School of Science & Engineering

H-NDIR GROUP TASK LIST MANAGEMENT APPLICATION

Final Report

March 3\textsuperscript{rd}, 2018
Capstone Report

Student Statement:

This project is designed with the intention of being a learning path before I start developing my own startup. All the project details included in this report are done by myself under the supervision of Dr. Naeem Nisar Sheikh, and followed ethical approaches. I did rely on different sources to better explain how every technology works and I made sure to cite every external source appropriately both in text and on the reference section. The external sources are used with the aim of allowing students to refer to this work as both a learning material for the different technologies used and as a documentation for the built application. In terms of the methodology followed in this project, I followed the software development steps explained in the Sommerville manual for Software Engineering.

Meryem Mansouri Idrissi

Approved by the Supervisor Dr. Naeem Nisar Sheikh

SUPERVISOR Dr. N. N. Sheikh
ACKNOWLEDGMENTS:

First of all, I would like to express my gratitude to my supervisor Dr. Naeem Nisar Sheikh for his help and support in achieving this project. I really appreciate the patience and assistance he demonstrates towards his students and his dedication to make their learning journey a beneficial experience.

Second, I would also like to thank my family for being supportive and patient with me throughout my learning journey at AUI with its ups and downs. Plus, I am so grateful for all my friends who motivated me to work and aided me during the hard times I went through especially my roommates.

Further, I thank Abdelghafour Mourchid and Ahmed Lekssays for their assistance in finding a good learning path to achieve this project, constantly asking about my work progress and helping me deal with the confusion I faced in my learning. Likewise I express my deepest gratitude to Khaoula Elomrani for helping me proofread this report.

Last but not least, I would like to thank every member of AUI community and faculty who have had a positive impact on my experience in this university.
Outline

ACKNOWLEDGMENTS: 2

1. ABSTRACT: 5
2. INTRODUCTION: 6
3. Project Scope: 7
   3.1. Project Overview 7
   3.2. Market Information 7
      3.2.1. Target Audience 7
      3.2.2. Similar Existing Applications 7
4. Project Methodology 10
   4.1. Feasibility Study 10
      4.1.1. Technical Feasibility 10
      4.1.2. Schedule Feasibility 10
      4.1.3. Legal Feasibility 10
   4.2. Software Engineering Process Model 10
   4.3. Project Management Plan 11
5. Project Requirement Specification 12
   5.1. Functional Requirements 12
      5.1.1. User Requirements 12
      5.1.2. System Requirements 12
      5.1.3. Use Case Diagram 14
   5.2. Non-Functional Requirements 14
      5.2.1. Ethical requirements 15
      5.2.2. Efficiency requirements 15
      5.2.3. Security & Privacy requirement 15
      5.2.4. Usability Requirements 16
      5.2.5. Scalability Requirements 16
      5.2.6. Reliability Requirements 16
13. Appendix A
List of Figures:

Figure 5.1: H-NDIR web application use case diagram

Figure 5.2: Types of non-functional requirements [2]

Figure 5.3: H-NDIR DB Model on Hackolade (for MongoDB)

Figure 6.1: The 3 lives of JavaScript [3]

Figure 6.2: Node.JS for server side use of JavaScript [3]

Figure 6.3: *The concept of MongoDB replication (MongoDB Documentation [10])*

Figure 6.4: Node.js application as microservices (*adrianmejia.com [9]*)

Figure 6.5: H-NDIR system architecture

Figure 6.6: H-NDIR application Activity Diagram

Figure 6.7: H-NDIR application Class Diagram

Figure 6.8: H-NDIR application Sequence Diagram

Figure 7.1: H-NDIR website routes.

Figure 7.2: The asynchronous nature of Node explained [3]
List of Tables:

Table 3.1: List of the best free to-do list applications[1]

Table 4.1: Project Time Plan table

Table 5.1: DB Requirements Specifications

Table 7.1: Folders created to organize the project files.
1. ABSTRACT:

The objective of this capstone is to create a web application that enables users to share a list of items to be purchased or tasks to be completed. The name of the app is taken from the Moroccan dialect, specifically the one used in the region of Tadla-Azilal. It is read as: ash-n’dir [æʃ -n'dir] and has the meaning: “What to do”. The analysis part is focusing on the different technologies that can achieve the major goal of creating a web application and contains information about the tools that have been chosen to complete it. We first focused on features that were implemented, then we worked on the user interface design. The software development model adopted is the incremental one. Different versions of the application have been produced and put to test using different test cases. Based on the multiple test results, I have been able to determine problems with the code, the functionalities implemented, and the connectivity between the web application and the server.
2. INTRODUCTION:

In the workplace, at home or in university, tasks are divided between a group of people and they usually have to do an extra work to update each other about the progress. Our web application is designed to help divide and complete tasks while minimizing delays and spending.

Starting with the division of tasks, there are basic functionalities that will be technically explained within this report. Part of these basic features that contribute to the idea of our application is that a user can create a list and add members to it. The lists’ admins are the only ones allowed to add or delete items from them as our first approach to avoid duplicates. Then, each member can mark their task done once they finish it and it will be updated for all other list members. Elements marked “Done” can be removed or kept within the list depending on what the admin decides.

Moving to explaining the part of minimizing delays and spending, it is necessary to clarify that the app can be looked at from two perspectives. The first one focuses on the use of the app as a to-do list or a task divider. The second perspective considers the app as a shopping list.

In the first case, it is easier for all members to be aware of the required action and start it right away. List elements can be updated in a real-time manner. When it comes to the economic side, our app is designed to save its users the costs of the different calls and text messages either to set a meeting, divide tasks or discuss the progress of their project. For the second case, the users will be able to know the necessary items to shop for without information duplicates, and choose/notify the elements they can take care of. On the economic side, again we’ll be saving our users the cost of calls, the cost of having the same item bought twice or forgetting something and driving back to the store to get it. Of course, in both cases the lists’ admins and members have different functionalities they can use like adding, deleting and modifying list items.

Our app targets students, families, groups of people living together (i.e roommates), etc. This application is offering a great tool to people looking for efficiency in their daily activities. It will also allow people living Kilometers away from stores to get what they need as well as save money and time.
3. **Project Scope:**

3.1. **Project Overview**

This capstone project is in the form of a web application that can easily be transformed into a mobile application. It creates a user-friendly interface for multiple shared to-do lists. The user can manage his/her own tasks along with the work that must be accomplished within a group. The structure of the application makes it easier for users to employ these shared lists for other processes. Additionally, the application structure is a basic platform to which we can easily add further features. The name as already mentioned is H-NDIR. The explanation of this term and the reasons behind choosing it is specified in the introduction. From the time-constraint perspective, this project is a capstone project during the Spring semester of 2018.

3.2. **Market Information**

3.2.1. **Target Audience**

The targeted audience is not limited. This application is designed for public use and provides coordination between different list members. However, the implementation technologies restrict the use of this application to users who have access to internet connection and basic familiarity with working with filling text fields and using interactive websites.

3.2.2. **Similar Existing Applications**

Similar applications are available online and either are mobile applications or are in other forms but can fulfill the goals set by H-NDIR app. According to TheGuardian.com [1], a collection of the best free to-do list apps are shown in the table below.
Table 3.1: List of the best free to-do list applications[1]

<table>
<thead>
<tr>
<th>Platform</th>
<th>Internet Access required?</th>
<th>functionalities</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wunderlist</td>
<td>YES</td>
<td>- Creating and sharing Lists</td>
<td>- More functionalities are in premium version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Creating reminders</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Has calendar for due dates</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Can create categories</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Premium: Files sharing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Google Keep</td>
<td>NO</td>
<td>- Has Google docs options to manage documents and list items</td>
<td>- Cannot be shared</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Todo-ist</td>
<td>YES</td>
<td>- Creating and sharing Lists</td>
<td>- More functionalities are in premium version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Creating reminders</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Awards point as a motivation feature for users</td>
<td></td>
</tr>
</tbody>
</table>

[1] Source: [1]
<table>
<thead>
<tr>
<th>App</th>
<th>Platform</th>
<th>Availability</th>
<th>Features</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any.do</td>
<td>Android/iOS</td>
<td>YES</td>
<td>- Creating and sharing Lists&lt;br&gt;- Has voice-entry features&lt;br&gt;- Helps in focusing on the urgent tasks and has a reminder</td>
<td>- More functionalities are in premium version</td>
</tr>
<tr>
<td>Remember the Milk</td>
<td>Android/iOS</td>
<td>YES</td>
<td>- Creating and Sharing Lists&lt;br&gt;-Synchronizing across devices&lt;br&gt;-Setting priorities and deadlines</td>
<td>- More functionalities are in premium version</td>
</tr>
<tr>
<td>Habitica</td>
<td>Android/iOS</td>
<td>YES</td>
<td>- Lists are local and individual (not shared)&lt;br&gt;-Tracking and completing task is made as a game (battle)</td>
<td>- Can be seen as not serious</td>
</tr>
</tbody>
</table>
Wunderlist is built upon a Microsoft service named Microsoft To-Do. This application can be used on the user’s phone (Windows/Android/iPhone) as well as on the web or other mobile devices.

The applications that can indirectly achieve the same goals set to our application are to-do list applications or shared documents like Google Drive, Onenote or Evernote as well as more experienced freelancers. Anyhow, we do have the advantage of providing a free service.
4. Project Methodology

4.1. Feasibility Study

4.1.1. Technical feasibility
Technically, this project is achievable. Among the technologies used I mention: Node.js, Express.js, MongoDB, Vue.js, CSS, HTML, jQuery, Backbone and Cordova. None of these technologies was introduced in the Computer Science major courses, nevertheless, the different programming competences and software development skills I acquired through my study at AUI allowed me to consider these tools. Regarding the development process of the application, I started by creating a responsive web application. The substantial documentation available online on how to use the aforementioned technologies as well as the software development method I chose made it easier to learn, develop, test and fix bugs.

4.1.2. Schedule Feasibility
This project has been elected feasible since the first week. The development started on the first week of February until the first week of April. The rest of the time was allocated to designing the front-end and testing, however, due to the amount of learning that is done, it might not be completely functioning in terms of User Interface.

4.1.3. Economic Feasibility
The cost of this project is minimal. All the software used is free and our database is on one of the free hosting websites: MongoDB Atlas. All the development steps of this web application are completed by myself making it cost-free. With regard to users, they will need to have access to internet. This requirement is not hard to achieve since many users have their own internet subscriptions or use Wi-Fi. Thus, we conclude that our application is economically feasible.

4.2. Software Engineering Process Model
By definition, “a software process is a set of related activities that lead to the production of a software from scratch in a standard programming language” [2]. Four activities are mandatory to
every software process, they are given a section for each on this report. We will start with
Software specification, followed by a section for software design and implementation, then
software testing, and last but not least the software evolution in the future work section. This
project is based on agile methods, the incremental model is more suitable as it allows us to create
initial versions upon which we can have users’ feedback and determine the specification that
needs to be updated for the intermediate versions then the final version. This method is more
favorable as the update of user requirements is reduced and documentation is not redone from
scratch. Besides, development and deployment are done rapidly and the customer can give
feedback even though not all functionalities are implemented.

4.3. Project Management Plan

Table 4.1: Project Time Plan table

<table>
<thead>
<tr>
<th>Week</th>
<th>Agenda</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Project selection</td>
</tr>
<tr>
<td>2</td>
<td>Initial Specification.</td>
</tr>
<tr>
<td>3</td>
<td>Feasibility Study and Analysis</td>
</tr>
<tr>
<td>4-7</td>
<td>Requirement Specification &amp; Design</td>
</tr>
<tr>
<td></td>
<td>Interim report submission and technologies</td>
</tr>
<tr>
<td></td>
<td>learning</td>
</tr>
<tr>
<td>8-12</td>
<td>Implementation Pt 1 &amp; Testing</td>
</tr>
<tr>
<td>12-14</td>
<td>Implementation Pt 2 and Testing</td>
</tr>
<tr>
<td>15</td>
<td>Final Report submission, front-end design &amp;</td>
</tr>
<tr>
<td></td>
<td>Testing</td>
</tr>
<tr>
<td>16</td>
<td>Project Defense</td>
</tr>
</tbody>
</table>
5. Project Requirement Specification

5.1.1. Functional Requirements

We will start by the different services our application should provide along with how it should behave to the different user inputs.

5.1.1.1. User Requirements

The list of user requirements is as follows:

- The user should be able to register with an unused username.
- The user should be able to register with an email address only once.
- The user should enter the password and re-enter it for confirmation.
- Each user with an account shall be able to access his/her profile and Lists.
- The user should be able to logout and unsubscribe.
- The user should be able to update their profile and either save or ignore the changes. (The user shall not be able to change their email address to one associated with another account)
- The user can create lists, add members and elements to them. (They will be these lists admin automatically)
- The user can update information, add, modify or delete users and elements of lists of which they are admin.
- The user should be able to add elements to the list of which he/she is a member.
- The user shall be able to mark elements of the list as “done” or “undone”.

5.1.1.2. System Requirements

- The system shall not allow any route to be executed when no user is logged in except for: register, login, unsubscribe, change password and contact us.
- The system should signal an error if the user tries to login with erroneous.
- The system should signal an error to the user if they try to register with an existing email or username.
• The system should inform the user through an error message if the password and confirmation password are not matching during registration.

• The system should check user data against the DB to login

• The system should load: the lists a user administrates and/or is member of, and his/her profile information.

• The system should update the corresponding DB collection(s) each time a user saves a change.

• The system should display the profile form with text boxes having old information that can be filled and saved.

• The system should not allow a user to change their email or username to one that is already used.

• The system should allow Admins only to update users and delete the list and view the list activity log. A violation of this rule ignores the change and sends an error to the violator.

• Once a user creates a list, the system should directly assign them to be the admin of this list.

• When a user logs out, all local variables are flushed, and they cannot access any user functionality until they login again.

• The system should allow all registered users to unsubscribe.

• When a user unsubscribes (by entering their email):
  ○ All the lists of which they are admin are deleted.
  ○ They can access the app user functionalities when they register again.
  ○ All their data is removed from the DB unless they agreed on sharing anonymously their data with the application owners.
5.1.3. Use Case Diagram

Figure 5.1: H-NDIR web application use case diagram

5.1.2. Non-Functional Requirements

Non-Functional requirements have many sub-requirements explained in the Figure below. However, This section will take into consideration the sub-requirements that have an effect on our application at this point in time
5.1.2.1. Ethical requirements

- The user data should not be used by any party unless the user agrees on that and is aware of that use.
- The application should be clear about any expenses the user will have to pay.

5.1.2.2. Efficiency requirements

- The application should create a good user-experience
- The application should not exceed 6 seconds to launch [7] in a normal device that does not have problems.
- Data retrieval from the DB should be fast and reliable.
- The application and DB connection should be able to recover from errors rather than crash in case of a bug.

5.1.2.3. Security & Privacy requirement

User data should be kept safe and must not be shared with any third party without the consent of its original owner.
5.1.2.4. Usability Requirements
This web application should be user-friendly and allow users to access all its features with an easy navigation interface.

5.1.2.5. Scalability Requirements
The web application should be scalable by using the latest technologies that allow a quick and reliable update of the different features and DB elements.

5.1.2.6. Reliability Requirements
The application should be able to recover from a problem by either solving it or sending a clear error message to the user. In extreme cases, the user should be able to report the issue they faced with the app.

5.1.2.7. Maintainability & Extensibility Requirements:
The application should be coded in a clear manner allowing the individuals working on it or maintaining it to understand the algorithms used. The way this web application is implemented should include enough documentation to ensure quick future growth.

5.1.3. Database Analysis
The type of application we are building has a fast rate of data growth. This latter is expanding due to the increasing number of users, the number of lists, but most importantly the number of elements in each list. Knowing that information about items added to lists must be kept as well to make sure that non-allowed operations do not happen; it is a must to verify that all data is stored. Still, there is a trade-off between the size of data and the time of its retrieval from the database.

For this and more, I selected the document database as the most suitable database model for this project on the short and long run. I chose data to be stored on a remote database rather than locally for two reasons: The first is the fast growth of the user lists that would affect the efficiency of the application, and the second is that the aim of the application in creating shared
and real-time updatable lists does not require the user to store data locally. A possible update would be to do something similar to what the “Skype” application opts for: keeping part of the latest data locally, while the older information is stored on the database. Once the user logs in, we check the database data against the locally saved one to know about updates and eventually notify the user about them.

5.1.3.1. Collections Diagram

![H-NDIR DB Model on Hackolade (for Mongodb)](image)

This database format might infer the existence of redundancy in our DB, yet is the principle of denormalization. Data is stored in more than one entity. For instance a list might be stored in the lists collections as well as within the admin’s object. The use of denormalization is to save time of fetching user data. When a list is updated all its copies in the DB will be modified too.

H-NDIR database requires four main collections: users, admins, lists & items.

The entire database designed is hosted using Mongo Atlas cluster and interacted with using the MongoDB compass interface. On the level of the code, our DB does not require Mongoose since
we connect using Mongo driver for Node directly to the online cluster to create elements. The relationship between different entities is clarified in the statements below:

- Each user **has** an account
- Each user **has** a profile
- Each user **is** a member of **one to many** lists
- Each list **has One** Admin
- Each list **has** two or more members
- Each list **has** one to many elements
- Each list **is** administered by one member at a time.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Input</th>
<th>Output</th>
<th>Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td>The user should be able to log into his/her account using his/her email or Username and password</td>
<td>Email or Username Password</td>
<td>If the information provided is correct, he/she can successfully access their account. An error message should be displayed when information is not valid</td>
<td>Comparing the information entered by the user against the information in our database</td>
</tr>
<tr>
<td>Logout</td>
<td>The user can log out from his/her account</td>
<td>None</td>
<td>After logout the user is directed to the main page</td>
<td>Close the user’s account and go to main page</td>
</tr>
</tbody>
</table>

Table 5.1: DB Requirements Specifications
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Process</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsubscribe</td>
<td>Each user has the right to unsubscribe from the service</td>
<td>Asking the user if they really want to unsubscribe. If they do, we delete their data from the database</td>
<td>Delete the user row from the list of users and remove dependencies of that user account from the database.</td>
</tr>
<tr>
<td>Reset password</td>
<td>If the user forgets his/her password, he/she should be able to reset it.</td>
<td>If two entries are matching, reset password. Else, send an error</td>
<td>Text message sent to the user confirming that the password is reset.</td>
</tr>
<tr>
<td>Create Group</td>
<td>The user can create a new Group and add members to it</td>
<td>New list added to the user’s tasks lists</td>
<td>The request should be sent to the DB to create a new row in the user’s tasks’ list and make the dependencies required with the other tables</td>
</tr>
<tr>
<td>Update Group List</td>
<td>The user can update the group information</td>
<td>List Updated confirmation</td>
<td>If successfully done, the update is saved in the database</td>
</tr>
<tr>
<td>Add Member</td>
<td>The user must be able to add a member to a list</td>
<td>Confirmation</td>
<td>Change of the selected fields in the Database</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Action</td>
<td>Result</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Quit Group</td>
<td>The user should be able to quit groups selected</td>
<td>Selected list to delete</td>
<td>Confirmation</td>
</tr>
<tr>
<td>Take care of list</td>
<td>In a shared list, only one user can mark the list as taken care of</td>
<td>None</td>
<td>Marked as taken care of by that user</td>
</tr>
<tr>
<td>Done</td>
<td>The user who took care of the list can mark it as done when they finish their work</td>
<td>None</td>
<td>Elements marked are removed from the list</td>
</tr>
<tr>
<td>Update Profile</td>
<td>The user can update the fields available in their product</td>
<td>Field Input</td>
<td>Confirmation sent</td>
</tr>
</tbody>
</table>
6. Project Design

6.1. Technology enablers

To develop this web application, I chose to use JS technologies. Many reasons backup this choice. First these technologies are starting to grow and are frequently used within the web development world.

![Figure 6.1: The 3 lives of JavaScript [3]](image)

Besides, it was an opportunity for me to both challenge myself and expand my knowledge on programming languages. By the same token, documentation and solutions to frequent bugs are available online on websites such as stackoverflow.com. Furthermore, most of the projects I worked on were computer software products and this was a great opportunity for me to explore the web development world. Another important reason is the ability to automatically make a web application, using different JS technologies, a cross-platform phone application using Cordova Apache, and that is one of the future work goals.
Below are the different technologies used to develop our application both on the front and back-end.

6.1.1.  **Server Side**

On the client side, we used the following technologies:

**Node.js:** It is a low-level *environment* but not a language or framework, it is actually an engine that helps run the JavaScript code on browsers embedded in a C++ code with new objects resulting in what we call now Node run-time environment [4], however, to avoid the repetitive tasks caused by the nature of Node, **Express.js** will do the work through the different libraries it provides. As explained in the Node.js course on “OpenClassroom” website, instead of using multiple languages for submitting the request from the client to the server and the response from the server back to the client, we rely on JavaScript also called the “english” of programming languages. Another advantage of using Node.js is that it is lightweight and does not require compilation. The feature Node.js brings is the ability to use JavaScript on the server-side as well to generate the **HTML** pages instead of needing a server language (i.e. PHP) to do that.

![Diagram](image)

**Figure 6.2:** Node.js for server side use of JavaScript [3]

**Backbone.js** was supposed to replace **jQuery** to create or update the HTML script. I had to switch to Backbone since Node.js does not support jQuery to interact with the **DOM** and
Backbone.js is designed to create HTML script to render from the JavaScript code. Nevertheless, after I learnt about Vue.js I automatically switched to it for several reasons detailed later.

For the second type of request responses, I used **MongoDB** for a database. It is a document based DB providing different functionalities. Besides the Atlas free cluster, I have been able to build a remote DB that can be accessed from anywhere instead of a local one. A document based DB was opted for because of the data normalization that fits the data-growth of our application and it ensures real-time data update (if internet connection is available). For the different collections’ **Schema**, please refer to appendix A. Another reason for using MongoDB is high availability. By the multiple copies of data it makes, it insures data availability even after a loss of a single database server by checking the primary database and once it fails, one of the copies is elected as primary.

![Figure 6.3: The concept of MongoDB replication (MongoDB Documentation [10])](image)

Concerning the limits of our DB, they are directly linked to the thresholds and limits set by MongoDB creators. A concrete answer about the number of connections allowed using a
MongoDB wasn’t found online. However in Quora.com many other users seem to have the same question about parallel connections to the DB as we do. As a user of MongoDB for web development and member of the discussion about MongoDB scalability on Quora, Yi Lin explains that the range of possible connection starts at 1000 up to 10000 connections. Considering that a web server would make a minimum of 5 data fetches and maximum of 20 fetches per user session, this implies that an application based on MongoDB can have roughly a minimum of 50 parallel users and maximum of 2000 ones [8].

Referring to a blog post about Node.js scalability, depending on the hosting method used. For a local host, one user is allowed to use the app. On a single server, it becomes possible for up to 9 concurrent users to work with our web application. Using vertical scaling, which refers to increasing the capacity of a single server hardware with more resources, and this allows between 10 to 99 concurrent users. The following method is horizontal scaling that means having multiple instances of the app on different server hardwares. This later solution is efficient in avoiding I/O bottleneck problems. It also allows between 100 and 999 concurrent users. For larger web applications may use a multi-server method that can allow more than a 1000 concurrent users. In this later case, a multi-server database needs to be implemented as well. Right after comes the microservices. It is based on dividing an app into small independent pieces that can be scaled independently as shown by the graph below [9].
Figure 6.4: Node.js application as microservices (adrianmejia.com [9])

On the coding level, I used Sublime Text 3 editor to write the code and the command line to run it with the help of Nodemon. **Nodemon** re-runs the code each time there is a change made by the programmer which saves time of re-running a program through the command line manually.
The way each specific technology is used will be further explained in the implementation section.

### 6.1.2. Client Side

As explained before, after receiving a user request to the server, it is necessary to handle it and send a response to it. In our application, the response will be of course in the form of an updated HTML page only (in case of changing routes), or it would include showing some data that belong to the user.

In the first case, the technology used is either jQuery or Backbone.js. I started with the first one to be able to modify the Document Object Model (DOM). It is a hierarchical structure representing the skeleton of the HTML script using the same tags’ names. At some point, I had to switch to Backbone.js for difficulties of using jQuery with Node. While learning about Backbone.js I was introduced to Vue.js as a better way to both create the user interface and keep the application safe. Backbone.js generating HTML code then rendering it can be dangerous if hackers find a way to modify it. Besides, the way Backbone creates HTML code within the JS script can create a mess and make it difficult for the developer to debug.

### 6.2. System Architecture

Before getting into the architecture, it is necessary to remind the reader that each browser has an engine called JS engine, it takes JavaScript code and converts it into Machine code (i.e V8 for Chrome). Below is a drawing of my system architecture:
Figure 6.5: H-NDIR system architecture
6.3. Activity Diagram
6.4. Class Diagram

Figure 6.7: H-NDIR application Class Diagram
6.5. Sequence Diagram

Figure 6.8: H-NDIR application Sequence Diagram
7. Project Implementation

7.1. Implementation Process:
There was no IDE used. I mainly relied on Sublime Text and the command line to run the code. The implementation phase took most of the time as I am relying on a code-and-fix development model.

7.1.1. Overview
First of all, I would like to introduce the website’s routes structure through the below schema:

**Figure 7.1:** H-NDIR website routes.

Now, I would like to clarify the link between the routes, in other words, what are the possible routes paths.
Starting with the first interface the user will interact with, we have the home page. Whenever a user enters our website link, the /home link is loaded. For this route, the only paths that can be accessed from the home page are:
- /Login
After the user is logged in that he/she has the right to request the following additional links:

/MyProfile

/MyLists

/Logout

Within MyLists link, the user can access his/her administered lists, or the ones in which they are members. At this point, there are restrictions in terms of what a member can modify in a list and what can only be updated by the admin. A list member can only add items and update their status, but it is up to the admin to add new members to the list, delete items (either done or not) from the list. A member can delete an item if they are the one who added it.

7.1.2. Installation of dependencies

After installing the latest version of Node from the official website, I installed Git functionalities, Postman for testing and GitKraken to save my code on GitHub. By running the Git Bash on my project files, I did run npm to install: Express, Nodemon, Mongoose and Kerberos. Later in the project I needed to install more dependencies such as: jQuery, Cordova and Bcrypt. I installed MongoDB compass as a user interface to better interact with my DB collections. It is a good start for beginners.

7.1.3. Implementation steps

For a first note this application was developed from scratch, and learning was done in parallel with the development. Some of the paths followed to achieve certain tasks in this project might not be the best, yet its is a good way to start.
The first file created is the test.js file. It requires all express modules that will be used further in the code. Then we require all the other local files that we will use later; these are files that have the code to be executed as a response when a link (route in the code) is requested.

Then we proceed in loading the DB. On the first days of project implementation, I used to open the db, query it and close it each time I need an information from the DB. This caused the application to crash, return undefined elements or take too much time. Moreover, the Asynchronous nature of Node caused other functions to run even before they get the required data from the database. This latter problem is because Node is *Non-blocking*. It used the concept of jacketing through callback functions; while a process is waiting for a resource or a response, the environment moves to other processes and runs them meanwhile. A diagram from the Open Classroom website visualises this idea very well.

![Figure 7.2: The asynchronous nature of Node explained](image)

Thus, the solution was to create another file named `connect.js` that connect to the db and saves an instance of it in a variable and implements functions to get that instance or close the database.

This method ensures that the db is loaded when a user starts a “session”, local variables are loaded with permanent data, while other constantly changing information (i.e list items) are
retrieved from the database each time the user requests them. Once a user logs off, all the variables are flushed.

Speaking of database connection, it is necessary at this point to introduce the Mongo Driver. MongoDB provides a driver that connects the application to the Atlas cluster based on the software version used. It in the form of a string that we pass to our Mongo connection function in the JavaScript code.

Once the connection is established, the port on which the application should run must be specified along with the different engines and express features to use. Last but not least, the test.js file must specify the routes to accept and an error function that for sending an error when non-existing link is requested; it is the 404 (page not found) error handler.

At the end of our code, we export the module(s) that we’d like to share with other files within our application folder.

For the sake of keeping the code organized, the following table identifies the files added, their purpose and their relative path to the parent project file.

Table 7.1: Folders created to organize the project files.

<table>
<thead>
<tr>
<th>Folder/File Name</th>
<th>Folder/File relative path</th>
<th>JS files included</th>
<th>Purpose of the folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>routes</td>
<td>H-NDIR/api/routes</td>
<td>Logger.js, login.js, Logout.js, Member.js, MyLists.js, MyLists.js, PasswordChange.js, Profile.js</td>
<td>Organise the code for the different routes handled in our project</td>
</tr>
</tbody>
</table>
Since the code structure is clarified as well as how the connection is established with the database, I will move on to give a brief description about the most relevant JavaScript files.

**Register.js:** When a user wishes to create an account, they are redirected to the register route. They are asked to enter a unique username and email, then the password and its confirmation. At
this level, the user is redirected to the register.js page again if a mistake has been made while filling the form. the type of error is signaled through the HTML.

**Logger.js:** The Logger file contains three methods. The first one requires the connect.js file to access the db and takes the data from login.js. It queries the database with the information submitted in the login form. If data is correct, a structure with different local variables is initialized. This latter mentioned structure can be accessed using the get() method. Then we have the last method that deletes all the structure data once the user logs out. One of the most important variables of this file structure is the “LoggedIn” variable. Once a user is logged in it is set to true, and through its boolean value we can either allow or deny a request to access a certain route.

**MyLists.js:** This code runs when the lists route is accepted, in other words our user is logged in. It loads user lists and divides them into: member lists or administrated lists. The mylists.js uses Vue.js event handler to act upon the user choice; either they chose to open their member lists or their admin lists.

For error handling, try-catch method was used along with sending specific error messages in the console.

Further, details of the implementation and notes of the different methods followed is available in the H-NDIR repository in my GitHub: github.com/midri.

### 7.1.3.1. Incremental Development steps:

<table>
<thead>
<tr>
<th>Increment Number</th>
<th>Work Achieved</th>
</tr>
</thead>
</table>
| 1                | - Requirement specification  
|                  | - Determination of tools  
|                  | - Installation of dependencies  
|                  | - Linking to the DB following online tutorials |
| 2                | - No change in terms of requirements |
- Learning Node.js, MongoDB, jQuery, HTML and CSS
- Start implementation of back-end based on actual knowledge acquired

| 3 | Completion of the Login and Logout back-end.  
- The concept of sessions isn’t implemented.  
- Moving to a remote DB rather than a local one  
  - Mongo UI: Mongo Compass  
  - Connecting to DB with each query  
- Building HTML for back-end files (login, logout)  
  - Successful  

| 4 | Completion of back-end.  
- Creating a Logger file to store some useful data locally  
  - Also, used to create session token later  
- Setting a unique DB connection per user  
- jQuery doesn’t work with HTML for DOM manipulation  
  - No interaction between back-end and front-end  
  - front-end design is stopped  

| 5 | Learning Backbone.js  
- Starting the first line of codes to update the HTML using Backbone.js  
- Discovering Vue.js  

| 6 | Switching to Vue.js  
- Linking interfaces to equivalent code.  
- Using vuetify.js to design and manage interactivity of UI components.  

7.1.3.2. Challenges

The main challenge was to stick with one learning source and work on my project in parallel. I enrolled in courses offered by codeschool.com to learn more about Node.js and MongoDB. I also started learning jQuery to be able to manipulate my HTML files. During the dependencies installation phase, different errors appeared and below is some important ones and how they were solved.
<table>
<thead>
<tr>
<th>Error Message</th>
<th>Error Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gyp Error: No Python</td>
<td>install Python version 2.7 and set it as default</td>
</tr>
<tr>
<td>Gyp Error: msbuild</td>
<td>run the following command as administrator:</td>
</tr>
<tr>
<td></td>
<td>npm install --global --production windows-build-tools</td>
</tr>
<tr>
<td>ERROR! System error 5 has occurred</td>
<td>run the command as administrator: net start MongoDB</td>
</tr>
</tbody>
</table>

On the GitHub project repository, a file named: *Explanation.txt* contains the different difficulties I encountered along with the options I opted for to solve the errors. Moreover, a file entitled resources.txt has all the necessary resources, organized by tool name, to learn about the technologies used in this project.

Another challenge that I faced has to deal with jQuery. This tool is used to manipulate the DOM from the JavaScript code. Unfortunately, even though I required the jQuery module for node and included it on the HTML script, communication between the two wasn’t possible and the error “document isn’t defined” was informing me that the “$document” used in jQuery wasn’t recognized. Thus, I moved to work with Backbone.js to generate the HTML rendered from the JavaScript file running the user request’s response.

Later, I learnt about Vue.js. It acts as a user-interface building framework that can be linked to the back-end through a soft router. It is more reliable than backbone. This latter is vulnerable to hackers; they can easily redirect the website users to a new malicious link by figuring out a way to change the backbone HTML generated code. Vue.js ensures a good connection between the code dealing with the logic and the one behind the page rendering. It also allows creating interactive web pages or even a single page website.
8. **Project Testing and Validation**

Project testing and validation followed a basic method. Using the activity diagram, I followed the different paths keeping in mind the restrictions already put in the functional and non-functional requirements and tried to create the different scenarios that are supposed to lead to errors. I started by testing the functioning of the different routes by violating the routes rules. I discovered that some paths were accessed even though the user isn’t logged in which is wrong. As a solution I added the private flag “LoggedIn” that signals if a user session is open to allow that link to respond. Once the first case was handled, the second test was directed towards the form response to erroneous input. The login form wasn’t leading to the profile link even though the user data was right. That error was generated within the code because of the asynchronous nature of Node; the result variable was checked if null to determine the following route (which must be the user profile one) while the database was still loading data. The solution was to refactor the connection of the database to avoid waiting for it to connect each time a query is to be made. This latter solution also helped create a user session.
9. **STEEPLE Analysis:**

Referring to the PESTLE analysis online manual, it is necessary to “identify external factors within [our] environment that could have an impact on [our] operations.”[5].

9.1. **Social Analysis**

9.1.1. **Target Market**

To put the readers in the context of my social analysis, I quote the research paper concerned by comparing STEEPLE and PMESII-PT definition: “[a society is] a population whose members are potential customers, share a common need for a product, and may be impacted by the actions of a company”[6]. This said, this work’s target market is composed generally the Moroccan community and specifically of educated individuals that have a decent knowledge about using websites. At this point, users can specifically be categorized as: families, a group of friends, team members, or individuals living in a shared space. The users’ age is of minimum 10 years old.

9.1.2. **Social Acceptance**

At this stage, the application is independent from any other company or entity. Also, after personal interviews with different people with a range between 15 years old and 60 years old, the service offered by our web application seems to satisfy the need of efficiency and clear/concise communication between people to achieve tasks.

9.2. **Technical Analysis**

Our project is based on very recent technologies. It is built in the form of a web application using JavaScript technologies; Node.js, Express.js. The database used is MongoDB and our data is hosted in the MongoDB free cluster.

For testing, we are using the Mongo Shell 3.4 for database commands, Postman to test the responsiveness and output of different links and Git Bash as a shell to run our Node.js code. Our code is shared on GitHub and is constantly updated using GitKraken. These technologies are commonly used nowadays web development which makes it easier to find answers online of
possible problems. Moreover, the documentation available is clear enough to guide programmers through how things work.

9.3. Environmental Analysis:

Our application doesn’t have a direct impact on the environment. However, it can be beneficial in case it is being used as a shopping list for individuals requiring multiple trips by car to the supermarket. Having a shared list reduces the likelihood of individuals forgetting what they might need to complete or shop for. Thus, our project can potentially contribute to less usage of vehicle fuel.

9.4. Ethical Analysis

Our application during development and while operating must respect the fundamental canons of Engineering Ethics defined by the American Society of Engineering Management (ASEM). Besides, user data must kept confidential and respect user’s privacy. For the sake of analyzing the work of our application, we might need user data, therefore, the collection of data will be made after the agreement of the users and will not be transmitted to a third party unless is breaks our application terms and conditions.

9.5. Political Analysis

Our application has no political implication. This means that our application is a civil product that has for aim serving the society and doesn’t have any political motive. Our web application is created to help users overcome difficulties they face in completing their daily tasks and user data won’t be used for any analytical work unless the users give their consent. The user data collected after agreement from its owners won’t be communicated to a third party unless it threatens the safety of the country, individuals or breaks a law that we agree upon as application founders -that we will include for users to consult publicly and agree upon during registration.
9.6. Legal Analysis

At this point, the application doesn’t belong to a company, thus we have no status requiring legal declaration and do not foresee any legal obstacles.

9.7. Economic Analysis

Taking as reference the definition of Walden about economic analysis in the STEEPLE, is takes in consideration external and internal expenses that our project will have to handle as well as the expected economic growth and impact of our project on the external environment [6]. As previously mentioned, our project is initially designed to fulfill basic needs and does not intend to make economic gain from that. Nevertheless, different costs might be included on the short term. First, the database might require additional cost that is variant depending on the selected service and the use per month. Second, a device for testing might be required.
10. Conclusion

The fulfillment of this project was following the requirements of the Capstone Design course. I also relied on previous computer science courses instructions to determine the structure of my project and how I should proceed with the requirement analysis. The choice of the topic was after a discussion with Dr. Sheikh to work on a project that will be beneficial to the community and will help me learn something I am interested in. This project’s idea is a part of a startup application I am willing to develop, so I seized this opportunity and have been able to achieve my goals. This application development process started by determining the target market. Right after, I gathered the user requirements and determined the system requirements. In parallel, I invested time to learn more about the tools I had to use. Once the feasibility study as well as the necessary requirements were gathered, I proceeded with the implementation. The implementation and testing phase was a spiral operation. I implemented parts of project features, tested them then applied the necessary refactoring. The database part was particularly hard as the documentation wasn’t clear in differentiating between MongoDB as a local database or within a cluster. Furthermore, different dependencies had to be installed and some versions had bugs.

All in all, this capstone project gave me the opportunity to learn plenty of new technologies and put them to practice. I learnt about Node.JS runtime environment and had some hard time with its callbacks. I have become more familiar with JavaScript and HTML. Furthermore, I explored MongoDB as a form of document based database, Backbone.JS for HTML creation and Vue.js for user interface building.
11. **Future Work:**

Different updates and features are planned and are listed below:

- Create a notifications service for the user to know the updated lists since the last login by using a local cookie.
- Allow users to decide upon joining a list when they receive a request.
- Create templates for commonly-used list types (such as shopping lists, group study project lists, etc.). A user can then create one of these types by invoking the appropriate template, but can also modify it or create a new list without invoking any template.
- Make this application a mobile cross platform application: This will require Cordova to open the web pages in a hidden browser. In this case, each of the routes in our website will be assigned an interface incorporated within the Android/ iOS application.
12. References


13. Appendix A

- Database Schema:

Users Schema:

```javascript
// Require Mongoose
var mongoose = require("mongoose");

// Define a Schema
var Schema = mongoose.Schema;

var usersSchema = new Schema({
  id: mongoose.Schema.Types.ObjectId,
  username: {
    type: String,
    required: true
  },
  password: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  lists: {
    NOT_ADMIN: {
      list: [
        Item_u_Details: {
          User: {
            type: String,
            required: true
          },
          Item_status: {
            type: Boolean,
            required: true
          },
          Item_Update: {
            type: date,
            required: true
          },
          Item_name: {
            type: String,
            required: true
          }
        }
      ]
    }
  }
});
```

Figure A1: Snapshot 1 of the User Schema code
Figure A2: Snapshot 2 of the User Schema code
Lists Schema

```javascript
var mongoose = require('mongoose');

// Define a Schema
var Schema = mongoose.Schema;

var ListSchema = new Schema(
  {
    id: mongoose.Types.ObjectId,
    List: {
      List_Name: {
        type: String,
        required: true
      },
      Users: [
        {
          type: String
        }
      ],
      List Elements: {
        Item_N_Details: {
          User: {
            type: String,
            required: true
          },
          Item_Status: {
            Type: Boolean,
            required: true
          },
          Time_Update: {
            Type: Date,
            required: true
          },
          Item_Name: {
            type: String,
            required: true
          }
        }
      },
      Admin: {
        type: String,
        required: true
      }
    }
  }));

// Compile model from schema
var Lists = mongoose.model('lists', ListSchema);
```

Figure B: Snapshot of the List Schema
Admin Schema

```javascript
const Schema = mongoose.Schema;

var AdminSchema = new Schema({
  USER_ID: {
    Lists: [{
      List: {
        List_ID: {
          type: 'String',
          required: true
        },
        List_Name: {
          type: 'String',
          required: true
        },
        New_Field: {
          item_N_details: {
            User: {
              type: 'String',
              required: true
            },
            item_status: {
              type: 'Boolean',
              required: true
            },
            Time_Update: {
              type: 'Date',
              required: true
            },
            Item_Name: {
              type: 'String',
              required: true
            }
          },
          required: false
        }
      }
    }
  }
});

var Admins = mongoose.model('Admins', AdminSchema);
```

Figure C: Snapshot of the Admin Schema