Katie Griffiths

T00175748

MongoDB

Advanced Database Programming

# Introduction

MongoDB is a free open source document database with the scalability and flexibility needed for querying and indexing. The name Mongo was derived from the word “humongous” to represent the idea of supporting large amounts of data. MongoDB stores data in flexible, JSON like documents which means fields can vary from document to document and data structure can be changed over time. The document model maps to the objects in the application code which makes the data easy to work with. Ad hoc queries, indexing and real time aggregation provide a powerful way to access and analyse the data within the database. A mongo document can be likened to a relational table row without a schema, whose values can nest to an arbitrary depth. Automatic sharding enables data in a collections to be distributed across multiple systems for horizontal scalability as data volumes increase. Users that use MongoDB range from Craigslist using the database for archiving data and The New York Times newspaper who use MongoDB to support a form building application for photo submissions.

The main features that MongoDB offer is

* Expressive Query Language & Secondary Indexes
* Strong Consistency
* Enterprise Management & Integration
* Flexibility
* Scalability & Performance
* Always On, Global Deployments

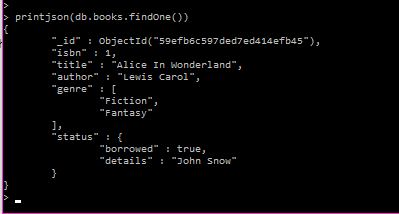
Mongo is an excellent choice for an ever-growing class of web projects with large scale data storage requirements but very little budget to buy big iron hardware, Mongo has the ability to grow and change along with your data model.

# Question 1

The idea for this project was a book depository web application to be created which would manage books in a collection. The application would allow for new books to be created as well as having a field called status, this would allow a book to be changed from being available or if it was issued out. The access of this application would be across multiple devices allow for updates and creations to be performed quickly and easily. MongoDB caters for large amounts of data so I found creating an application like a library system would be a good fit. Each Book would have a unique ISBN number attached as well as a Title, Author, Genre and Status. Status would also manage the ability to remove a book from the collection. I felt using JavaScript as the language for this project was good idea as JSON is native to it, making it trivial, in many cases to work with the MongoDB documents in a variety of ways without having to worry about issues that can occur in other languages when switching back and forth between JSON and their native language implementations. I found that running MongoDB with node and express as the web framework a much quicker way to get it up and running. This allowed me to easily implement a webserver that could handle requests, templating and other useful helpers for creating a website.

# JSON Documents

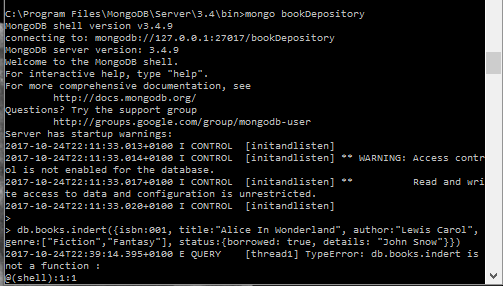
An example of a JSON document is



# CRUD Commands through REST Interface

## Create

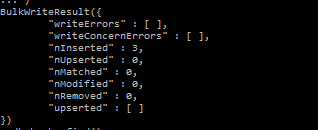
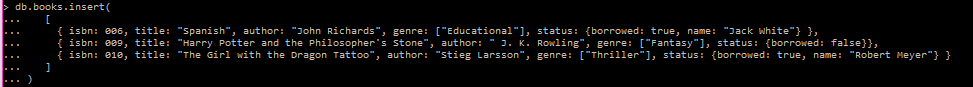
To create a database in MongoDb, I ran the command mongo bookDepository in the terminal, this then created an empty database to be populated with document inserts.



After creating a database called books, I populated the database with the following code which creates a books collection in the database.

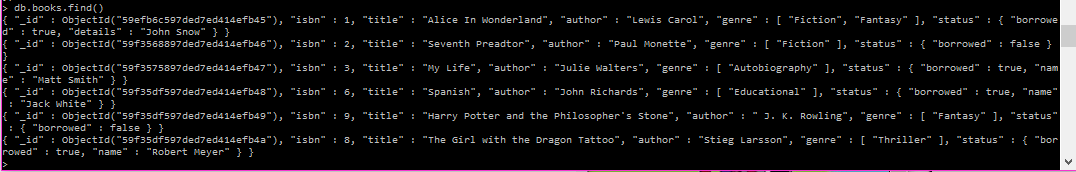


After creating the first collection I decided to populate the collection by adding in a few more documents, this was done by inserting multiple documents as a bulk insert.



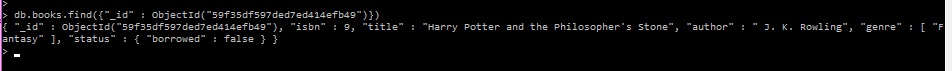
## Read

The following code db.books.find() is used to read the documents that are currently in the database. By using this you are able to see the \_id ObjectId that was assigned to each inserted document.



While using the find() function retrieves all of the documents, you can also access a specific document by setting an \_id property which is of type ObjectId and this will then be used to convert a string by wrapping it in an ObjectId(str) function e.g. db.books.find({"\_id" : ObjectId("59f35df597ded7ed414efb49")})

This then finds the specific document with the unique ObjectId.



The find() function also accepts an optional second parameter, a field object which can be used to filter the document fields that are being retrieved. As shown below I chose to find a document that only had the Author J.K. Rowling, by passing author with a value of 1 (true) into the second parameter of the find() function, I was then able to retrieve the document with just the ObjectId and Author. db.books.find({ \_id : ObjectId("59f35df597ded7ed414efb49") }, { author : 1 })



Ad Hoc quieres can be constructed in mongo, they can be constructed by querying field values, ranges or a combination of criteria. By running this function:

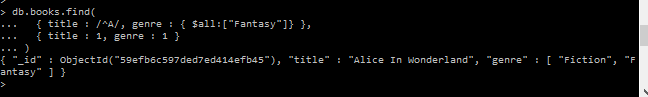
db.books.find(

{ title : /^A/, genre : { $all:["Fantasy"]} },

{ title : 1, genre : 1 }

)

I was able to retrieve all books that begin with the letter A and that have a genre of Fantasy. The result of the function was one book titled Alice in Wonderland with a genre of Fantasy.



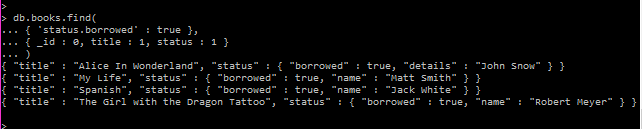
An ability of Mongo is being able to query a subdocument. This can be done by using your field name as a string separating nested layers with a dot. I used the following code to find all books which have a status of being borrowed set to true.

db.books.find(

{ 'status.borrowed' : true },

{ \_id : 0, title : 1, status : 1 }

)



## Update

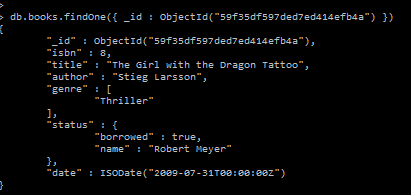
The update function requires two parameters these are update(criteria,operation) the criteria query which would be the same sort of object that would be passed into the find() function, the second parameter can be either an object whose fields will replace the matched document or a modifier operation. In this function the modifier is to $set the field date with an ISODate. This then shows the date in which a book was borrowed out at.

db.books.update({ \_id : ObjectId("59f35df597ded7ed414efb4a") }, { $set : { "date" : ISODate("2009-07-31") } } );



To the verify if the update was successful I decided to use the findOne() function to retrieve only one matching ObjectId.

db.books.findOne({ \_id : ObjectId("59f35df597ded7ed414efb4a") })



## Delete

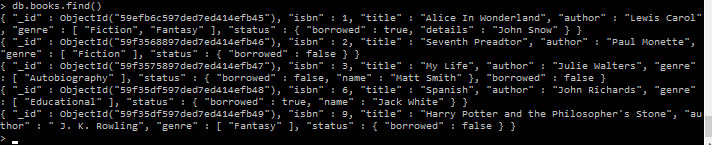
To remove a document from a collection you call the remove() function, which will then remove all the matched criteria. For this I chose to remove a particular book from the collection. By call this function:

db.books.remove( { date: ISODate("2009-07-31") } )

I was able to remove the associated document that had that particular borrowed date.



Calling the find() method I can see that isbn:10 title Girl with a Dragon tattoo had been removed from the database.



# Question 2

For the purpose of this project I chose to develop the application using JavaScript (through node). MongoDB is very intuitive to use from JavaScript, as they both deal with JSON exclusively. Mongoose is an Object Document Mapper (ODM) which means that Mongoose enables you to define objects with a strongly-typed schema that is mapped to a MongoDB document. Mongoose provides an huge aount of functionality around creating and working with schemas, it currently contains eight SchemaTypes that a property is saved as when it is persisted to MongoDB. They are currently:

* String
* Number
* Date
* Buffer
* Boolean
* Mixed
* ObjectId
* Array

Each data type allows you to specify:

* A default Value
* A custom Validation Function
* Indicate a field required
* A GET function that allows you to manipulate the data before it is returned as an object
* A SET function that allows you to manipulate the data before it is saved to the database
* Creation of indexes to allow data to be fetched faster

Mongoose is a JavaScript library that allows you to define schemas with strongly typed data. Once a schema is defined, Mongoose lets you create a Model based on specific schema. A Mongoose Model is then mapped to a MongoDB Document via the Model’s schema definition.

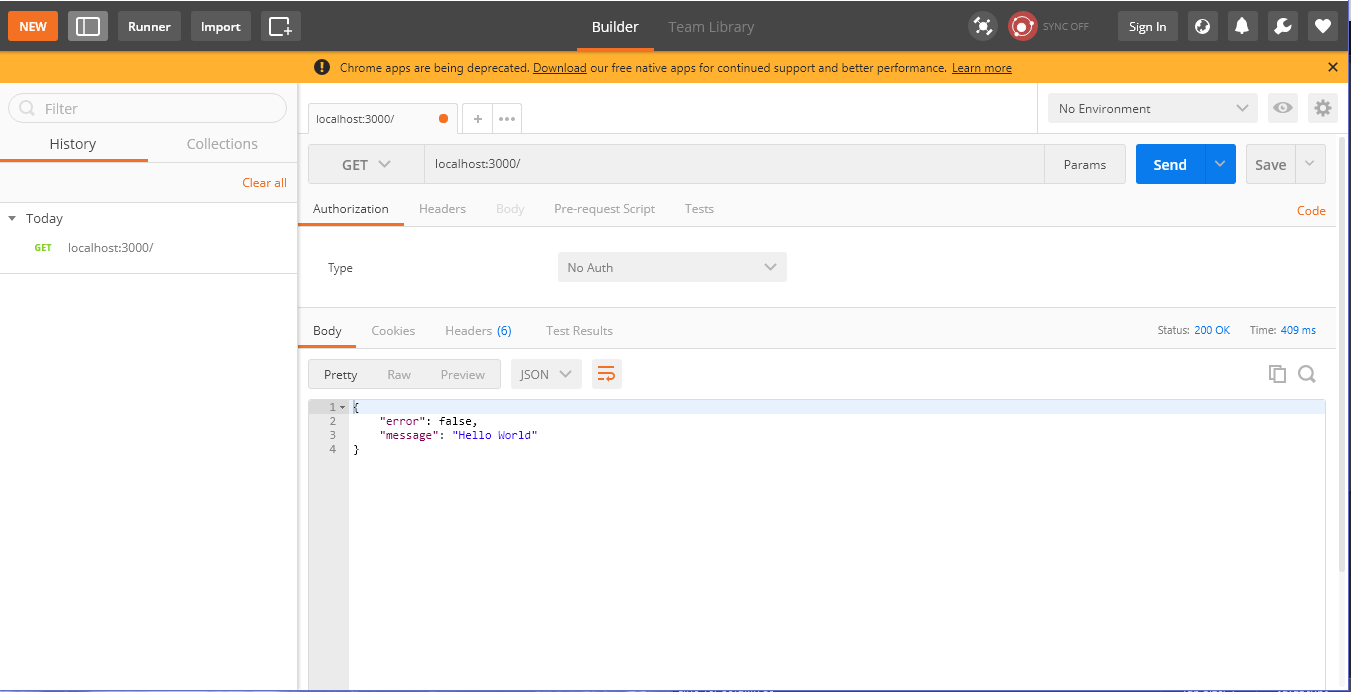
As I previously had Node.js installed I created a new application and installed the Mongoose NPM Package. By running npm install mongoose –save through the command line, the mongoose package is installed. To then connect to the MongoDB database I placed the following code into my index.js file as it was the starting point of the application:

var mongoose = require(‘mongoose’);

mongoose.connect(‘mongodb://localhost/bookdepository’);

# CRUD through the API

Testing first route using PostMan REST simulator chrome extension



## Create

I to create a book I used the POST function which would insert a book into the collection, the criteria needed in creating a book are the title, author, isbn number and an array of genres. Once these fields have been filled the function will then save the book into the collection.



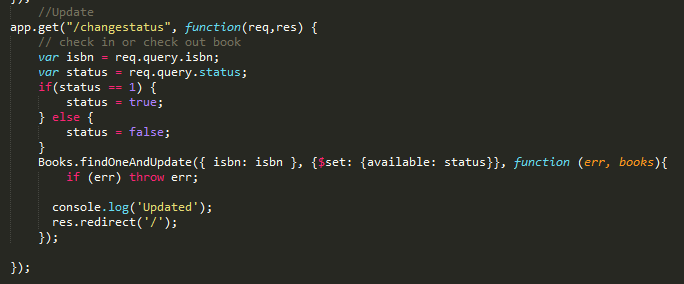
## Read

This function lists the books in the UI, the function uses the GET method, if there are no books availiable then an error message will be displayed into the console and returned to the user on screen, once the function starts it goes through an if statement containing a foreach method, then displays the list of books.



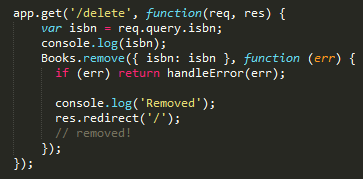
## Update

This JavaScript function allows the user to update the status of a book to indicate if the book is in or out, by selecting the change status button, this function is called and searches for the matching book by ISBN. If found, then the status will be changed to either true to false as required. The findOneAndUpdate is used to set the available status.



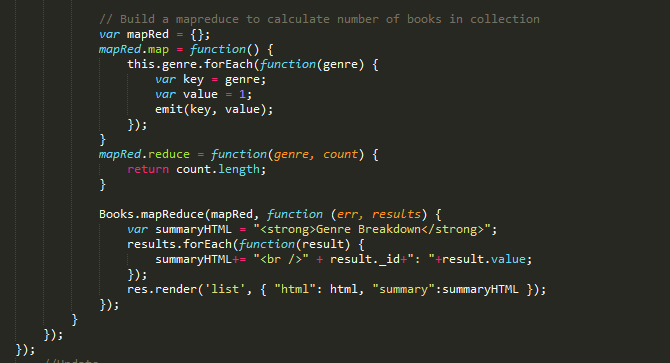
## Delete

This function uses the GET method that deletes a book, indexed by it’s ISBN. If a book is not found, not a handle err is sent.



# Mapreduce through the API

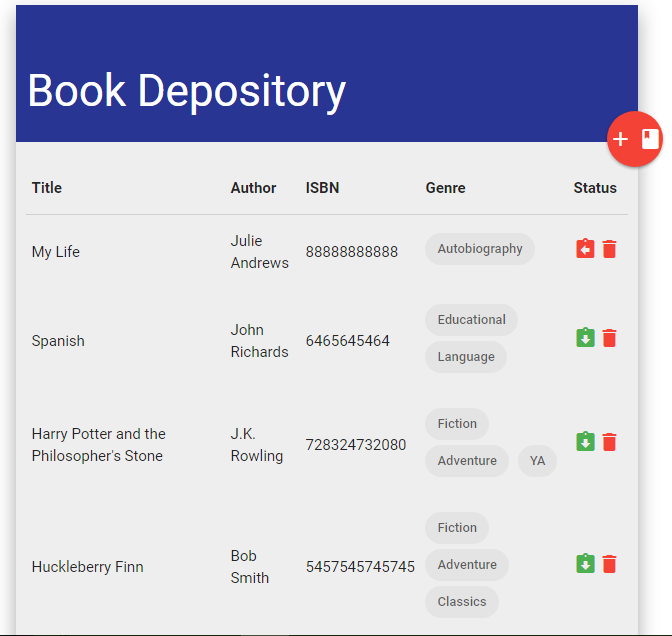
This mapreduce calculates the number of books present in each unique genre. A mapRed variable is created for the mapreduce function. For the map, the method emits once for each genre in each book. The reduce function calculates the total number of times the genre was present with the javascript length() method. The results are then put into the application through the html. Showing the genre and the amount of books that are assigned the specific genre



# User Interface

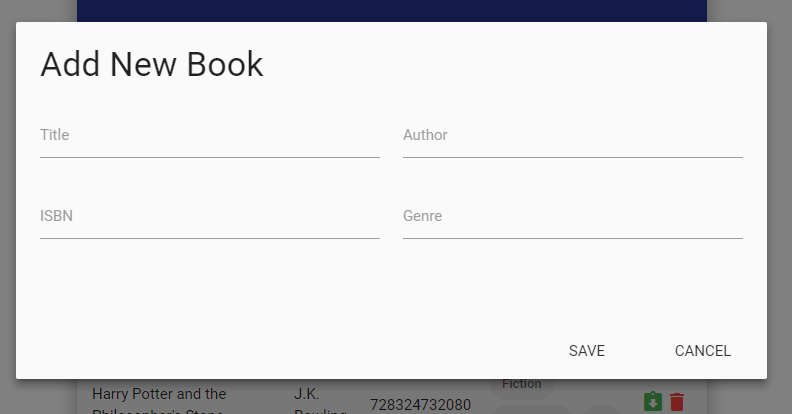
## Main

This is the main interface that the user will view, the book Depository application lists the details of the books that are in the book depo collection. The main details shown for the books include the Title, Author, ISBN number, Genre and Status. These fields are required when adding in a new book to the collection. The red plus button will allow the user to add a new book into the collection. Under the status there are two buttons this shows if the book is out or if it is available, simply by showing the colour green or red. The trash can button is a deletion button which will call the delete function and remove the book from the collection.

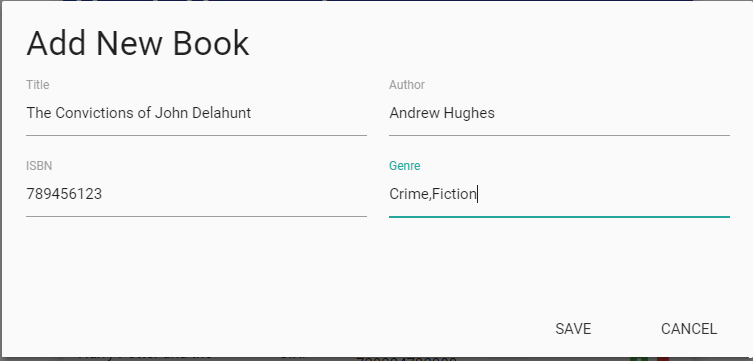


## Create

This interface is called by pressing the add book symbol. The create function is called by pressing the save button after all of the details of the book has been entered into the system. All text fields must be filled in for a new book to be saved into the collection.

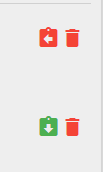


This screen shot shows data being entered into the text fields for a book to be stored in the book depo collection.

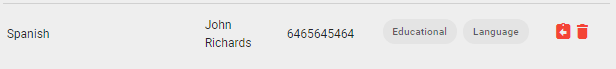
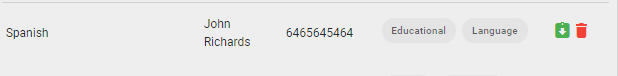


## Status Alteration Buttons

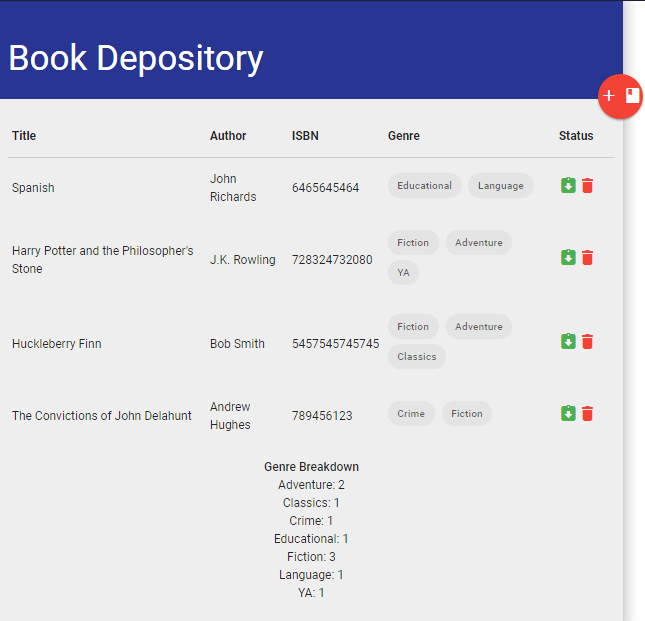
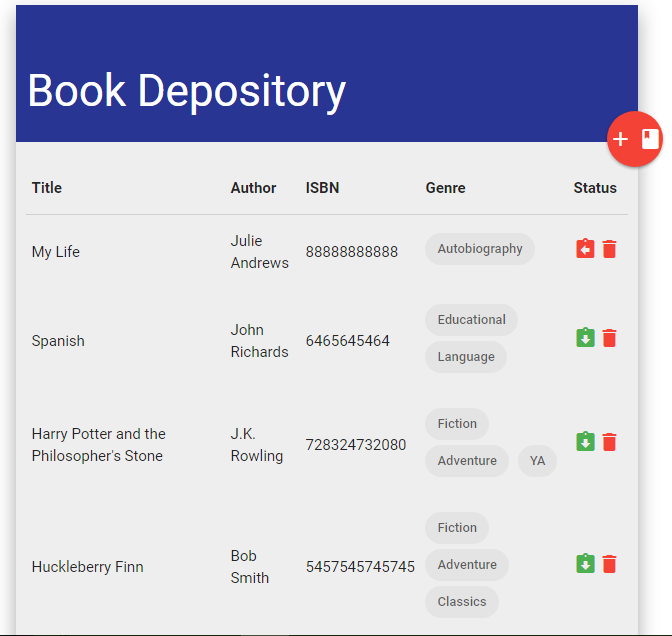
These buttons are used when the status of the book needs to be changed from available or out. The green shows that a book is available in the system, the red shows that a book is currently issued out. These are changed by calling the update function. The trash can symbol is the delete button, this is for when the user wants to delete a book from the collection, this calls the delete function.



This screen shot shows a before and after of a book status being changed.

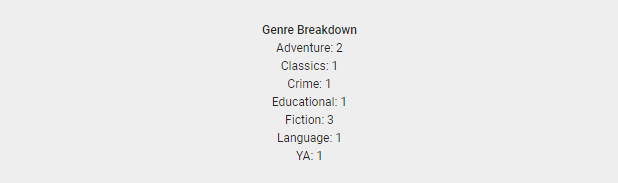


When the trash can button is pressed the delete function is called. These screen shots show a before and after of the list of books when a single book is deleted.



## MapReduce

The mapreduce function allows the breakdown genres from the book collection to be seen. The function allows for all the books to be counted by their genre and then put into the categories.



# Advantages and Disadvantages

Using Mongoose as a Node.js library translates data in the database to JavaScript objects for use in the application. Everything in Mongoose starts with a Schema, each of those schema’s maps to a MongoDB collection and defines the shape of the documents within that collection. A main advantage of using Mongoose with Node.js is the data validation that is built into it and that most of the MongoDB code will be abstract away from the rest of the application. It also offers features to make sure the schema is consistent when inserting/updating and finding documents from the collection, it also creates model abstraction which makes it easier to work with and also includes middleware, plugins, population and validation. The added ability to modify the schemas as your application grows is not a big task.

Using Mongoose to develop an application quickly is an advantage as there is no overhead of creating a connection and closing it in proper time also optimizing the connection, making promises etc. The abstruction layer of Mongoose does all of those for you. Maintainability is a big factor for any long term project, Mongoose has the ability of using wrappers which forces the implementation of creating schemas and the ability to do things with the models gives the project a structure which allows the code to be easily maintained.

**Disadvantage**

A disadvantage would be having to create schemas from the start which defeats the purpose of using NoSQL. Encapsulation with the model is hard initially unless the criteria for the application is clear and have a clear understanding of the data flow before starting to build the application.

# Conclusion

I found that using MongoDB very easy to learn and get to grips with. The project at the moment is quiet small but with enough time I feel there would be more collections added as well as many more books into each of the collections. When researching for API’s I found it much easier than a previous project to find up to date documentation and reliable API’s. I decided to use JavaScript as the language and in my opinion was a good choice, as MongoDB can handle JavaScript to perform CRUD operations quiet easily. At first when starting with Mongoose I found the creation of schemas by models quiet hard but once they were instantiated the running of the CRUD operations became much easier.