. Express rendered files and allowed for simple UI integration with NodeJS. Ideally I would have liked to use Mongoose. Mongoose provides a straight-forward, schema-based solution to model your application data.

I did not use any other drivers to complete this project.

I plan to use MongoDB and NodeJS again for other projects.

# References

Bland, J., 2016. *Saving data to MongoDB database from nodejs application tutorial.* [Online]   
Available at: http://www.jenniferbland.com/saving-data-to-mongodb-database-from-node-js-application-tutorial/  
[Accessed 31 October 2017].

EJS, 2017. *EJS.* [Online]   
Available at: http://ejs.co/  
[Accessed 25 October 2017].

Liew, Z., 2016. *Building a Simple CRUD Application with Express and MongoDB.* [Online]   
Available at: https://zellwk.com/blog/crud-express-mongodb/  
[Accessed 31 October 2017].

MIT, 2017. *Mongoose JS Docs.* [Online]   
Available at: http://mongoosejs.com/docs/  
[Accessed 31 October 2017].

MongoDB, 2017. *MongoDB Architecture.* [Online]   
Available at: https://www.mongodb.com/mongodb-architecture  
[Accessed 20 October 2017].

node-mongodb-native, 2017. *MongoDB Node.JS Driver.* [Online]   
Available at: https://mongodb.github.io/node-mongodb-native/  
[Accessed 31 October 2017].

W3Schools, n.d. *NodeJs Tutorial.* [Online]   
Available at: https://www.w3schools.com/nodejs/  
[Accessed 31 October 2017].