

**Graduate Diploma in IT**

**Level 7**

**GIT701 System Development Integration II**

**Project Report:**

**2.1 Design**

**2.2 Implementation**

**2.3 Testing**

(Worth 50% of final Mark)

**Final Result: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Assessor Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |
| --- | --- |
| **Student Name** | **Diego Evangelisti** |
| **Student ID** | **YIB00002MN** |
| **Student Signature** |  |
| **Student Name** | **Saeed Masoodnia** |
| **Student ID** | **YIB0000306** |
| **Student Signature** |  |
| **Assessor** | **Cassie Po** |

**Table of Content**

**INTRODUCTION…………………………………………………………………………………………..3**

**SCOPES AND OBJETIVES…………………………….………………………………………………..4**

**DELIVERABLES……………………………………………………………………………………………5**

**PROCESSES………………………………………………………………………………………………….6**

**TIMEFRAME AND MILESTONES……………….…………………………………………………..7**

**TEAM MEMBERS: RESPONSABILITIES……….………………………………………………..8**

**DOCUMENTS UPDATED……………………………………………………………………………….9**

RISK REGISTER**…………………………………………………………………………………………….…9**

COST REPORT **…………………………………………………………………………………………….12**

GANTT CHART**……….………………………………………………………………………………………13**

**REFERENCES………………………………………………………………………………………………14**

# 2.1 Design

# Introduction

Based on the fact that the application is made for smartphones, and specifically for apple users, the design was planned to follow some pre-establish templates, buttons, etc that are already proved and tested. All of apple cell phones users are familiar with some design and functionalities and we found interested to keep them, with two purposes:

1. Get user attention and make the app easy to use and navigate throughout all pages.
2. Make our design easier to develop (less time consumer when designing it)

We want the users to find an easy to navigate application, that will allow them to find products and have the change to compare price easily, create lists and find the nearest branch for both shops (Countdown and New World Metro).

The following sections will explain the colour pattern selection, user flow, workflows, mark-ups, backend (includes database explanation, hosting and deployment of the project) and testing part.

**Structure and Navigation**

The following graph shows how the user can navigate throughout the application based on the decision made. It is called the user flow and explains mainly step by step the path the user will take when using the app.

The available pages are:

* **Start screen:** It shows background image related with doing shopping, the application name, a brief description and a button to start using the app.
* **Home screen**. In this page some random products are shown and on top there is a text search to enter the product name the user wants to find (see below: fixed bar on top).
* **List creation**. The first time the user should create a new list so then the product search can start.
* **Add items:** Includes search box to find a product by its name.
* **Decision table**: Compare products that matched with the product name search from Countdown and New World metro. The first products are the cheapest one available in each market.
* Product information: The data for each product is represented here. A big picture of it, price, quantity selection, product name, shop name and a brief description for some of them.
* **Products list:** Shows all the products chosen by the user. Here those products can be managed as follows:
  + - Delete from the list
    - Increment or decrement the quantity
    - The summarized price per each item is shown and the total price from the list as well.

Fixed bar on top:

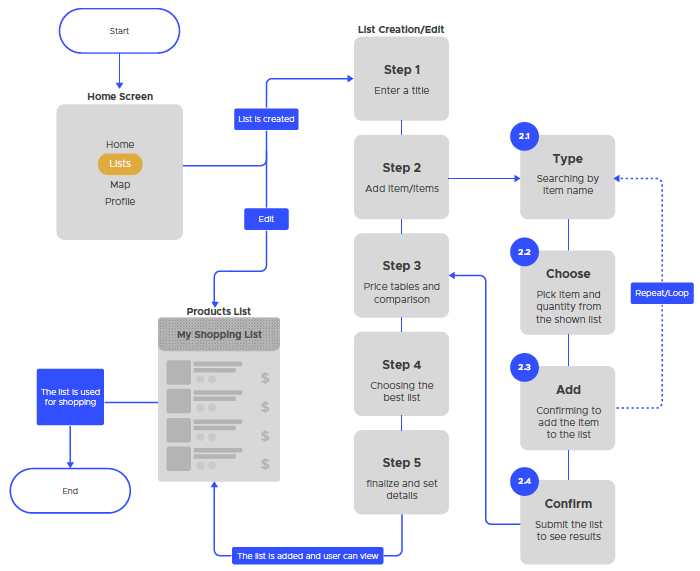
Text search that allows users to look for products by its product name. The products obtained are all that contains that specific word inserted in the search area.

Fixed bar on bottom:

On the bottom of every page there is a fixed bar that contains four options:

* + Home: Takes the user to the Home screen previously explained.
  + List: Show the products list previously mentioned.
  + Map: Show all markers with all branches’ location from both shops in a google map.
  + Profile: For future purposes. Should show the user information.

**User flow**



**System design**

1. **Brainstorming:**

Since the very beginning of this project we performed a brainstorming to come up with the best decisions about UI/UX attributes.

The following pictures show the first approaches taken during the preliminary sections.

Pictures here:

Brainstorming

1. **Paper sketches:**

Once we had our design planned, Saeed started to perform the sketches in order to get closer to our final design. Some examples of it:

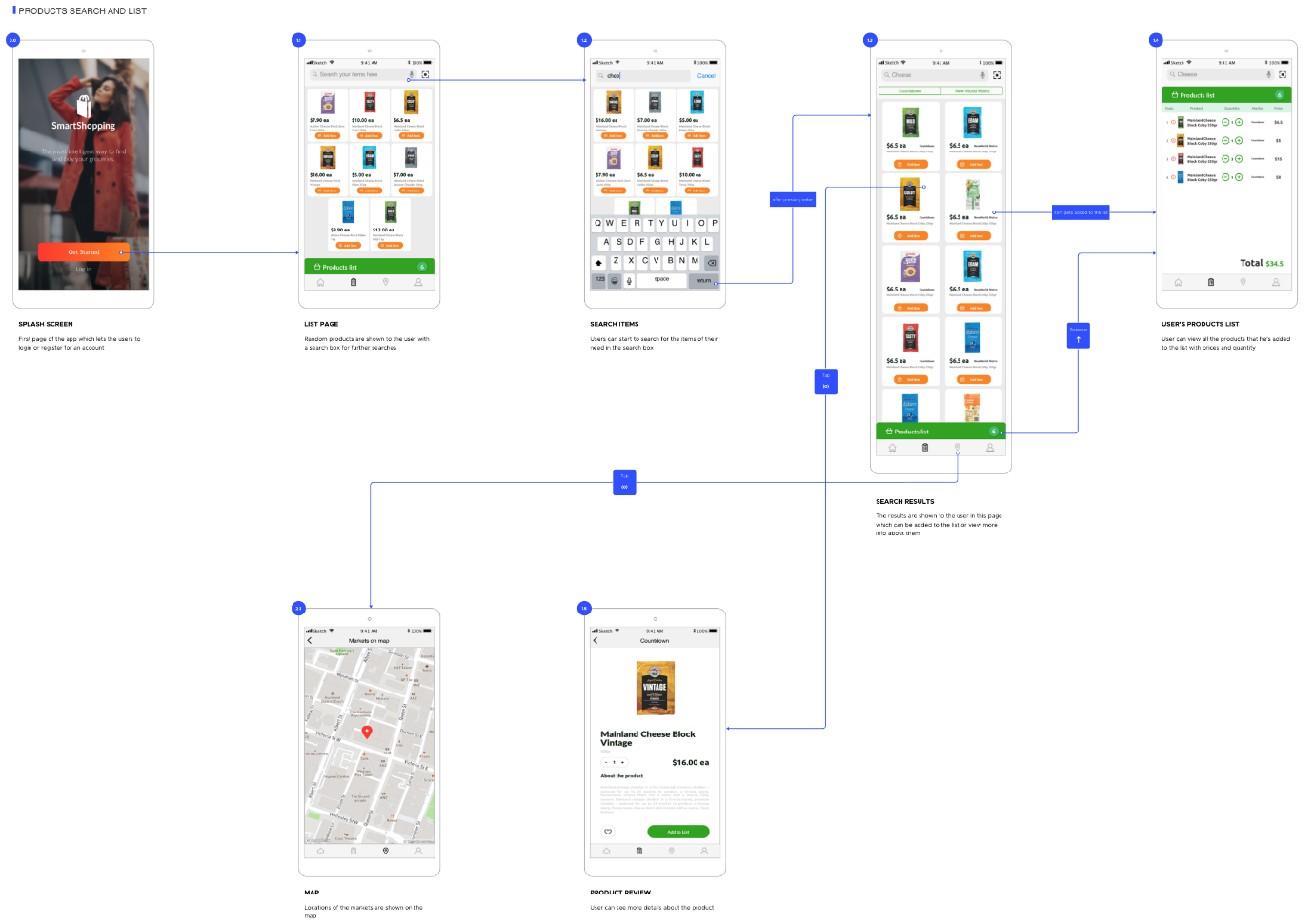
Pictures here:

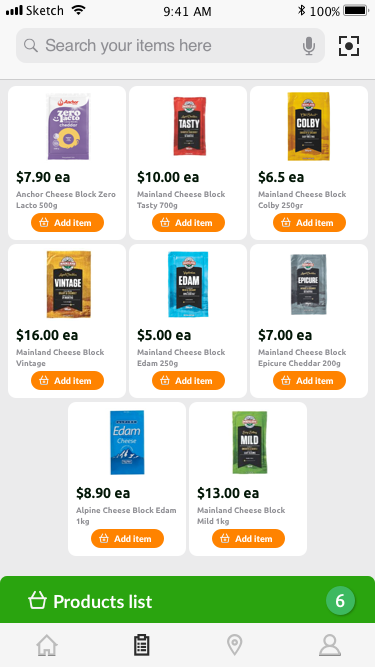
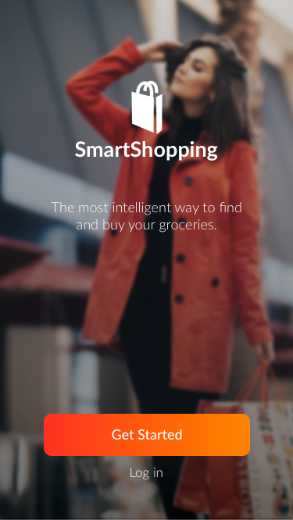
1. **Digital sketches**

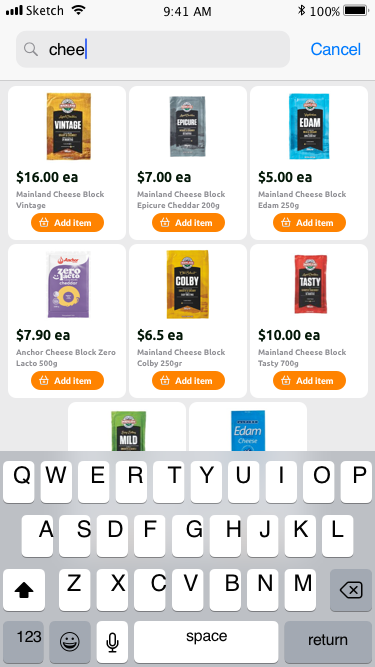
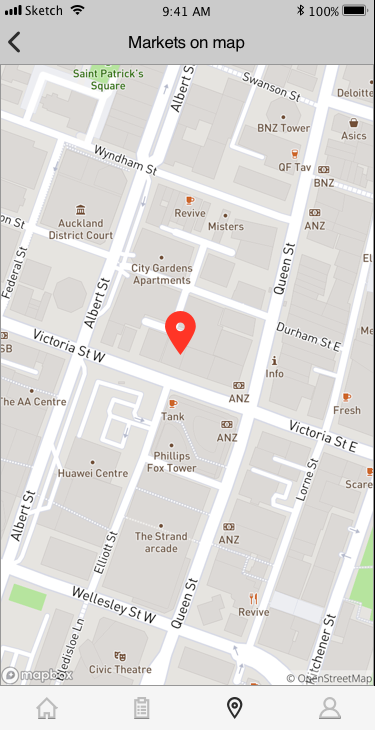
After sketching in paper all draws were translated to digital format using the Sketch software application, a powerful tool for building UI/UX design and saving the data in its own format (sketch format).

1. **Wireframing:**

After some iterations and discussions with the rest of the team, the final design was ready, and this is the result, which includes the wireframing with the user interaction included:

****

****

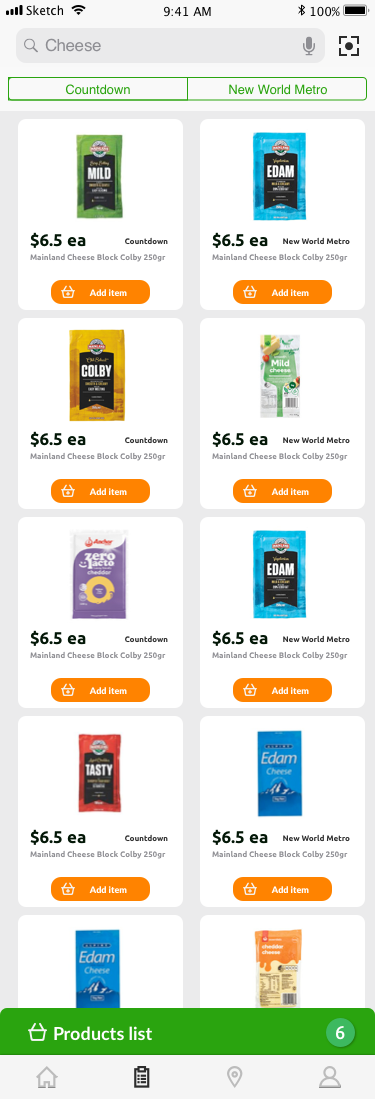
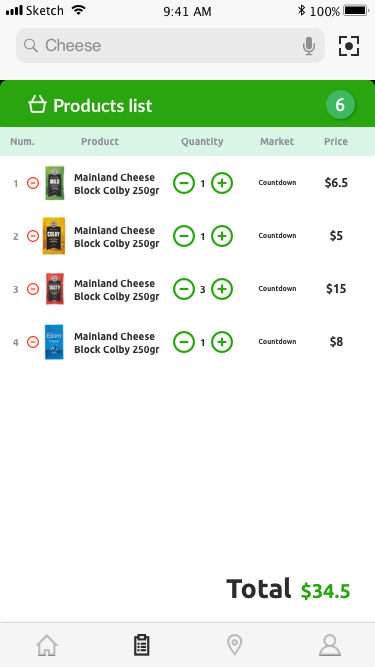
****

**4. Product search**

**3. Product search**

1. **Splash page**

**2. Product search**

****

**6. Product search**

**5. Product search**

**Design Users evaluation:**

The UI/UX was tested by three different people…

[EXPLAIN THE RESULTS HERE]

**2.2 Implementation**

**Functionality:**

* **Front end pages:**

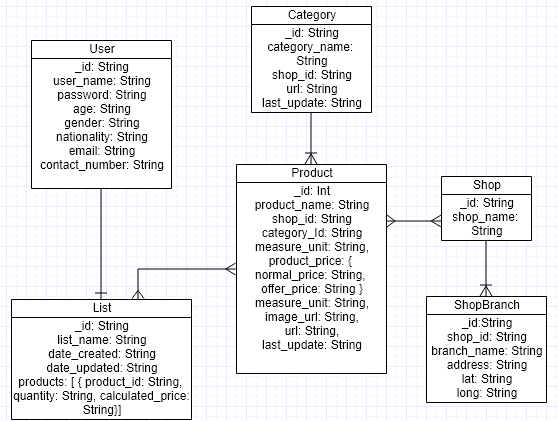
**[ Explain how the front works, tools, etc]**

* **Databases access:**

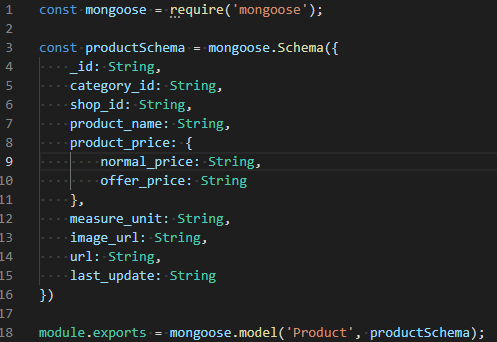
For storing the data, we have chosen MongoDB (non-relational database), and we worked with Mongoose, a library for managing MongoDB in Node.js, the runtime application that uses the JavaScript language for back end purposes.

First of all, an ER Diagram (Entity Relationship Diagram) was performed in order to define all entities and its relationships needed.

The final result:



Following that ER Diagram and using Mongoose schema and models to structure the data, (similar concept of schema and tables for relational databases) each of all models were created. The name taken by those models is Collection, and each one has Documents (records) inside. In order to clarify this concept, we include the following example:



That schema and model is needed to create the data structure for the Product entity and to declare of the all attributes. In the example (taken from our project) the Product model has all the coming attributes: \_id, category\_id, shop\_id, product\_name, product\_price, measure\_unit, image\_url, url and last\_update.

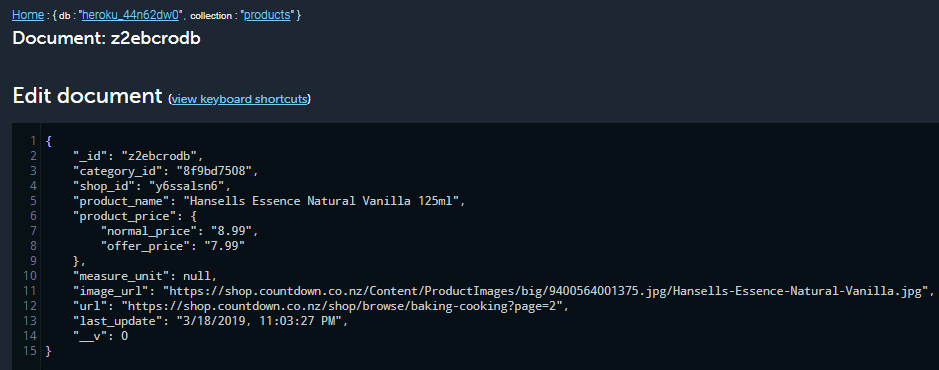
**ID explanation:**

\_id: When a Document is created (this rule applies to all Collections) we are created an ID with very high changes or being unique. The formula uses some JavaScript functions called Math, random () and toString () and reduce the string lenght to only 9 characters.



**Data storage format:**

When Documents are created, the data structure is similar than JSON object ( stands for JavaScript Object Notation). This is really useful for us, because for handling request from the client side to the server we use a RESTful API that mainly sends and receives data in JSON format. As a result of that similarity our operations are simpler.

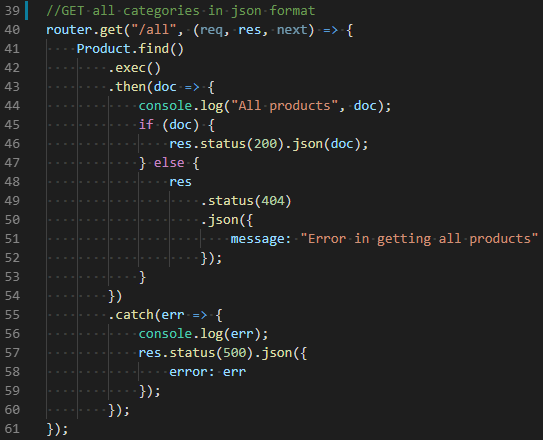


**Document from Products collection, using mLab for MongoDB**

As mentioned before, there’s data transfer between the front-end (client side) and back-end (server side). In other to manage that flow we created a RESTful API, an application program interface (API) that uses HTTP requests to mainly GET, PUT, POST, FETCH and DELETE data.

All routes that the front-end side have to fetch to request data were created in the back-end application and customized with the objective to accomplish all the data interchange. The endpoints (locations from which the API can find the resources they need for their functionality).

This following example is a GET route and the endpoint is the URI where the back-end is hosted followed by “/products/all”

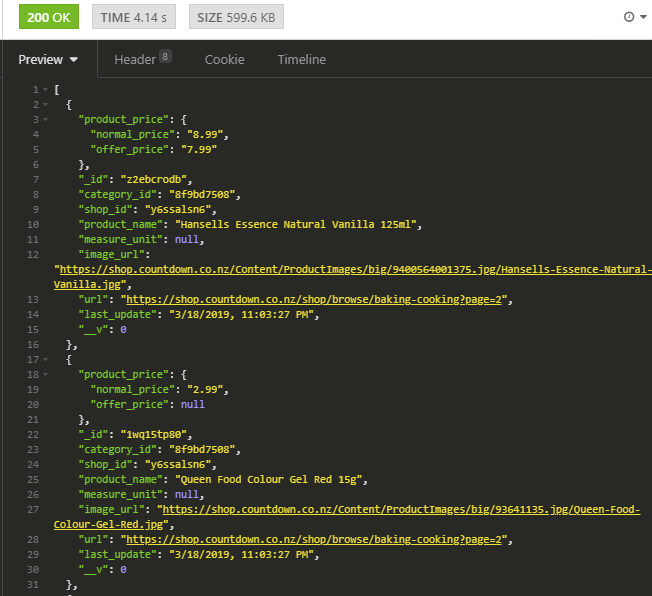


All routes include error handling. So, for that example, when some data is request and for some specific purpose cannot be retrieve, an error message is received by the front-end side to then can take a more proper decision.

For our specific case the complete route is:

<https://smart-shopping-application.herokuapp.com/products/all>

and the result, as shown in the following picture, is all the products from the data base, retrieved in JSON notation.



**All products coming from the database (Software: Insomnia)**

**Web scraping:**

All the products and categories information from both shops are obtained performing web scraping. That means, that using code in the back-end applications we manage to go through the shop’s web-sites pages where the data is shown and then performed DOM manipulation using some packages available for Node.js.

Node.js uses a powerful tool called **npm** (a package manager that contains the biggest library available among all languages created until today). Taking advantage of that, some really useful libraries were used to perform the data scraping:

* + **Cheerio:** Parses markup and provides an API for traversing/manipulating the resulting data structure. Allows to use jQuery to perform DOM manipulation in the back end.
  + **Axios:** Use to perform a GET request to each categories and products URL. Combined with an async/await function gave us the possibility to scrape throughout all the pages without loosing the execution control, considering that Node.js is asynchronous.

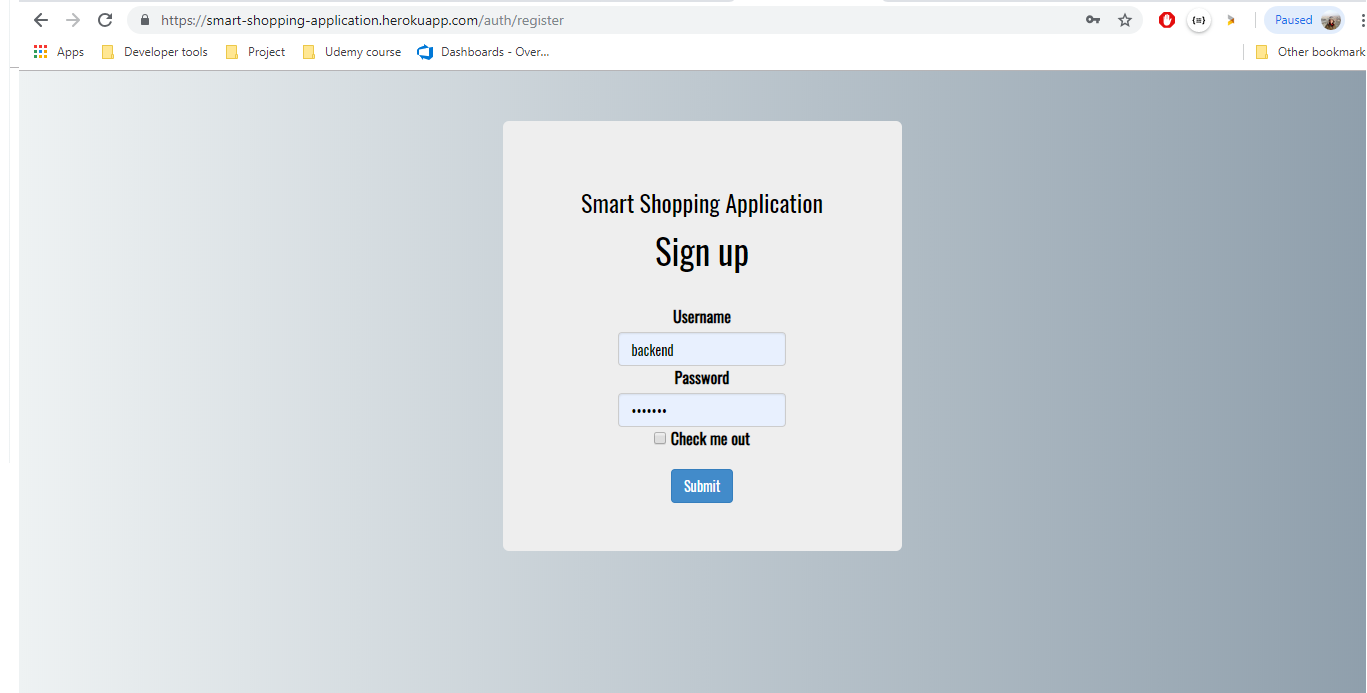
**Back office:**

A back office was created with the main propose of performing the scraping with a friendly view and in a simple way, just pressing a button.

The back office was created in the same repository where the back office is allocated, and we use **ejs** (Embedded JavaScript Templates) to generate HTML markup with plain JavaScript.

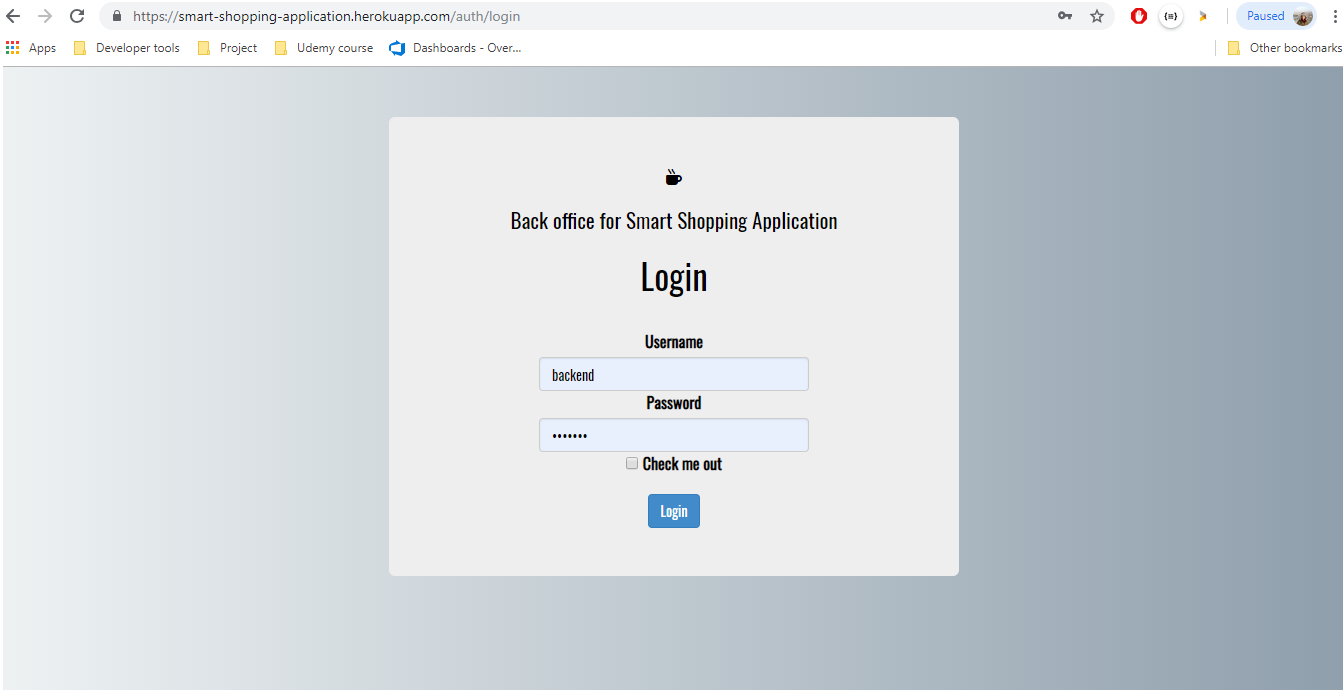
Includes the following pages and functionalities:

* **User registration** (just for administration purposes)



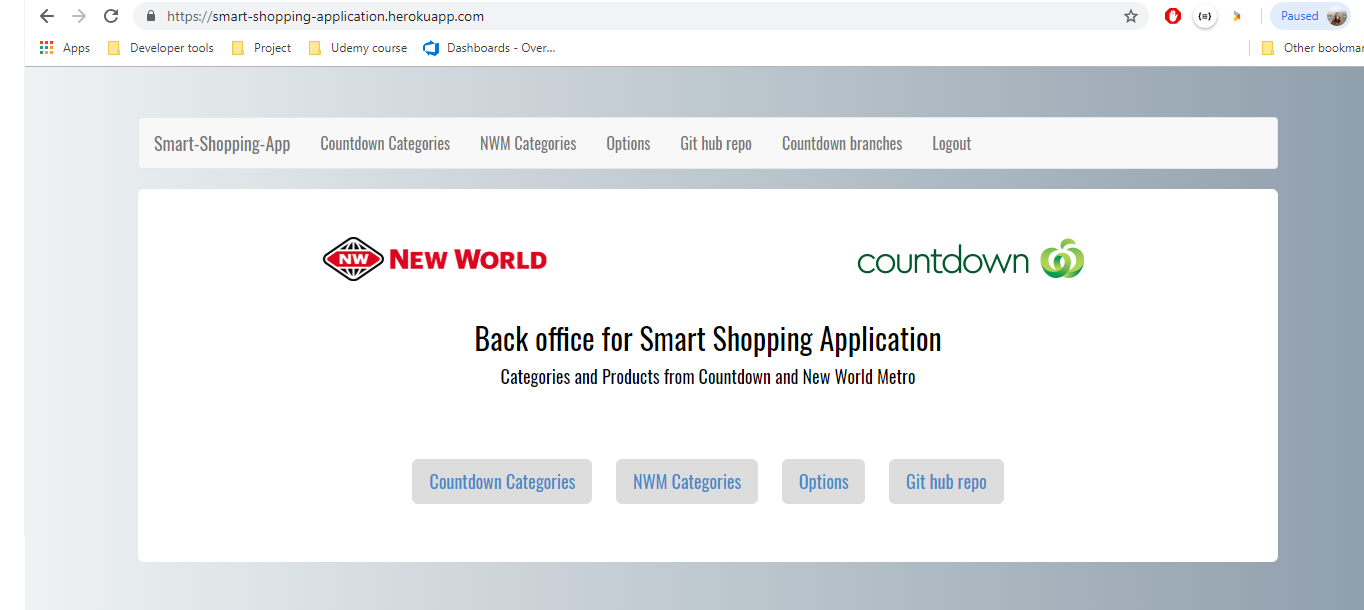
* **Login page:**

User login that uses authentication. The package used for those purposes is called **passport.**

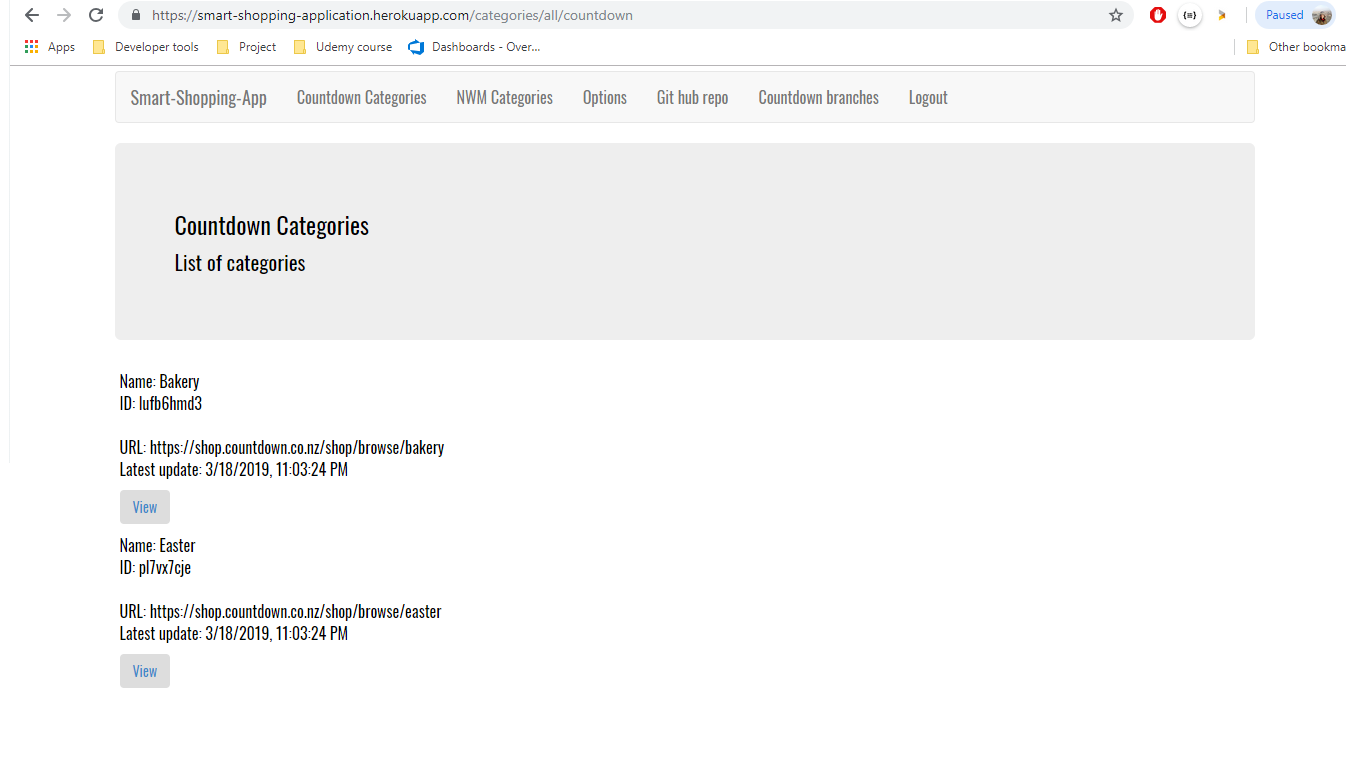


* **Landing page:**

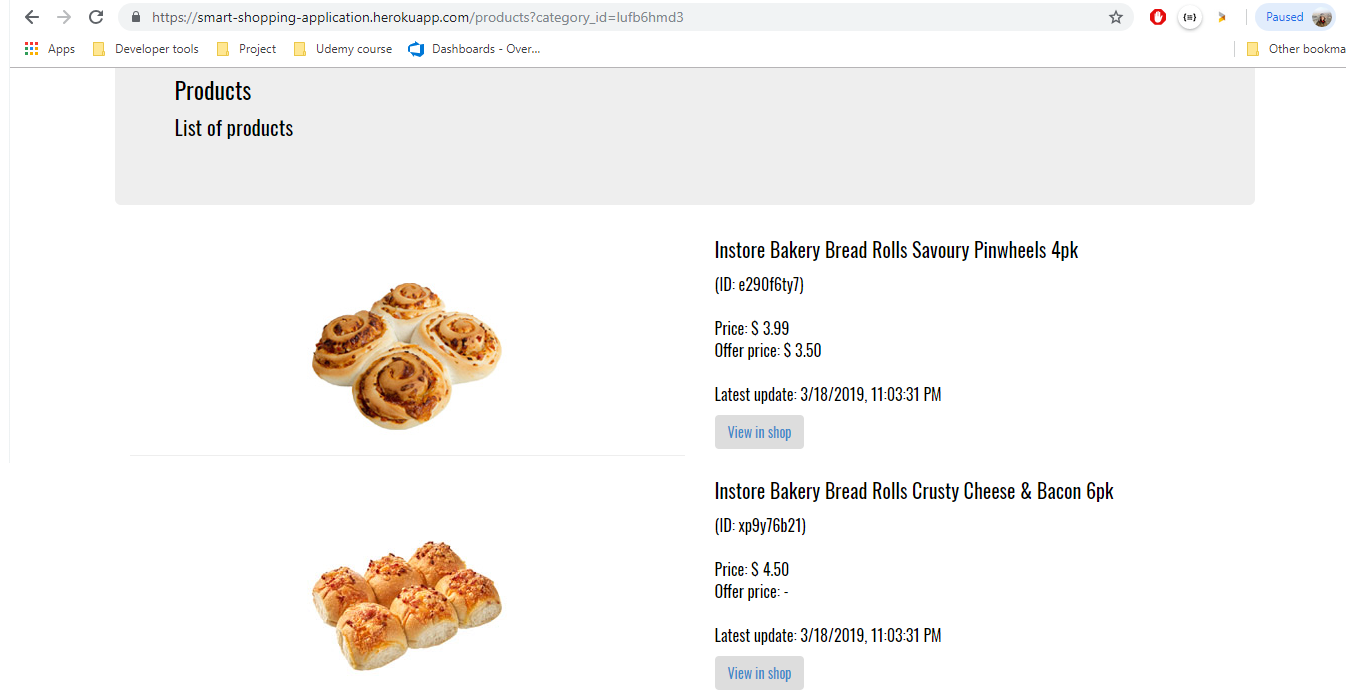
Here the administrator can choose among all the following options, which will render other pages as well:



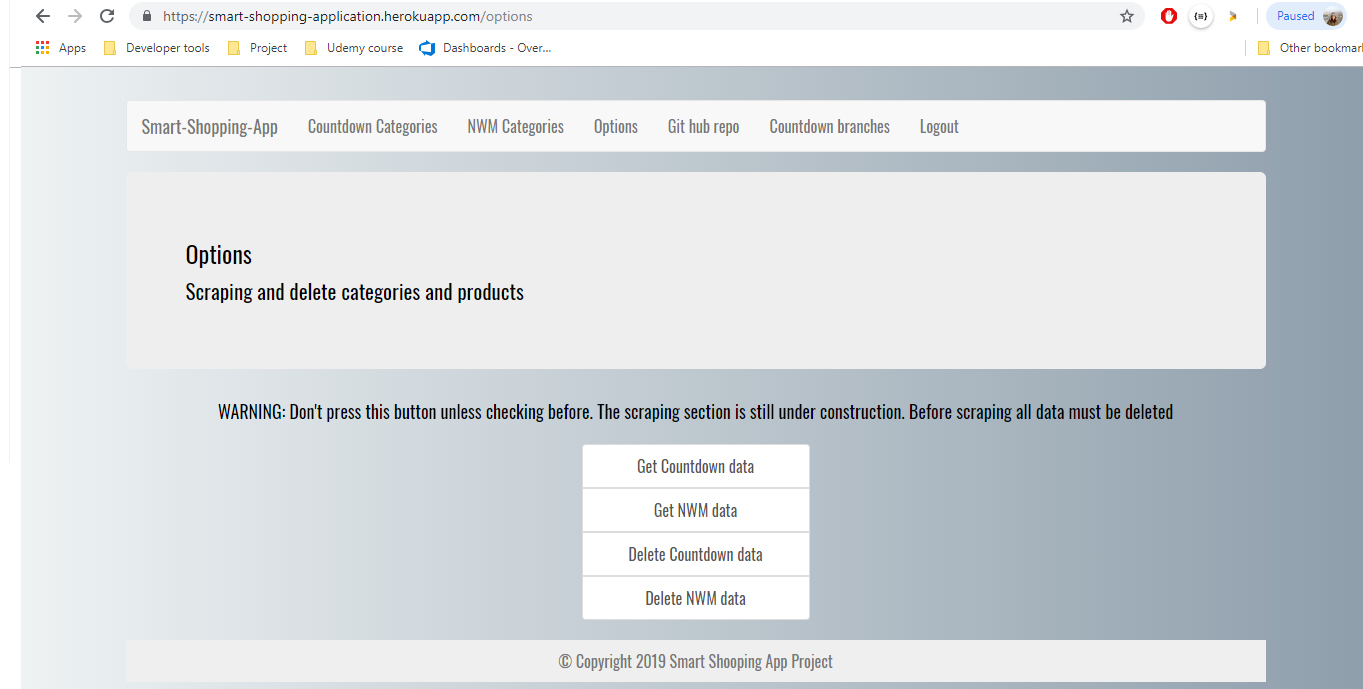
1. **Get Countdown / New World Metro categories (2 different options)**

Both pages show all scraped categories with a link to all their products.

**Categories from Countdown**



**Products from Countdown that belongs to an specific category**

1. **Options:**
   * Get Countdown data
   * Get NWM data
   * Delete Countdown data
   * Delete NWM data

**Options page**

1. **GitHub repository:** Link to the GitHub repository where all the backend files are allocated.
2. **Logout:** Because we use passport to manage authentication, the main routes that render a page are being protect. When the user logout, until it does not sign in again, the access to those pages is denied.

**Other tools for backend purposes:**

* **Heroku:** We use the cloud platform to host our database, using an add-on name mLab, which is basically a MongoDB database. And, what is more important, the platform is the one we chose to deploy our application. We founded really easy to setup, with just giving the GitHub repository link the application is auto-deployed any time that detects any commits in the repo. It is also free or charge (up to 500mb of data storage) which for us is more than enough because we just data.
* **Visual Studio Code:** A great editor that allows us to commit, push and pull easily in our repository, among a lot of attributes that helps to code faster and cleaner, such as customized shortcuts, colour differentiation for reserve words, many PowerShell terminals available, etc.
* **GitHub:** It is the most famous service for version control using Git. It is simple to set up, add collaborators, commit changes, perform pull request, etc. It was really useful for working together as a team.
* **Postman/Insomnia:** Both are famous software use for API testing.
* **jQuery docs:** It was the main source of information to find the best way to performed DOM manipulation at the moment of scraping.

**2.3 Testing**

In our application we perform manual testing in order to find bugs or problems according to our functional requirements. This kind of testing doesn’t require a software, and all parts of the testing are planned and executed manually.

We decided to perform incremental testing because we found it more secure to do and test instead to finish coding everything and then, in case of errors, have to change some pieces of code that can affect the rest and can make the system fail as a whole.

Because in our project a very differentiate front-end and back-end side was defined, most of the test were declare for each part separately, and then more checking was added at the moment of putting both parts together.

All test declared were done, and after finding and fixing the bugs, another re-testing was taking part in order to be sure that the problems found was correctly solved.

**Testing Scope**

As mentioned before, we separate the testing part into front-end-testing, back-end-testing and whole-system-testing. Following there is the scope for each of the three mentioned sections.

In order to clearly separate all the tests from the different sections, we divide all the cases into different main sections and for each testing case we give a Priority status. Then, only High priority case were executed.

Priority status:

**Low** = The result of the test is not affecting so much the rest of the functionalities.

**High** = The test has to be performed and all bugs found must be fixed.

1. **Front-end-testing**

[add front end testing plan here]

1. **Back-end-testing**

# Testing section 2.1: Application and database set up testing

The main idea here is the check that Node.js and MongoDB are installed and working properly, and all packages are ready to use.

|  |  |
| --- | --- |
| **Testing case definition** | **Priority** |
| * + 1. **Node.js and MongoDB are installed properly** | **High** |
| * + 1. The app with the Localhost connection is stablished | Low |
| * + 1. After installing all npm packages the app runs | Low |
| * + 1. **The connection between app and database is establish** | **High** |
| * + 1. **All schema models are well declared** | **High** |

# Testing section 2.2: Web scraping testing

This part is to check if the scraping part is giving some results from a simple task and the step by step get the categories and products information as required.

|  |  |
| --- | --- |
| **Testing case definition** | **Priority** |
| * + 1. **Some data scraped is being show in the console** | **High** |
| * + 1. Categories from Countdown are being storage it in a JSON file | Low |
| * + 1. Some products from Countdown are being storage it in a JSON file | Low |
| * + 1. Get the number of pages per Category in Countdown | Low |
| * + 1. **Get one page per Category and storage it in a JSON file** | **High** |
| * + 1. Set a manual page limit for pages and categories | **High** |

**Testing section 2.3: RESTful API (All of them using Postman or Insomnia)**

We performed this kind of test to be sure that the created endpoints with its corresponding routes are working as expected.

|  |  |
| --- | --- |
| **Testing case definition** | **Priority** |
| * + 1. POST request for categories and products | **High** |
| * + 1. GET request for categories and products | **High** |
| * + 1. POST request to add shops | Low |
| * + 1. PATCH request to edit shops | Low |
| * + 1. DELETE request to erase shops | Low |

**Testing section 2.4: User registration and Login/logout to the back office**

After creating the authentication part, the idea is to check that the Registration, login and logout are all fine. When the user logout, for example, any page can be access unless login again. The user password should be encrypted as it was planned.

|  |  |
| --- | --- |
| **Testing case definition** | **Priority** |
| * + 1. **Register a new user to use the back office** | **High** |
| * + 1. **Log in the new user previously created** | **High** |
| * + 1. Logout from the back office | Low |
| * + 1. Password encryption working | Low |

**Testing section 2.5: Views for Categories, Products and Options in menu bar**

All views should respect the HTML structure, style and functionality and the most important, the categories and products data have to show up.

|  |  |
| --- | --- |
| **Testing case definition** | **Priority** |
| * + 1. **Show categories and products data** | **High** |
| * + 1. Scraping buttons in options section is working | Low |
| * + 1. Delete all data buttons in options section works fine | Low |

**Testing section 2.6: Heroku deployment and mLab database**

|  |  |
| --- | --- |
| **Testing case definition** | **Priority** |
| * + 1. The latest changes are committed to the repository properly | **High** |
| * + 1. The repository is clone is Heroku | Low |
| * + 1. **The back office is deployed in Heroku** | **High** |
| * + 1. The mLab add-on is connected with the application |  |

**Testing section 2.7: All-back-end test**

|  |  |
| --- | --- |
| **Testing case definition** | **Priority** |
| * + 1. **App running in the cloud, routes, views and scraping is working** | **High** |

1. **Whole-system testing**

After the front-end and back-end is ready, both tests individual tests are checked with all bugs solved, now we perform a complete checking of the whole system.

**Testing section 3.1: Front-end and Back-end working together**

|  |  |
| --- | --- |
| **Testing case definition** | **Priority** |
| * + 1. **The app is running with data is coming from the database** | **High** |
| * + 1. **The list of products information is being storage in the database** | **High** |
| * + 1. The map is showing the branches locations after updating them | Low |
| * + 1. **After a new scraping all data requested by the front-end is properly updated** | **High** |

# References

PayScale, 2018. Entry Level Software Developer Salary (New Zealand). Retrieved from: <https://www.payscale.com/research/NZ/Job=Software_Developer/Salary/89c3dc87/Entry-Level>

Udemy, 2019. Complete iOS App Development Bootcamp. Retrieved from: <https://www.udemy.com/ios11-app-development-bootcamp/>

Udemy, 2019. Node.js: The Complete Guide to Build RESTful APIs. Retrieved from: https://www.udemy.com/nodejs-master-class/