CouchDB project

T00193959 | 04/10/2019

Advanced Database Programming

Seán Hogan – T00193959

2019

Table of Contents

[Introduction 2](#_Toc23801671)

[Node.JS 2](#_Toc23801672)

[CouchDB 2](#_Toc23801673)

[Application 3](#_Toc23801674)

[View books 3](#_Toc23801675)

[Delete Books 7](#_Toc23801676)

[Update Books 9](#_Toc23801677)

[Add Books 13](#_Toc23801678)

[Filter Books 18](#_Toc23801679)

[Conclusion 23](#_Toc23801680)

[References 24](#_Toc23801681)

# Introduction

I decided to create a web application using node.js. The application I created is a book store management application which allows the user to be able to manage books on the system. The application allows the user to view the books on the system, delete books off the system, update books, add books and view the books depending on the genre that is chosen.

# Node.JS

*“Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Node.js was developed by Ryan Dahl in 2009 and its latest version is v0.10.36.”* (Point, n.d.)

I wanted to use node.js as I have some prior experience with it. I wanted to be able to understand it in more detail as I find doing projects is the best way to improve on something. At the end of this project I want to be able to use Node.JS to add, delete, update and view data from CouchDB on a user interface.

# CouchDB

My database stores the details of each book on the System. Each book consists of a book name, cover, author, genre, price and stock. The following is an example of one document that in on CouchDB

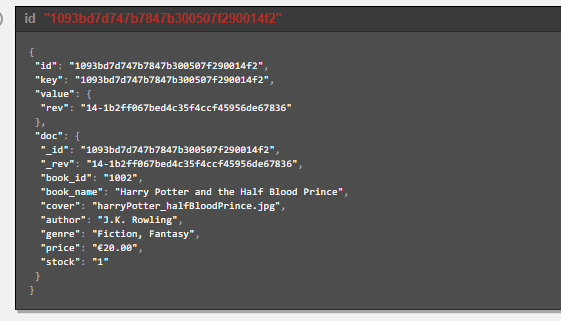


Figure 1 - CouchDB Document Example

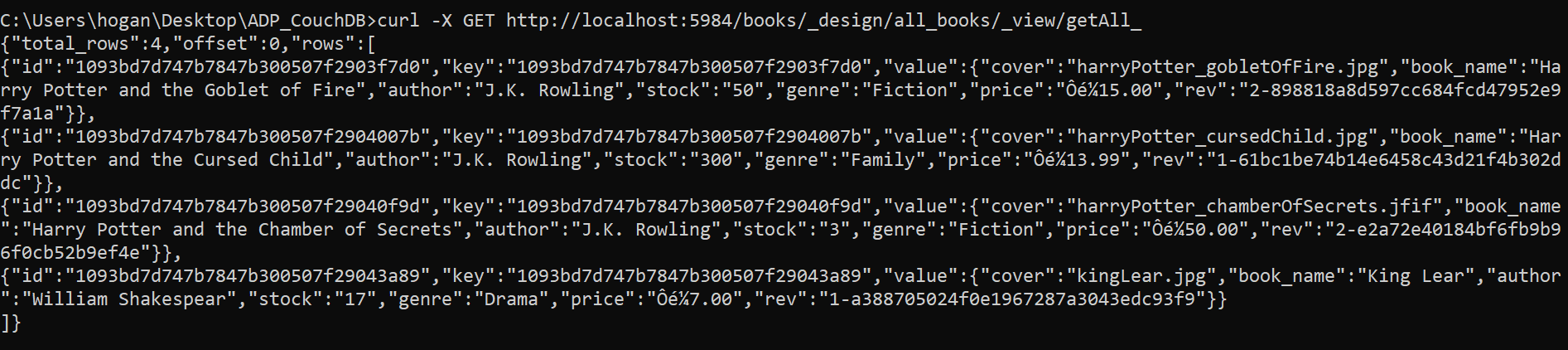


Figure 2 - CURL get all books

# Application

For this project I used node.js along with the application framework Express. I also used embedded JavaScript files (.ejs) to be able to write HTML code along with JavaScript embedded within the HTML. I have experience with express from previous projects and I wanted to use it again to firstly not complicate the project too much and to secondly understand more on express.

## View books

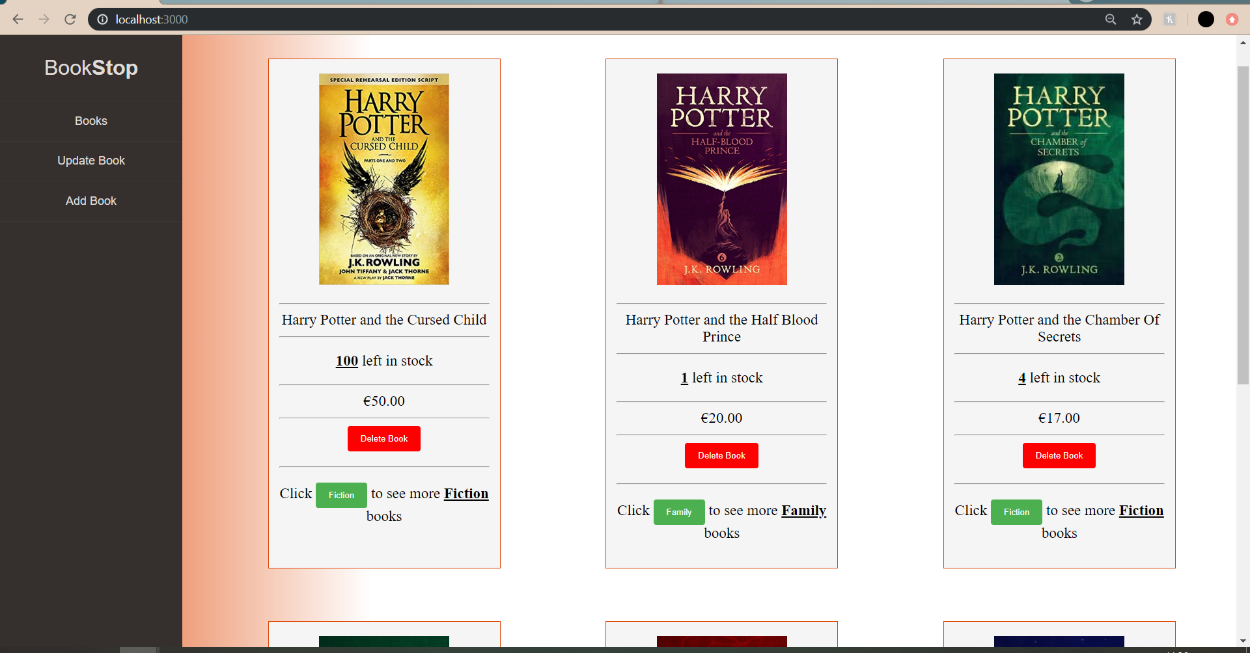
The Home page consists of the books that are currently on the system. This will allow the user to see the name of the book along with the number of stock and the price of the book. There is also an image of the book cover for the user to see. 

Figure 3 - Home page

I created a view on CouchDB that allows all the books on the system to be returned to the screen. The map function will go to each document and get each value of the book from the document. As it can be seen from the above image, each book is returned with the cover image along with the name of the book, stock and price. Each book is returned in its own box for easier visibility of each book

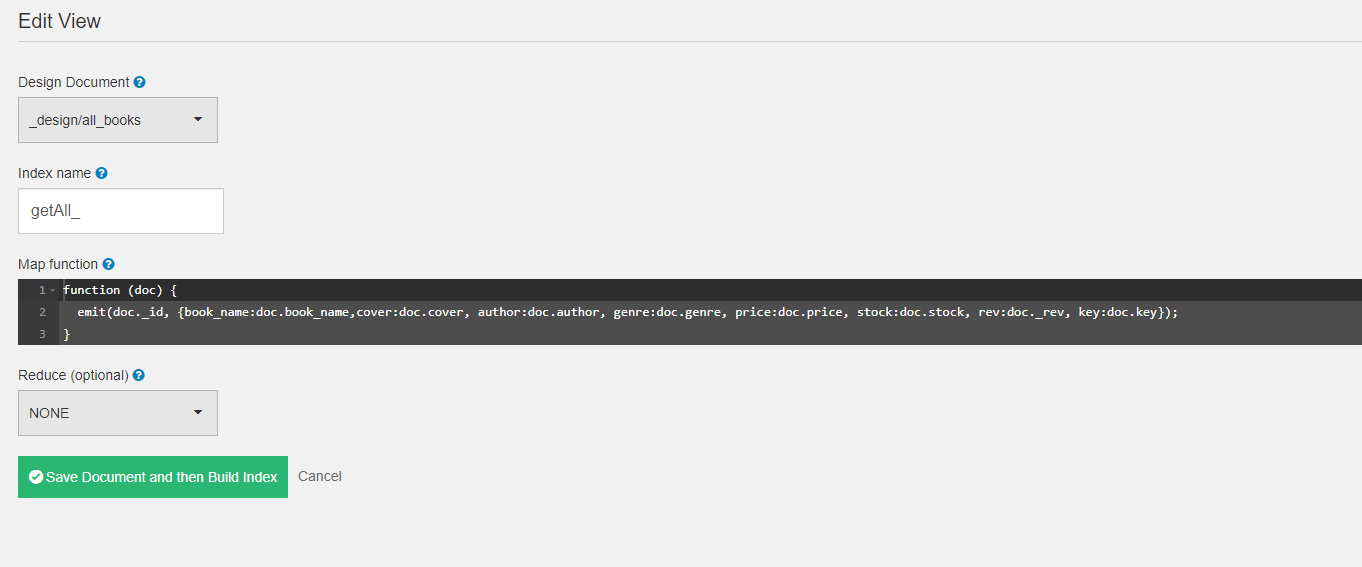


Figure 4 - Get all books map function

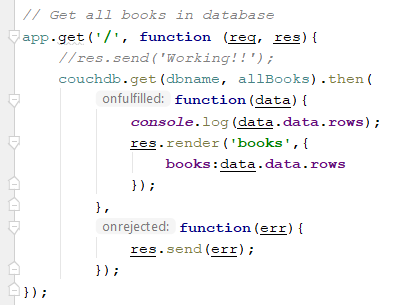


Figure 5 - Get home page code

The above function written in JavaScript uses the view that was created in CouchDB to get all the books in the database. The “couchdb.get()” takes in two parameters. The first one is the database name and the second being the URL of the view. The function will then render the embedded JavaScript file where the data will be available to use. If there is an error, an error message will be sent to the screen. The data is retrieved by using JavaScript within the embedded JavaScript file. As

there is more than one document in the Database a loop will have to be created to be able to iterate through all the documents getting all the data from each document.



Figure 6 - Code to display books on home page

In the above image we can see how the loop was created. The name of the database followed by the forEach method that has a function which takes in a parameter. In this case, the name of the parameter is “books”. Each bit of data is called by using some JavaScript. To embed JavaScript within the HTML you must use the “<%= %>” tag. Within the tag the function parameter followed by the word value and then followed by the data you want. For example if I wanted the book name I would do as follows <%= book.value.bookName %>. The forEach method will loop through until it reaches the last document which it will then finish its loop.

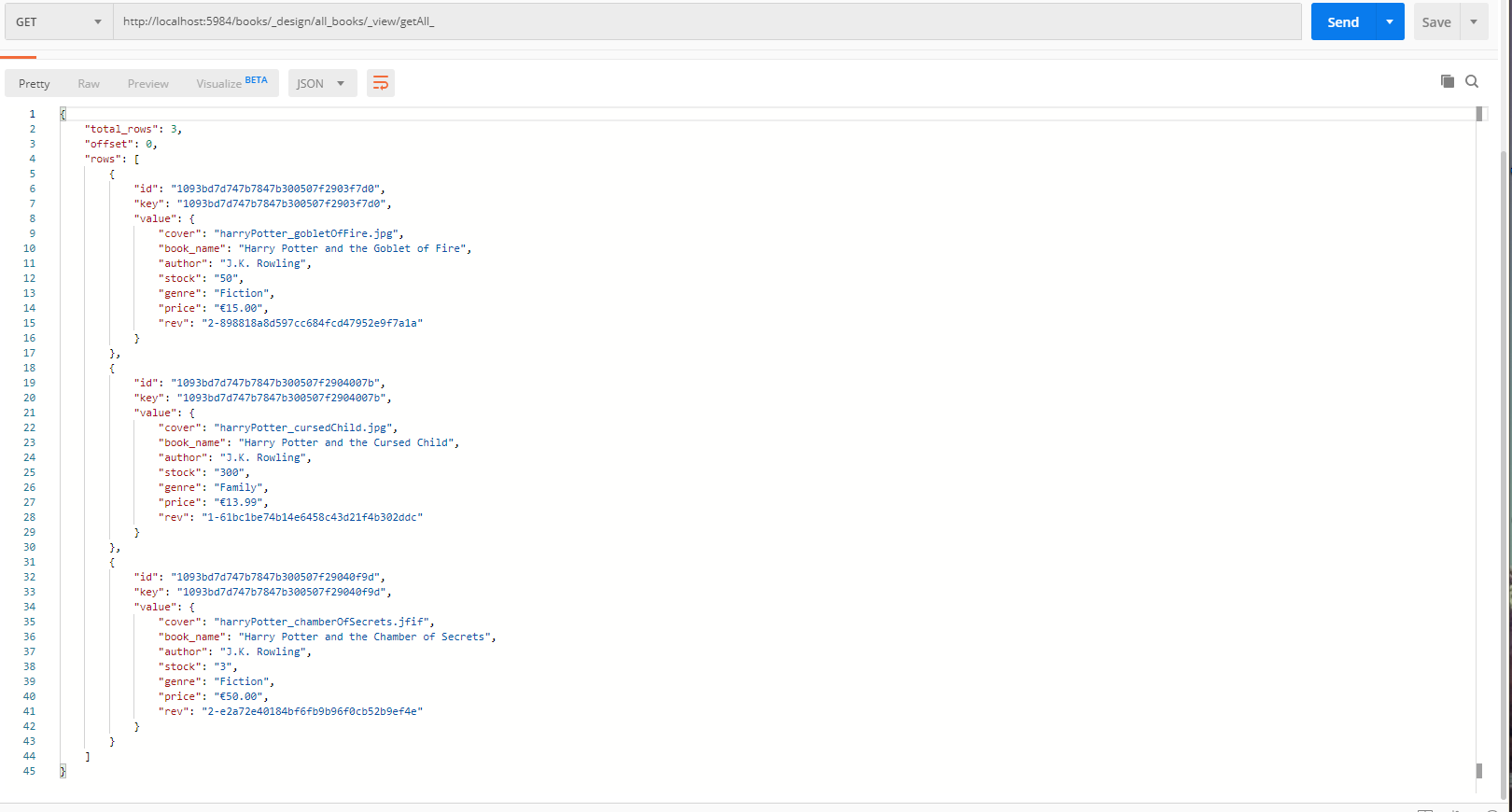


Figure 7 - Postman get all books



Figure 8 - Command line all books returned

## Delete Books

The delete book functionality is working with the view for getting all the books. As the books are returned a button is also being created as the function loops, in turn associating each book returned with a delete book button in red as can be seen below.



Figure 9 - Example of book

The form for the Delete book button uses a post request created in the app.js file that will send a request to delete the selected book. Adding a button to the loop for each book will not be enough to be able to determine what book it is when you click the delete button as it won’t know what to delete. Each document has a unique key generates with it so in the action part of the method I include the key at the end of the URL as can be seen below

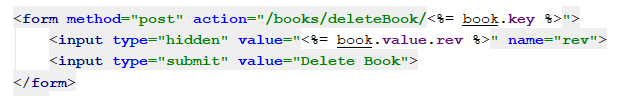


Figure 10 - Form for deleting a book

This will generate the key onto the end of the action URL allowing the button to know what it is it will be referencing. An example of this can be seen if we look at the source code.

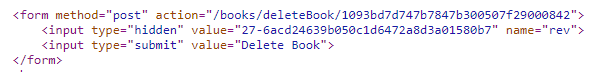


Figure 11 - Source code of form for deleting a book

Once the form has been created and is displaying the correct key, a post request must be created in app.js to delete the selected book.

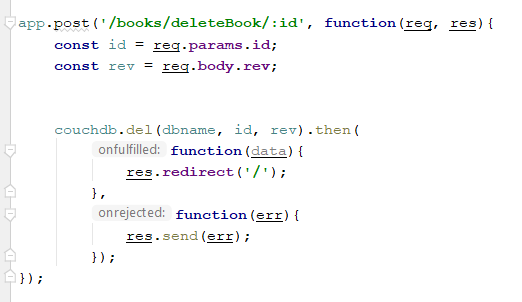


Figure 12 - Post request code for deleting a book

A post request with the corresponding method name is created and allowing for the key to be inserted with the “:id” at the end of the URL. Two variables are created, the id of the document and the rev of the document. Both these are gathered from the form with the delete button. The couch.del method takes in three parameters which include the database name, the id and rev of the document. Once the method is satisfied with the parameters, the post request with the delete will be sent and the user will be redirected back to the home page. Below is a delete request from Postman to show that it works.

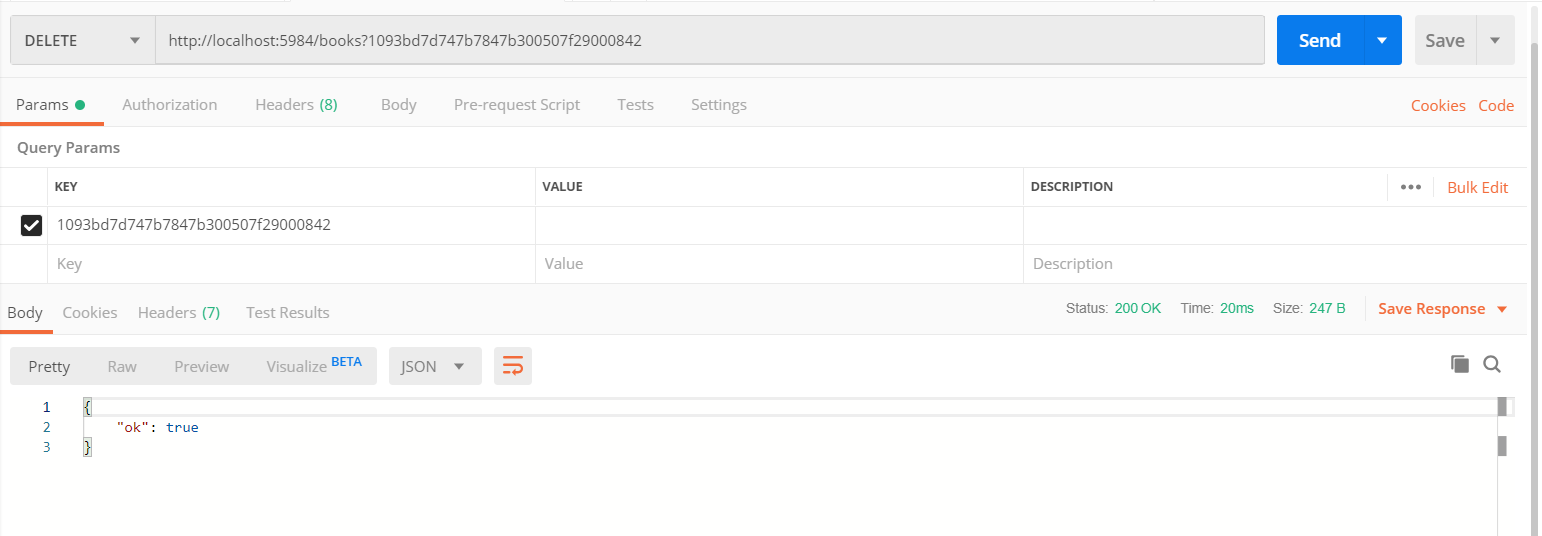


Figure 13 - Postman deleting a book

## Update Books

The update books work very similar to the delete books. It will get all the books on the database and return text boxes back with placeholders of the original data. The user will be able to select the box to be updated and enter in the needed data. The user will also be able to update the cover image of the book by selecting the image file that they want to change it to. Below is an example of what is displayed to the user on the update book page.

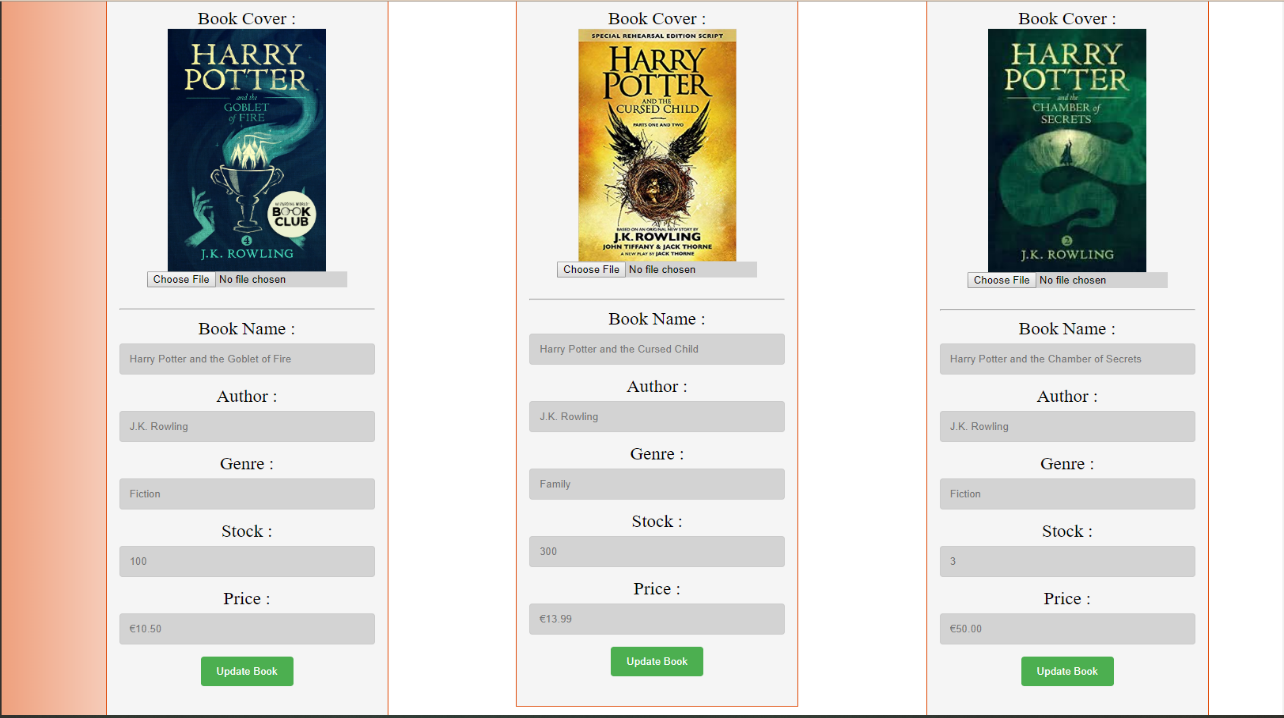


Figure 14 - Update book User interface

Firstly, to be able to get the update book page a get request for the update page URL must be created. This will be very similar to the home page get request as I still want to retrieve all the books on the database. The only difference will be the URL for the page. The code can be seen below.

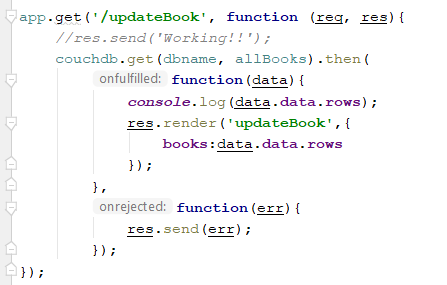


Figure 15 - Update book get request

The next task to do is to create a post request to the database with the updated data. A new embedded JavaScript file is created that is called UpdateBook that will allow the books to be loaded onto. A form with the post method and an action of the URL with the document key is also created. Within the form the details of the current book are brought back but as placeholders of the current book. This will show the user the current details of the books before any update is done. The user will fill out the details of the book they wish to update and they will click the update book button that will send a post request with the details entered and the book will be updated. An example of this is shown below of a book before and after the update.

Figure 16 - Before and after update book

As it can be seen from the above two images, before the update there were 100 books in stock and the price of the book was €10.50 and after the update there are 50 left in stock and the price has been increased to €15.00. Below is a screenshot of the post request for the update book.

Same as the delete book post request, the document key will be inserted where the “:id” is in the post URL. The will allow the correct book to be updated as the key for which document is unique. An example of this URL is shown below.

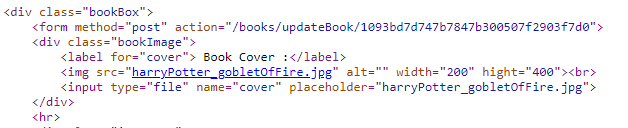


Figure 17 - Source code of book update form

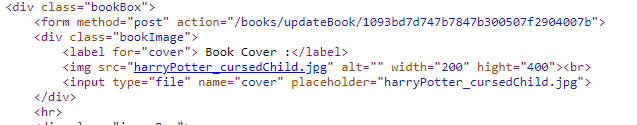


Figure 18 - Source code of book update form

In the above two screen shots of the source code of the update book page, there is an example of two books the can be updated. The first screenshot is for the book Harry Potter and the Goblet of Fire. It can be seen in the form action that the key is different to the second book. For the book to be updated, values will have to be sent back to the database, so it knows what the new data for the document is. With the delete post request there was no new data being sent with the post request so there was no need of multiple parameters. In the image below, we can see how the post request was created.



Figure 19 - Post request for update book

The couchdb.update method will take in two parameters. The first being the database name and the second being an array of what it is that is being updated. As it can be seen above, I have named the variables the same as they are in the database to avoid confusion. The post request will be sent to the database as it is defined in the first parameter along with the id and rev number to distinguish which document it is that’s being updated. Once the book is updated the page will reload along with the data that was just updated. If there are any errors, the error will be outputted to the screen.

## Add Books

I wanted to create an add book functionality for the user to be able to add books as well as update and delete them. The user will be able to choose the “Add Books” on the left of the screen and they will be brought to a page where they will fill out the details of the book as well as supply an image for the book cover. Below is an image of what the user will be prompted with when they click the “Add Book” tab.

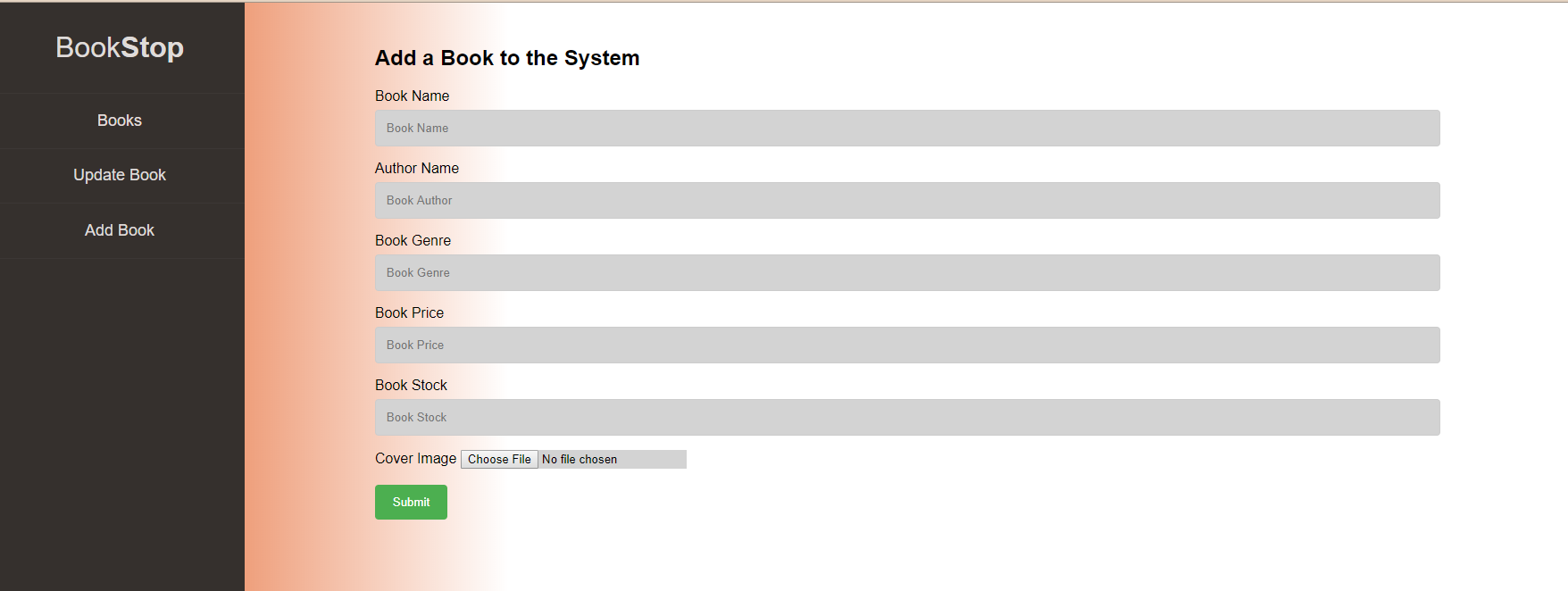


Figure 20 - Add book User interface

For the page to be able to display a get request must be created same as the update books and books page. The code is the exact same as the get request for the home page with the only difference being the URL that is it being directed to. See below image for the code.

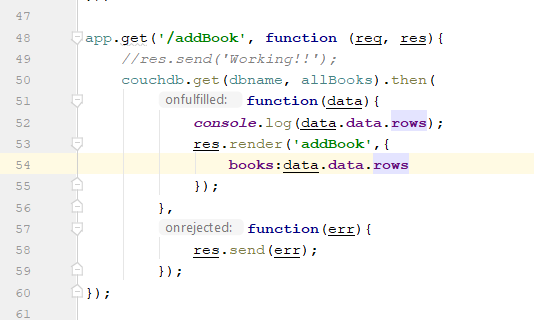


Figure 21 - Add book get request

This get request is just allowing the user to be able to go to the selected page, and for this example it is the update book page. As well as a get request, a post request is also needed to send the details to the database. The post request will, like the update book post request, has two parameters. The first being the name of the database, and for this example I used the actual name of the database instead of the variable to show that it does work this way, and the second parameter being the details that are being sent to the database. As I have more than one piece of data going to the database, I must enclose the variables in curly braces to allow the multiple pieces of data to be sent. Below is the post request code that I wrote in the app.js file.

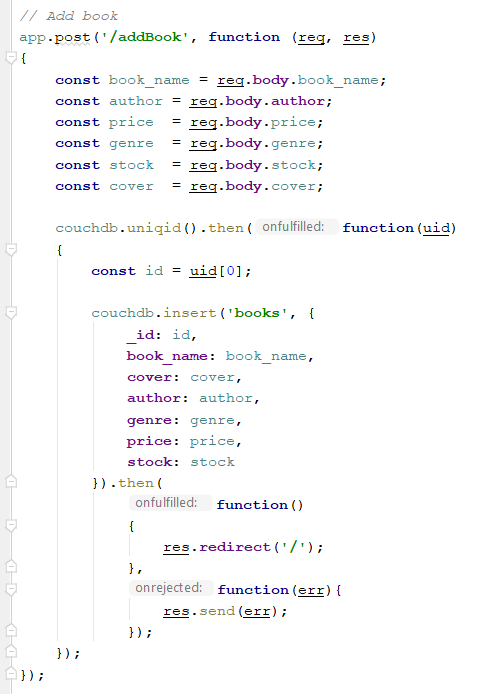


Figure 22 - Add book post request

For an example I will add the book “King Lear” to the database from postman to show how it works. See the image below of a post request to CouchDB from postman adding king Lear to the database.

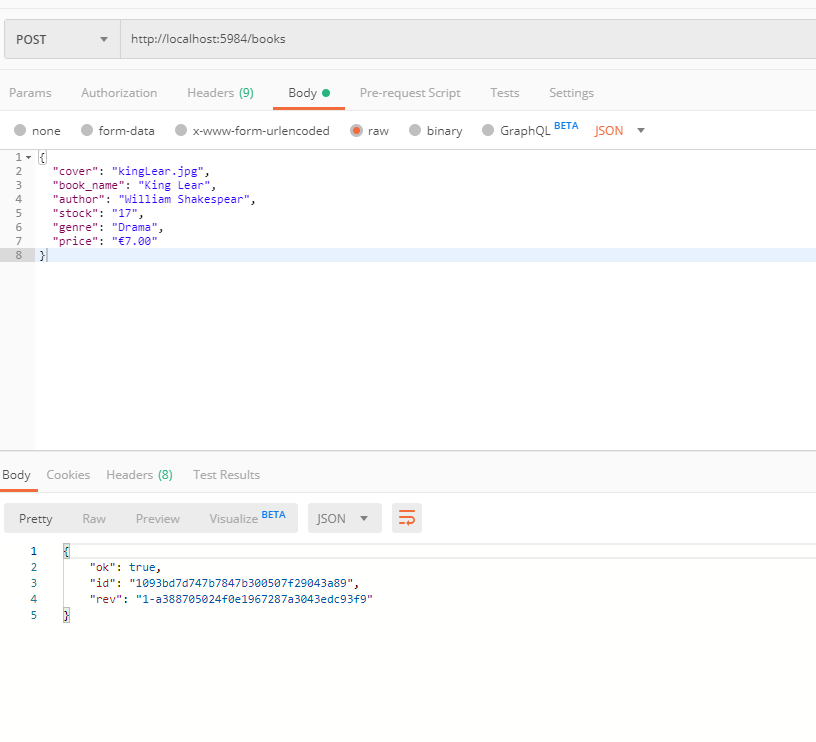


Figure 23 - Postman add book

In the above image it can be seen that the post request to the books database has been successful as we get the response of true. It has been issued with a unique id along with a rev number. To make sure that it has been sent to the database we can check CouchDB. See the below image of the King Lear document that has been sent to the database and on the user interface. It can bee seen that the id from the postman application is the same as the one on the CouchDB document.

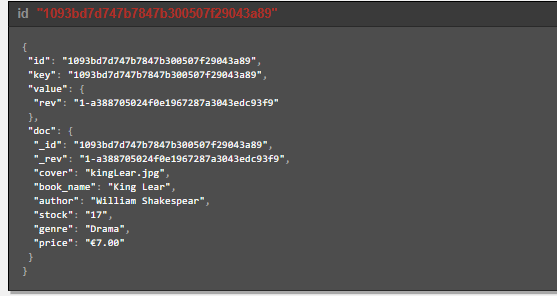


Figure 24 - Document of added book (King Lear)

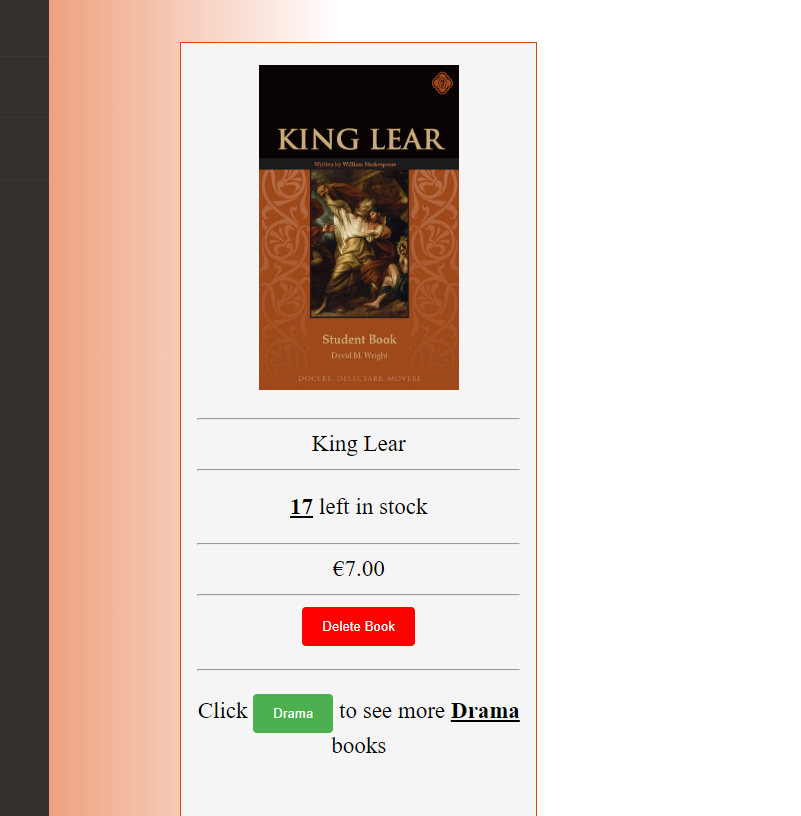


Figure 25 - Book is added to the user interface

## Filter Books

Finally, I wanted to be able to filter down the books depending on what genre was selected. In the home page I included a button at the end of each book that was named the genre of the book. See image below for an example.

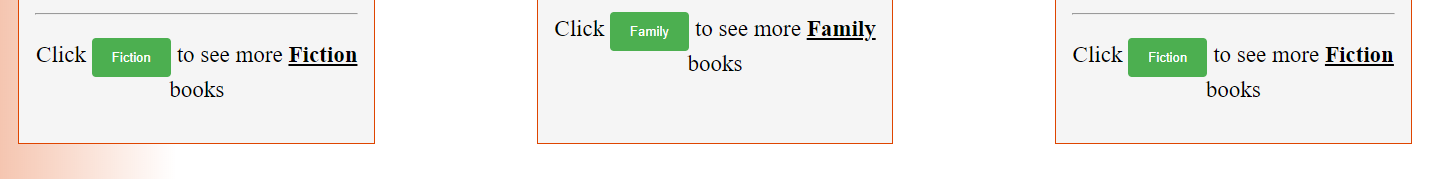


Figure 26 - Genre buttons

When the user selects a button a post request to the database is sent with the genre selected and it will retrieve all the books with the selected genre. This is my example of map reduce. This will allow the user to be able to filter down the books to the genre of their choosing. Below is an image of what will be returned if the “Fiction” button is selected.

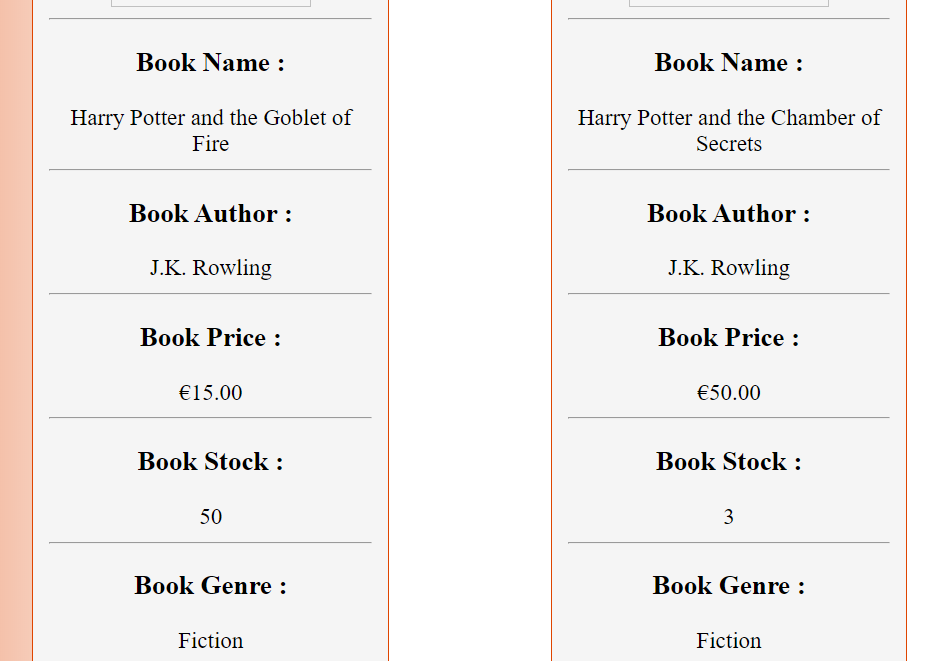


Figure 27 - Genre select page

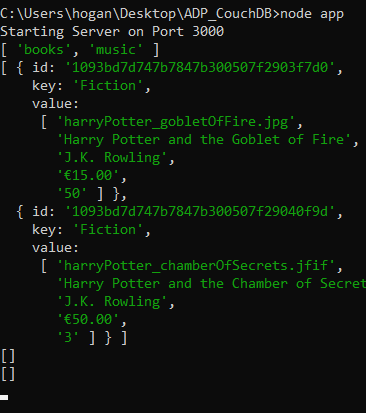


Figure 28 - Command line after genre selected

Again, a get request will have to be created so that the page can be displayed with the books. This is done very differently to the other get requests. Firstly, to be able to display the page with the books on it, a map function must be created to allow a parameter of the genre to be able to filter down the books.

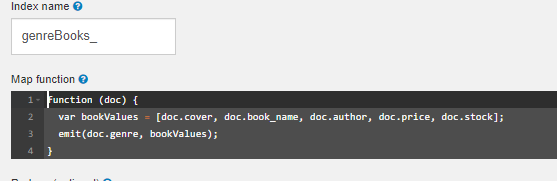


Figure 29 - Map function for genre select

The above function is what will be able to determine what genre is selected and what are the books that have that genre. First, I created an array called bookValues which have all the values of each field in the database in an array. The key in this example is genre and the value is the array. For example, if the key was “Fiction” then the values would be all the books that had a fiction genre. I will show this on a post request from postman to make this clearer. See image below.

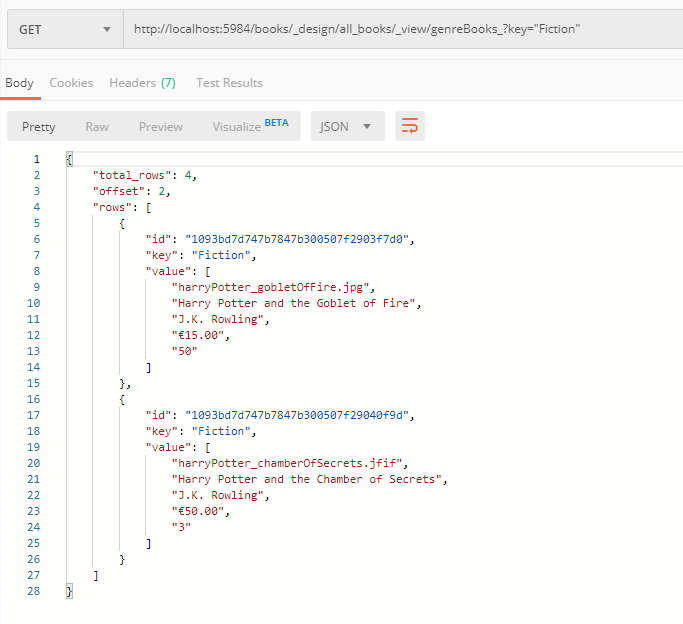


Figure 30 - Postman genre select

In the above image, I have chosen to filter the books by the genre of “Fiction”. It can be seen that there are two books in the database with fiction as their genre. The key for both books is “Fiction” and the value that will be returned is the array I created with the details of the book. Also, I used the reduce option in CouchDB to be able to count how many of each book is on the database.



Figure 31 - Map reduce to show how many books of each genre

This map reduce shows that there are two fiction books and one drama and family book on the database.

To be able to get this data, a post and get request had to be created. I started with the get request to be able to display the data on the page. It required a parameter of genre to be passed in to be able to distinguish what genre was selected. Similar to the update book request, I added a genre parameter at the end of the URL. Once that was done, I had to include the database name along with the view URL. Unfortunately, this wasn’t a simple case of just adding the URL, this time it had to have a value inputted into the URL that could only be done once the user selects a button. I did this by creating a genre variable that would be requested upon the button click and then inserted into the URL. See the image below on how this was done.

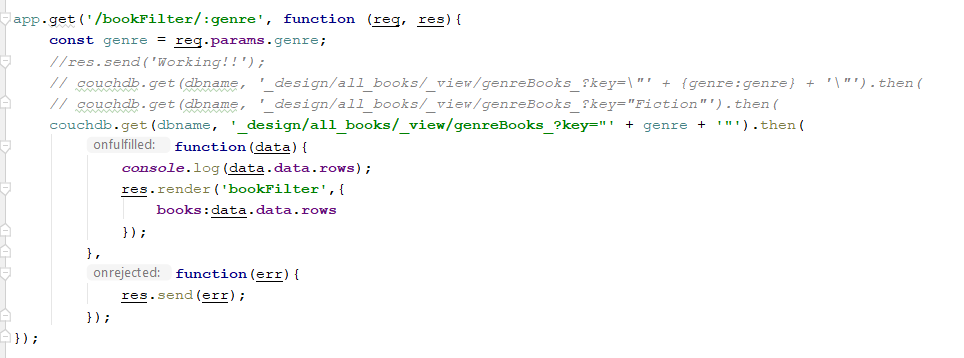


Figure 32 - Get request for genre filter

Once that get request was complete and working, the next task to do was to create the button within a form. This was done very simply by creating a form with a method of get along with an action the same as the get request in app.js. Where “: genre” is in the URL of the get method in app.js, this will be identified as the key in the form. See image below.

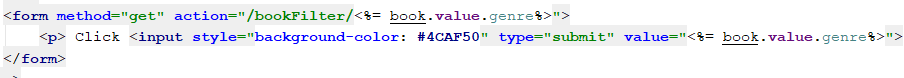


Figure 33 - Form for filter button

As the button is within the loop, each book will have a button with its genre on it. At the end of the action URL in the image above, we can see that the genre value is added onto the end. This will be inputted to the get request in app.js and then the API will be able to get all the books with the selected genre.



Figure 34 - Example of source code for Filter form



Figure 35 - Example of source code for Filter form

The above are snippets from the source code to show how the genre value is stored at the end of the URL

# Conclusion

Overall, CouchDB is a very simple and efficient NoSQL database to use for any application. It communicates very well with Node.JS by just adding “const NodeCouchDB = require('node-couchdb');” to the beginning of the API class. I learned more about how CouchDB works, how to create databases and how to write JavaScript map functions. Not only did I learn about CouchDB, I also learned more about how Node.JS works and how to create an API to communicate with a database. I got to get a deeper understanding on how to use postman to communicate with the database also. To finish up, this was a very beneficial project as I have grown more skills in JavaScript, JSON, research, CRUD, CURL and postman.

# References

CouchDB, n.d. *CouchDB.* [Online]   
Available at: https://docs.couchdb.org/en/stable/ddocs/views/intro.html  
[Accessed 26 October 2019].

Media, T., 2016. *YouTube.* [Online]   
Available at: https://www.youtube.com/watch?v=R6LUMXrAoCE&t=1115s  
[Accessed 25 October 2019].

npmjs, 2017. *npmjs.* [Online]   
Available at: https://www.npmjs.com/package/node-couchdb  
[Accessed 25 October 2019].

Point, T., n.d. *Tutorials Point.* [Online]   
Available at: https://www.tutorialspoint.com/nodejs/nodejs\_introduction.htm  
[Accessed 2 November 2019].

w3schools, n.d. *w3schools.* [Online]   
Available at: https://www.w3schools.com/js/js\_arrays.asp  
[Accessed 26 October 2019].

w3schools, n.d. *w3schools.* [Online]   
Available at: https://www.w3schools.com/howto/howto\_css\_contact\_form.asp  
[Accessed 25 October 2019].

Ziobro, M., 2016. *YouTube.* [Online]   
Available at: https://www.youtube.com/watch?v=tnjv956vdHM  
[Accessed 28 October 2019].