**National College of Ireland**

**Higher Diploma in Science in Web Technologies**

**2018/2019**

Student name: Andrea Baccolini

Student number: 18147518

Student email: x18147518@student.ncirl.ie

Project name: rockonnect

**Technical Report**



**Table of Contents**

[Executive Summary 5](#_Toc15895743)

[1 Introduction 6](#_Toc15895744)

[1.1 Background 6](#_Toc15895745)

[1.2 Aims 6](#_Toc15895746)

[1.3 Technologies 6](#_Toc15895747)

[1.4 Structure 7](#_Toc15895748)

[2 System 8](#_Toc15895749)

[2.1 Requirements 8](#_Toc15895750)

[2.1.1 Functional requirements 8](#_Toc15895751)

[2.1.2 Data requirements 11](#_Toc15895752)

[2.1.3 User and usability requirements 11](#_Toc15895753)

[2.2 Design and Architecture 11](#_Toc15895754)

[2.3 Implementation 13](#_Toc15895755)

[2.3.1 Project Setup 13](#_Toc15895756)

[2.3.2 Sprint 1 implementation phase 15](#_Toc15895757)

[2.3.3 Sprint 2 implementation phase 18](#_Toc15895758)

[2.3.4 Sprint 3 implementation phase 21](#_Toc15895759)

[2.4 Testing 21](#_Toc15895760)

[2.5 Graphical User Interface (GUI) Layout 24](#_Toc15895761)

[2.6 Customer testing 26](#_Toc15895762)

[2.7 Practical evaluation 27](#_Toc15895763)

[3 Conclusions 28](#_Toc15895764)

[4 Further development or research 29](#_Toc15895765)

[5 Appendix A: Project Proposal 30](#_Toc15895766)

[5.1 Objectives 30](#_Toc15895767)

[5.2 Background 31](#_Toc15895768)

[5.2.1 The idea behind the application 31](#_Toc15895769)

[5.2.2 Target audience 32](#_Toc15895770)

[5.3 Technical Approach 32](#_Toc15895771)

[5.4 Special resources required 33](#_Toc15895772)

[5.5 Technical Details 34](#_Toc15895773)

[5.6 Evaluation 34](#_Toc15895774)

[6 Appendix B: Project Plan 35](#_Toc15895775)

[7 Appendix C: Requirement Specification document 37](#_Toc15895776)

[7.1 Introduction 39](#_Toc15895777)

[7.1.1 Purpose 39](#_Toc15895778)

[7.1.2 Project Scope 40](#_Toc15895779)

[7.2 User Requirements Definition 41](#_Toc15895780)

[7.2.1 Technologies used to meet the requirements 41](#_Toc15895781)

[7.3 Requirement specification 42](#_Toc15895782)

[7.3.1 Functional requirements 42](#_Toc15895783)

[7.3.2 Non-Functional Requirements 50](#_Toc15895784)

[7.4 Interface requirements 52](#_Toc15895785)

[7.4.1 GUI 52](#_Toc15895786)

[7.4.2 Application Programming Interfaces [API] 55](#_Toc15895787)

[7.5 System Architecture 56](#_Toc15895788)

[7.6 System Evolution 56](#_Toc15895789)

[8 Project design and specification document 57](#_Toc15895790)

[8.1 Introduction 58](#_Toc15895791)

[8.1.1 Purpose of the Technical Report 58](#_Toc15895792)

[8.2 General Overview and Design Guidelines/Approach 58](#_Toc15895793)

[8.2.1 Assumptions / Constraints / Standards 58](#_Toc15895794)

[8.3 Architecture Design 58](#_Toc15895795)

[8.3.1 Logical View 60](#_Toc15895796)

[8.3.2 Software Architecture 60](#_Toc15895797)

[8.3.3 Security Architecture 61](#_Toc15895798)

[8.3.4 Communication Architecture 62](#_Toc15895799)

[8.3.5 Performance 62](#_Toc15895800)

[8.4 System Design 63](#_Toc15895801)

[8.4.1 Use-Cases 63](#_Toc15895802)

[8.4.2 Database Design 63](#_Toc15895803)

[8.4.3 Data Conversions 64](#_Toc15895804)

[8.4.4 Application Program Interfaces 64](#_Toc15895805)

[8.4.5 User Interface Design 64](#_Toc15895806)

[8.4.6 Section 508 Compliance 64](#_Toc15895807)

[9 List of figures 65](#_Toc15895808)

[10 Other Material Used 67](#_Toc15895809)

[11 Definitions, Acronyms, and Abbreviations 68](#_Toc15895810)

[12 Bibliography 69](#_Toc15895811)

# Executive Summary

There is an incredible amount of web applications in the web, each of them doing something different. Through a web application we can many things. rockonnect is a web application like those because it tries to bring something new in the music experience made social. This report discusses the idea and from which issue does that come from, and how to address it and create something relevant that solves the actual issue. This document describes in detail the development process and how the development team tried to adhere to the actual product specifications and project plan which are the core part of the entire project to determine its success. Also, this document performs a clear analysis on how some of the requirements have evolved from the beginning to the end. There is a clear example of this in the section describing the requirements. During this journey a lot of milestone have been met and issues have been encountered and overcome, with the aid of the technology used and the progress made by the development team itself. There is not much difference in the technology that were meant to be used vs those that were actually been used for the development and this is highlighted in the report, also. Additionally, the report allows the evaluation of the overall web application experience through a detailed objective test cases document that have been submitted and completed by the team and by external people. This third party performed additional subjective measurement on the overall usage and experience with the web application. This is an important aspect of this report, because testing was performed by the team during the development, but not officially tracked in documents till now; and this is the opportunity to summarize the test performed.

It was a great journey , thanks for the opportunity and for reading.

# Introduction

This template for technical report is provided for your convenience. It should be seen as a guide rather than an obligatory form. Your individual report might require changes in terms of format or content (i.e., headings) or both.

Print on one side of the paper only (this will be the right-hand side when the pages are bound).

## Background

The idea of creating a web application focused on classic rock experience and the possibility to share this experience with other users has it roots in the fact that there are not many applications in the internet world focused on those two subject therefore, after a short market analysis as shown in the project proposal document (Baccolini, 2019), this resulted to be a good idea to develop.

## Aims

The goal of the web application is to provide an online platform for users to share discussions, opinions, images on their experience with classic rock and to connect with other users.

## Technologies

The server-side programming language used is node.js. on the client side the system is written in node.js, JavaScript, and pug [Jade: this latter for HTML templating].

There are few reasons for choosing JavaScript among other server-side technologies [python or ruby-on-rails, for example]. According to some statistics (Rosalie, 2019), (Geek for Geeks, 2019) JavaScript is one of the most popular programming languages in the last few years and in 2019 is growing even more due to the growth of web applications.

Among the JavaScript world of frameworks and library node.js have been chosen as preferred framework, due to its versatile structure for client-server applications and abundance of libraries. So, for the purpose of this web application, the following main node.js libraries and generic software have been used:

* Node.js and npm: server and client side javascript language
* Express: framework for web applications
* Mongo db for data storage and manipulation [CRUD]
* Mongoose [ORM Object Related Mapper] for db communication [in case of mongo db]
* Pug: template engine [for the views]
* Bower: package manager
* jQuery: for DOM [Document Object Manipulation]
* bootstrap: for website framework
* bcrypts: for password encryption
* passport: for user login/session/logoff features
* multer: for image/file storage on the server side

## Structure

The document is structured in the following way:

Section 1 gives an introduction of the document, focusing on what has been developed, the reasons behind the nature of the project and the technologies used for the development of the applications, with an understanding of the technologies used.

Section 2 is focused on the development. This section highlights what has been developed and how, focuses also on the main issues encountered during the development and how those have been overcome to progress with the development. Screenshots of the code are also provided to explain what they achieve and how they have been developed in the web application. Finally analyses also how the development and testing was carried out and provides also extensive screenshots of the web application. Finally, this section evaluates the work done, the results achieved and the tightness to the project plan timelines and milestones.

Section 3 provides a conclusion of the work, with personal consideration from the development team.

Section 4 shows a roadmap of the features for the next releases of the web application [yes, this journey moves to its second leg!]

Subsequent sections are describing the following documents that have been written during the months of the development:

* Project proposal: this document highlights the idea and what is going to be developed
* Project plan: a gannt, wbs and project plan with the milestones to keep track and properly manage the development.
* Requirement specification; this is one of the most important documents for the project. Describes in detail the requirements including extensive use cases with the correct flow that need to be followed during the implementation. It also describes the non-functional requirements and expected pages/views of the webapp.
* Design and architecture: this is the document that addresses how to implement the different requirements and use cases described in the previous document, through the best technology to use and the architecture chosen to develop such requirements.

# System

## Requirements

This section will be similar to your original requirements specification. Requirements have probably evolved somewhat since. Where this is the case explain what changed and why.

The following are the requirements that have been developed for the first release of the web application. The main target market has not changed during the implementation phase, so the “persona” identified in the “requirement specification document, section 2” (Baccolini, 2019) has not changed, therefore minimal changes to the requirements occurred during the development. The main changes are explained in the section 2.1.1

### Functional requirements

The following use cases have been developed for this initial release of the web application:

1. Register to the community
   1. Create a profile
2. Sign in the application
   1. Browse users’ profiles and connect to users
   2. Create new threads
      1. Upload images
3. Read threads
4. Read the list of users
5. Logout the web app

Figure 1 shows the list of use cases.

The web application is available for both anonymous users and logged in users. The anonymous users can only read the threads and not browsing the detailed profiles of the users: this is strictly due to privacy concerns. Only users registered in the community [i.e. the web app] can see the details of the other users and, eventually, connect to them if they like.

So registered users can enjoy the full features of the web app only when they login. At that time, they can add new or updated existing threads under the discussion forum, they can browse the details of the user’s profiles and then connect to them.

A close up of a map

Description automatically generated

Figure 1: use cases

#### Requirements Registration and Sign in

So, requirements 1 and 2 [users registration and users login] of rockonnect, have not changed during the development, so for the explanation of those use cases, please refer to Sections 7.3.1.1, and 7.3.1.2 of this document.

#### Requirement read the threads/discussion

One of the biggest changes in the use cases is with regards to the “View Threads” use case. This has been developed for anonymous users and not only for logged in users. In the very first draft of the requirements, the feature of viewing the threads, was subjected to users logged in. during the development phase the development team [which includes the product owner of the web applications] decided to make this feature available for anonymous users as well to make the web app “appealing” to non-registered users. In this way the anonymous users are invited to register and eventually log in, if they find the content of the web application interesting.

So, the requirement has been implemented as the following use case is showing:

A close up of text on a white background

Description automatically generated

Figure 2: thread view requirement use case

Anonymous users as well as logged in users, can view each thread by clicking on its title under the discussion forum page of the web application. The use case does not change as users need to be logged in to update/create new threads, therefore the flow shown in section 7.3.1.4 at page 48 of this document is still the flow that has been implemented. The edit/update threads functionality has been put into a separate page that needs the user been logged in to be viewed. Please refer to section 7.3.1.4 of this document for a complete view of the use case.

#### Requirement create new thread/update existing:

There have been no major changes in this use cases during the development. For a review of the implemented use case, please refer to section 7.3.1.4.

#### Requirement browse users’ profile and connect to users

There have been no major changes in this use case during the development. The users cannot see the detailed profile of other users unless they are logged in which makes them part of the web application/community.

If users are not logged in, they can still surf the users page and see the list of [username only] the users that are part of the community but not their detailed profile. This functionality has been required since the beginning and haven’t been changed during the development.

#### Requirement logout

There have been no changes in this use case, please refer to section 7.3.1.5 of this document for a description of the entire use case.

### Data requirements

The webapp is designed to store the minimal user data. the data is stored into a remote cloud-based database which is accessible only by the administrator’s user.

As for the other meaning of data [content], the web app is accessible through a standard broadband internet connection, which needs to be present during the use of the webapp. It is a mandatory non-functional requirement.

There are distributed modules that need to be reachable therefore also because of this reason the underlined internet connection needs to be available.

#### Content and User data

rockonnect is designed for people over 18 years and does not provide any mean for user to share material that could possibly infringe copyright or could also possibly be obscene. This is clearly stated in the proposal document (Baccolini, 2019). In the first production deployment of the webapp, rockonnect will be equipped with all the necessary disclaimers to address the points just mentioned before. In addition to that, it is responsibility of users not to use copyrighted material in the threads or in the user profile, and the system does not store sensitive information like age, locations etc.

The passwords used by the users for login are encrypted and saved into a remote cloud-based database, same for email username and the other users’ info necessary for the registration.

### User and usability requirements

Emphasis during the development has been put on the easiness to use and accessibility of the webapp features. So, rockonnect is easy and light in order to give the best user experience possible. The minimum weight is on the user interface, so the app pages are lean and very clean, responsive and easy to use. By purpose there aren’t many images in the pages as the layout is designed to be plain and lean, and users can upload their own images in the thread.

Additionally, the system is designed to be accessible through a standard browser that could be a desktop or a mobile browser, and that requires also a standard internet connection.

For more detailed requirements, please refer to section 7.3 of this document.

## Design and Architecture

This section describes the architecture of the system and the design of the software modules that compose the application.

rockonnect is a client server web application. Therefore, it is a semi-distributed system that allows the interconnection between different software modules that can be distributed in different locations. Please refer to *Figure 3*.

A picture containing text, map

Description automatically generated

Figure 3 - rockonnect architecture diagram

As shown in figure 3, there are different building blocks in the architecture. rockonnect, as a web application, is built at the top of a reliable internet connection throughout the client-server process so that HTTP/s protocol can run. The figure shows also the other main building blocks of the web app that are:

* The core application server hosts the webapp server and client frameworks. This is the core of the web application and hosts the app.js file, plus all the necessary modules and directories used in the app.js file.
* The User interface CSS and generic associated styles is rendered by bootstrap through its API. In order to make the user interface as clean and lean as possible, a standard web template has been adopted and modified, to accommodate the user interface of the web application. By purpose, since the app features are available under the condition of a reliable underline standard broadband connection, the user interface has been designed very light in order to make the web app available under the critical condition of an undatable internet connection.
* mongo DB: this is the database that hosts users’ data and content. It is the cloud version, and it is hosted remotely and separated by the web server of the application. The decision of using a non SQL database vs a more traditional SQL approach, has been dictated by the fact that the development team wanted to demonstrate, during the development, that the learning experience did allow the team to take a certain degree of “hazardous” decisions, like using a somewhat unknown piece of software [mongo DB for example] to learn something new during the entire progress. Another advantage of using mongo db, was the fact that the syntax is somehow like JavaScript and allows manipulation of JSON file easier than traditional SQL database. For the team this last was an advantage.

Note: being a distributed system, the availability of an underline internet connection is very important to allow the different modules to exchange what’s required to give the end user a good user experience.

In the first release of the application, there is no redundancy due to the distribution of the main software modules, as there was no time to develop such an architecture in the short period of time available. This is a considerable important feature that can be added in the roadmap. Refer to section 4 for the proposed future enhancement of rockonnect.

## Implementation

The implementation has followed the tasks timelines as per project plan: please refer to section 6 for the actual project plan, wbs and gantt.

The following sections describe the 4 main implementation phases as per project plan.

### Project Setup

The first one is the project setup. Approximately 4 weeks and no code has been developed in this section. The main purpose of this 4 weeks was to deliver the required upfront documentation such as the project proposal and the project requirement specifications. The last one was the most important document as it describes the web application in its functions and expected use cases so the team has put a lot of efforts in providing a detailed specification to be able to develop with most of the understanding cleared upfront and minimizing the unclear areas. Another important milestone achieved in the project setup phase was the prototyping and the setting up of the environment as shown below in the *Figure 4* and *Figure 5*.

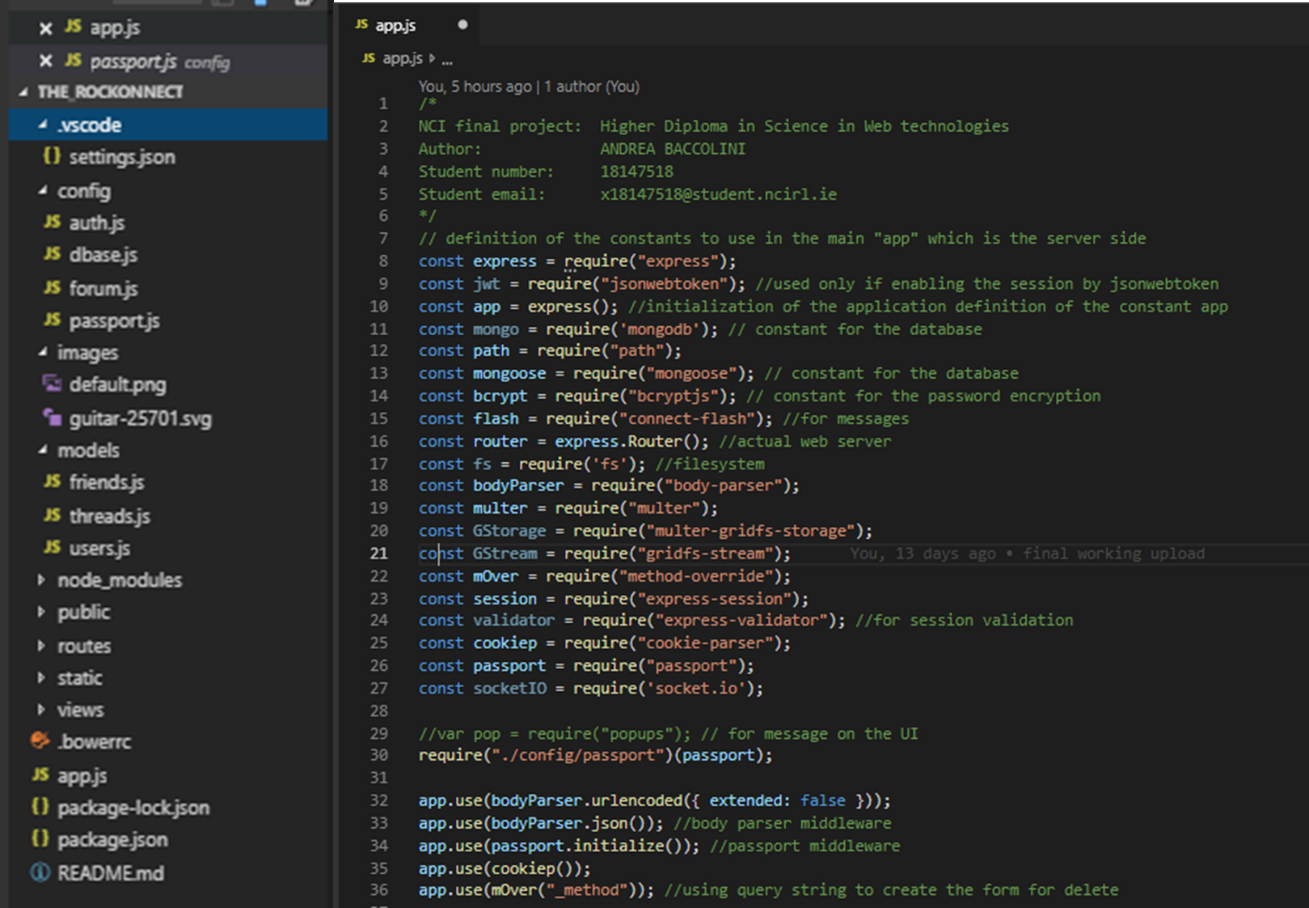


Figure 4 - environment setup 1

As shown in *Figure 4*, one of the design aspect that the team adopted was to keep separation of few libraries and create few files instead of keeping everything in the main server file called app.js.

Therefore, the files named “passport.js” is used for implementing the user login logic [or known as strategy in the passport library], while the “auth.js” is implementing the logic to associate the user to the session if logged in. “dbase.js” file is containing the database information.



Figure 5 - environment setup 2

### Sprint 1 implementation phase

Approximately from end of June to end of July 2019 has been dedicated to development and testing, and to necessary writing of the remaining documentation needed. Sprint 1, that took place from June 28 to July 11, was focused on the development and test of the server side routes, CRUD to the database and modelling of the user data.

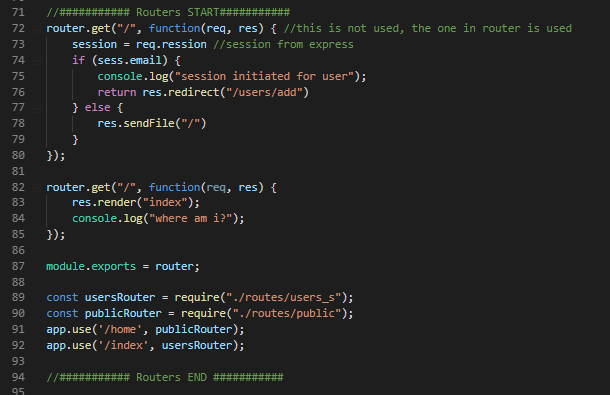


Figure 6 - routes development example

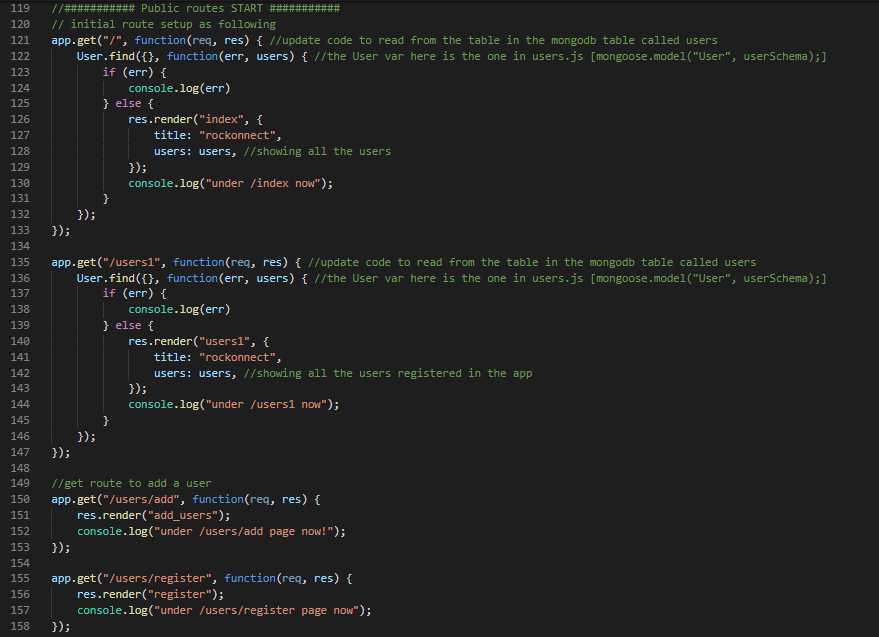


Figure 7 – public and private routes

One important aspect of the routes is that since the beginning the public and private routes have been designed. Public are, obviously, for the anonymous and logged in user to be visible and reachable, while the private can be reached on if users are logged in.

In sprint 1, one of the achieved milestones was to create a registration logic that could do the following:

1. Allow the user to register only if he/she chooses a unique username [unique means that is not used by another user previously]. This is shown in lines 171 to 186 of *Figure 8*, where the system looks into the mongo DB if the username is present already using the following method:

User.findOne({ username: username })

1. Crypt the chosen password using the library Bcrypt (Community Node.Js, n.d.), by which the password field is salted and then set to the hash value to substitute the clear one before saving the new user into the database as from lines 188 to 204 in *Figure 8*.
2. Save the user object into the remote database

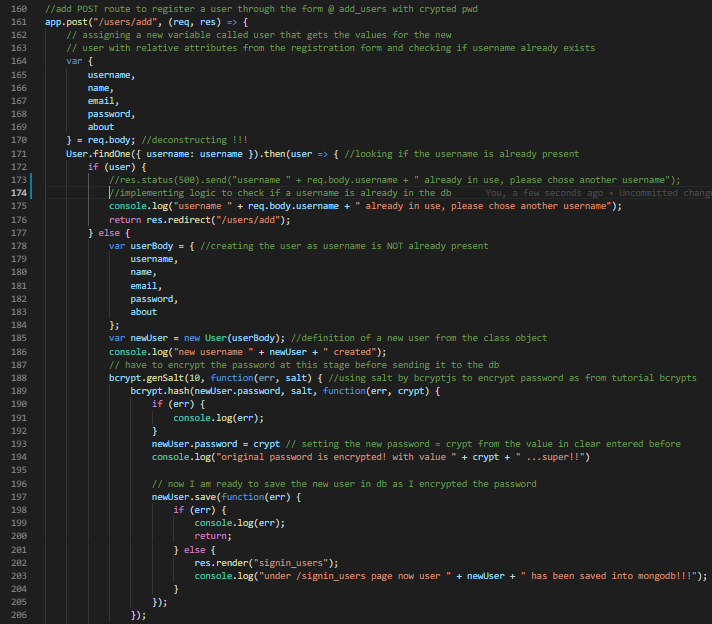


Figure 8 - users registration

Another important achievement in Sprint 1 was the implementation of the logic for the user login, using passport library (Passport, n.d.), as shown in *Figure 9*. As per passport documentation the user login as been implemented using the local strategy, by which the user can login using standard username and password. To do so, there was the need of generating a decrypt function for the password stored into the db, in other to match with what the user has input at the time of login. Thanks to bcrypt this functionality was implemented by using the method compare as shown here:

var pwdcheck = bcrypt.compare(password, user.password, function(err, match)

Please refer to lines 20 to 29 in *Figure 9*.

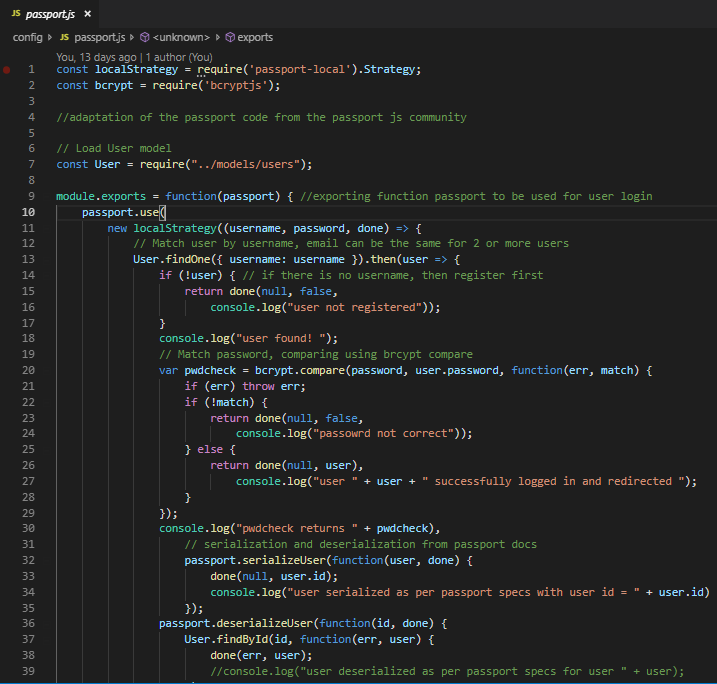


Figure 9 - user login [passport]

A function called “isLoggedIn” has been developed to protect the private pages. More in the following section of this document.

### Sprint 2 implementation phase

Sprint 2 took place from middle July to beginning of August 2019. It was mostly focused on the client-side framework development, finishing the implementation of the remaining server-side part and testing with the server side, plus the additional documentation generation.

The following figures show some of the achieved results during sprint 2 implementation. The library called “multer” has been used for storing the images into the database. The users can upload images in the threads/forum page so that they can enrich their discussions. To do so the multer library helped in determining a unique name for the file that user A was uploading to avoid any potential conflict with another user in the remote possibility of both uploading an image with the same exact name. this is shown in *Figure 10*.

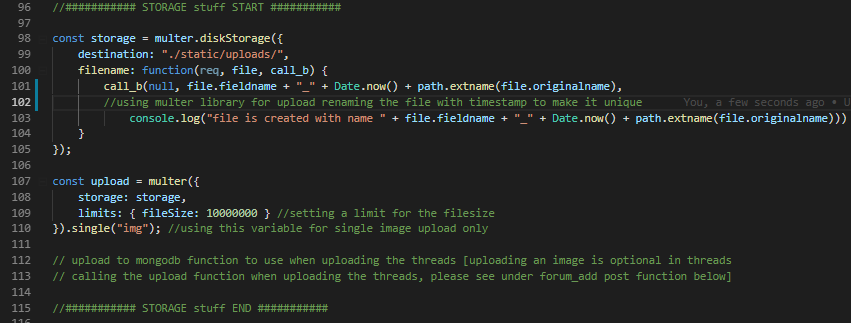


Figure 10 - multer upload

Another part implemented in this second phase was the Read part of the CRUD to/from mongo DB. The code in *Figure 11* shows the concept.

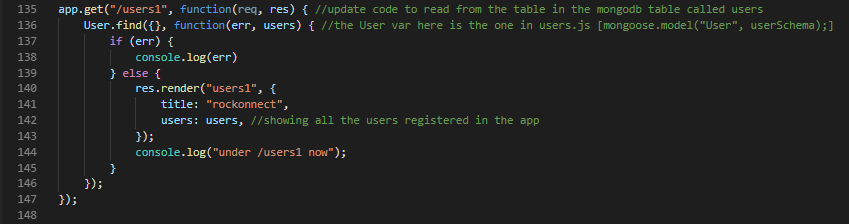


Figure 11 – DB read

This implementation phase also resulted into the development of the logic to protect the pages.

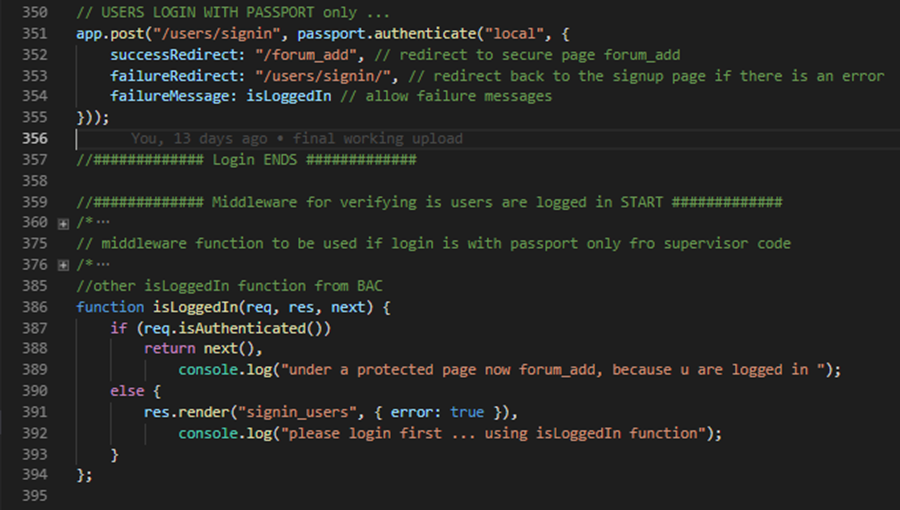


Figure 12 - isLoggedIn

By implementing the function shown in *Figure 12*, users can be redirected to the login page in case they want to access that resource and they are not associated to a session [a.k.a. they are not yet logged in]. this function has been used to protect the user’s details and the add forum views as per original requirements.

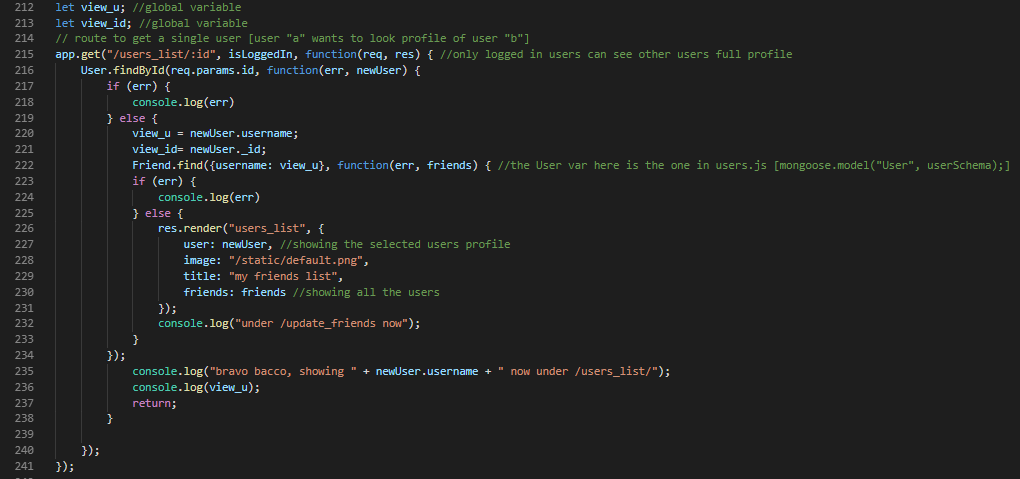


Figure 13 - Users view

Finally, sprint 2, led also to the implementation of some logic that allows rockonnect to be a social web app which is the capability of adding friends to a personal list. This has been achieved without the help of using additional libraries, just pure code specifically written as shown in *Figure 13*, where the system is designed to look for a specific username in the collection called “friends”, and then render the “view\_u” [users associated to that username]. The development was focused on implementing a separate table for the logged in users that listed all their associated connection [by username], as shown in *Figure 14*.

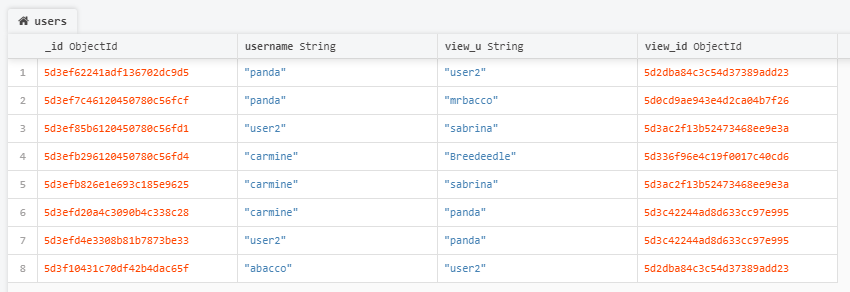


Figure 14 - Friends db collection

### Sprint 3 implementation phase

As previously stated, Sprint 1 and 2 were focused on the development more than any other phase of the implementation. The last sprint, Sprint 3, was just less than 2 weeks long and focused on the clean-up, wrap up of the code as well as the creation of the video for the final presentation of the web application.

## Testing

The web app has been tested from a functional and user perspective. Development team has performed testing throughout the implementation phase, and potential users of the web app have been invited to do the same by just using the web application.

That means that the following test cases have been performed:

Following the pre-conditions for each of the test.



Based on the type of test that was performed additional preconditioned were added under the column “Assumptions and test steps”. Also, the tests were conducted on three senior users 2 of them using a windows 10 laptop and using edge and chrome as web browser. The third person was using a MacOS and chrome web browser.



Figure 15 - test results 1



Figure 16- test results 2

## Graphical User Interface (GUI) Layout

Following some screenshots of the application.

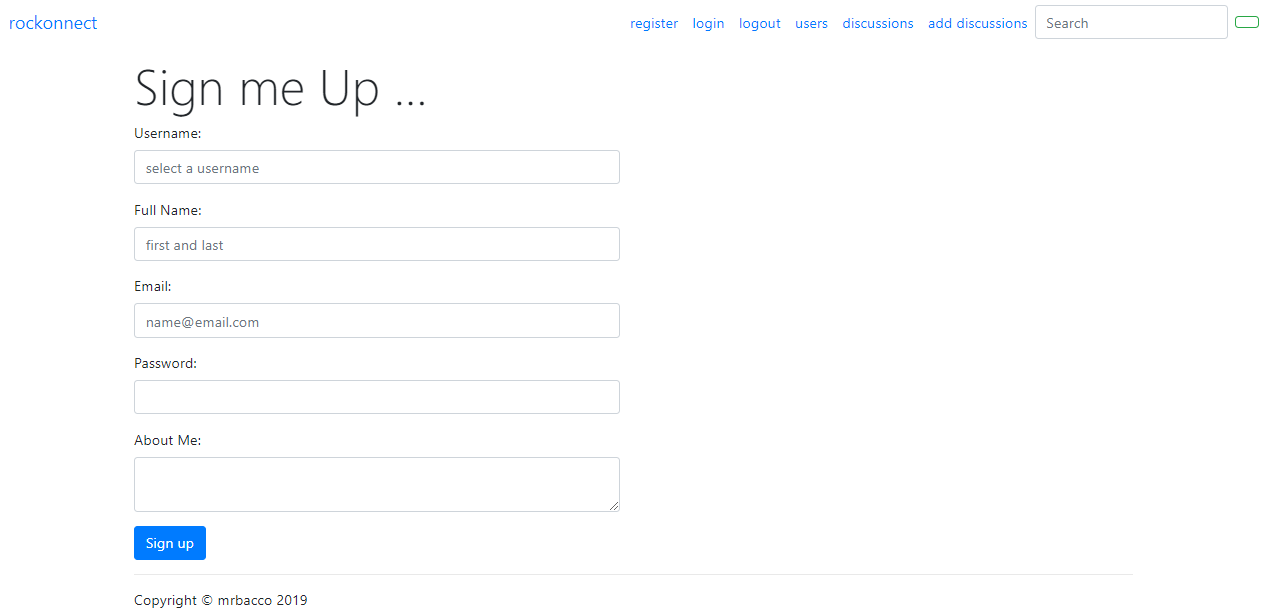


Figure 17 - registration page

This page allows the registration of the users: all the fields are mandatory.

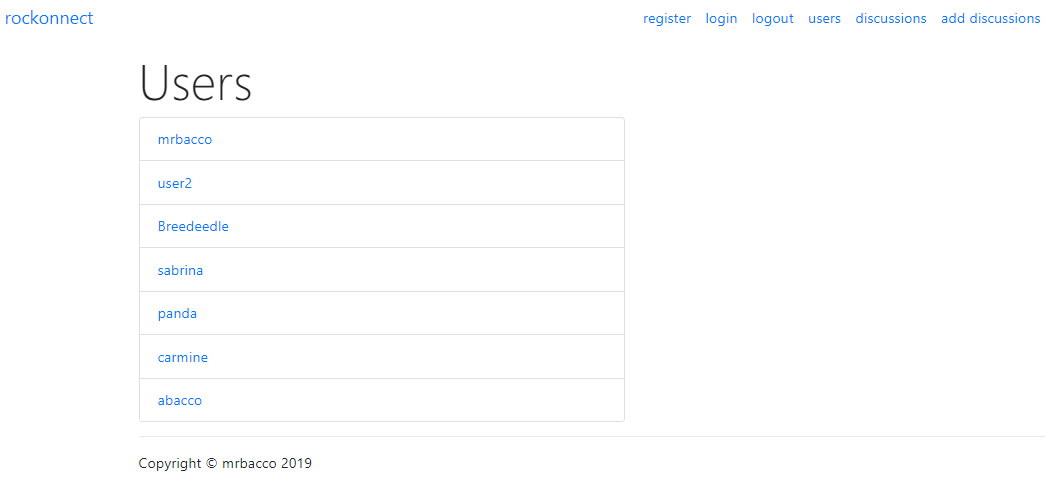


Figure 18 - users list page

User list carries the list of the registered user, if a user is logged in then he/she can click on the user profile of interest and get to the following page:

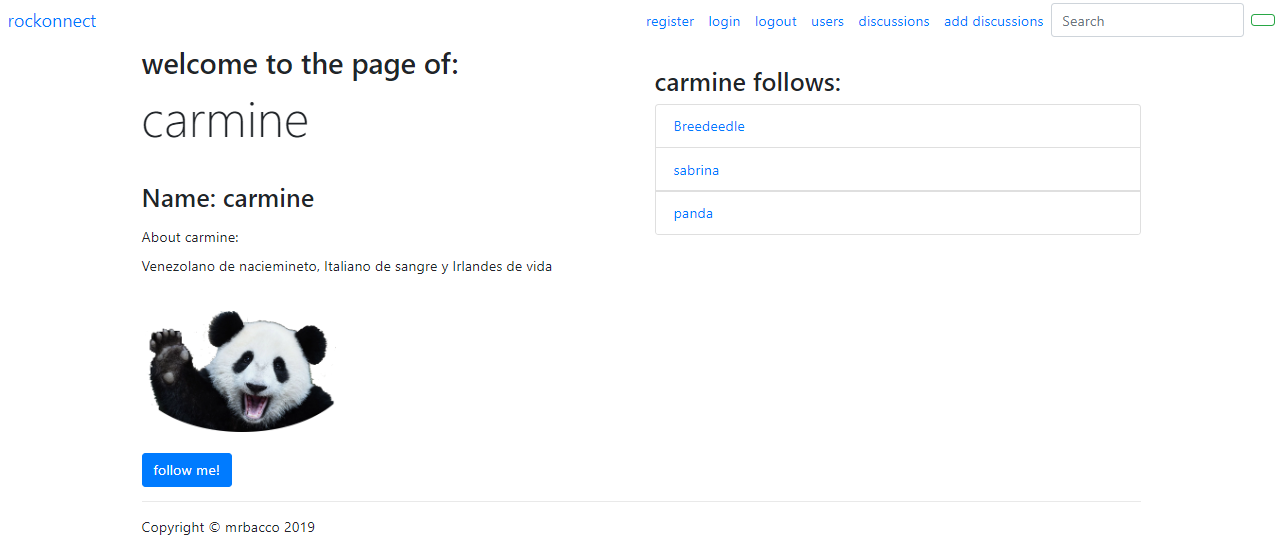


Figure 19 - user profile page

*Figure 19* shows the profile of the users. As mentioned already this page is accessible only if the user is logged in. under this page, “User A”, who had previously enjoyed a lot of posts by “User B”, can connect to “User B” by clicking the follow me button located under the “User B” picture or avatar.

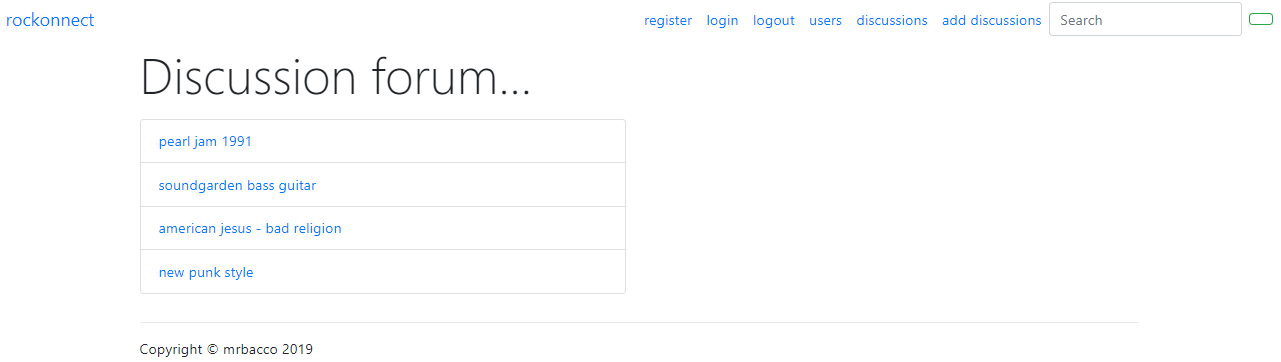


Figure 20 - Discussion forum

*Figure 20* shows the list of the discussion items that all the logged in users have already created in the application, by clicking on one of them, anonymous users and logged in users can see the details of the item being discussed, as shown in *Figure 21*.

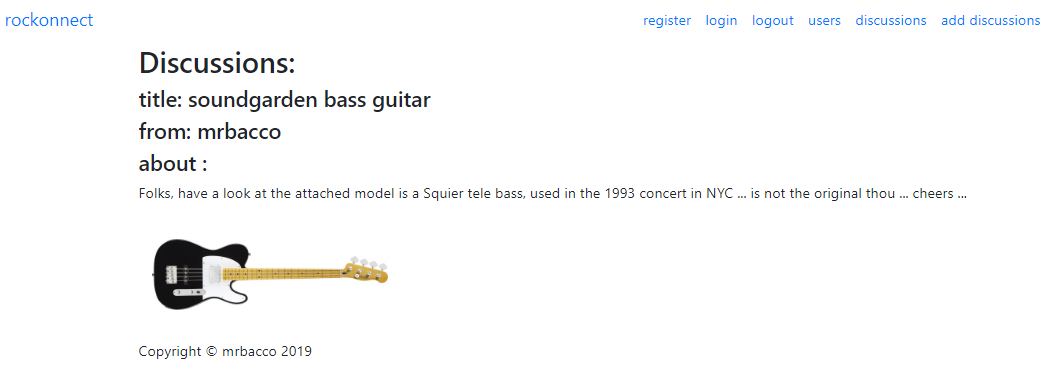


Figure 21 - discussion item

The last but not least important screenshot is the post thread page screenshot, which is shown in *Figure 22*.

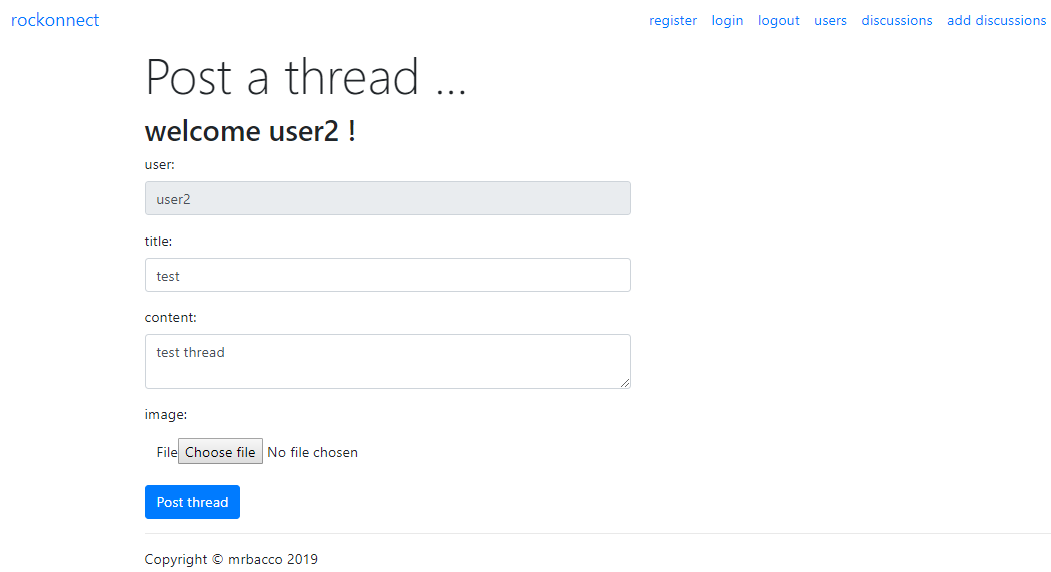


Figure 22 - post a thread

In this page, logged in users can create their own discussion items and upload a picture to be more effective. As shown in *Figure 22*, once user accesses the page is greeted with a welcome message and presented with the form to add the content. The form also is precompiled as far as the username is concerned, which the system knows from the passport session associated to the logged in user.

## Customer testing

The same form used for testing as per section 2.4, was used by real users to test the app that has been made available under “heroku”.

This test was extremely useful as the tester, in this case, was not aware of any details of the implementation and was informed prior to the test, of the goal of the web application. So, the user was giving very important feedback about the overall user experience of the application.

Lots of comments came back among with the following [they are from real users]:

* “you should use more CSS to make it prettier”
* “there are some page redirections that are not logical, in my opinion: why should I be redirected to the homepage if I login?”
* “overall is good, I would add more messaging and alerts from the system, but I like the idea”
* “I would like to know more features in this”
* “ … can I communicate directly 1-to-1 to a user that I particularly like?”

Most of the feedback have been logged in the roadmap backlog and will be part, together with other features, of a plan for the next releases of the web application. yes, as mentioned, this journey continues!

## Practical evaluation

The system has been evaluated in two main ways.

One is the objective evaluation: does the system comply to the use cases? Does it perform the tasks and achieve the results that are expected following the requirement specification document?

The second way of evaluating rockonnect is in comparison to the real world and usage: does it solve a problem in real life or is it useful to anybody? Are people enjoying the web app? Is it easy and straight forward to use?

All the answers above have been summarized in the testing document that has been used by technical and non-technical people to evaluate the web app.

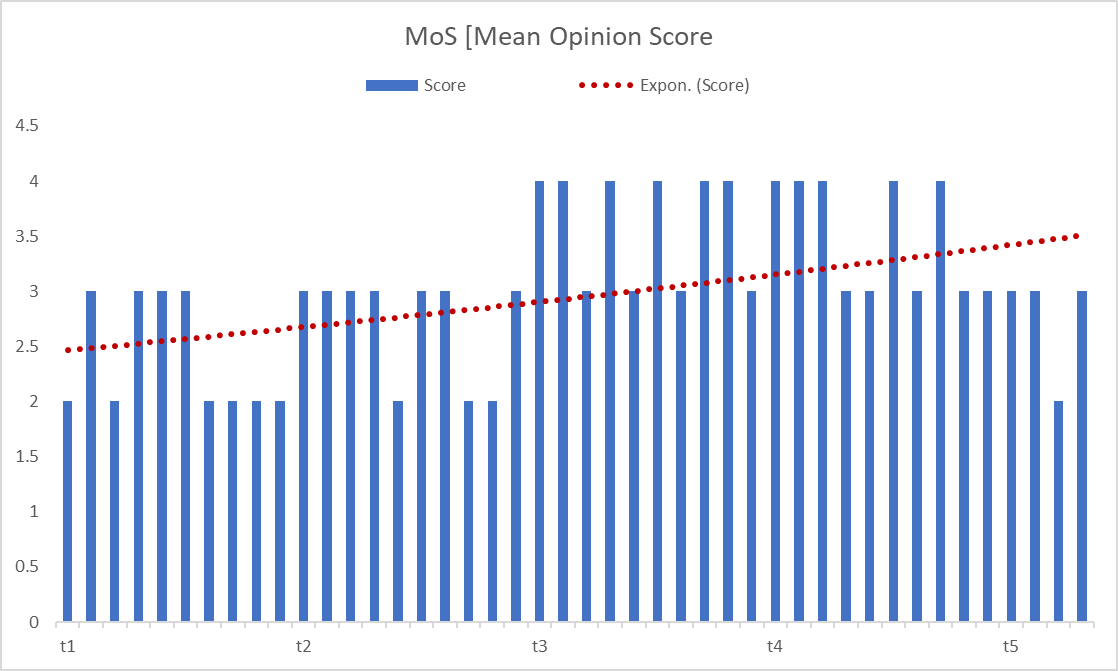


Figure 23 - Mean Opinion Score

The MoS [Mean Opinion Score] evaluation system used in the test document, has helped a lot to understand the trend of the development on the overall web app experience. The chart at *Figure 23 - Mean Opinion Score*, is plotting the same questions asked in different points of time during the development to the same group of people that tested the application. The red dotted line in the chart shows the exponential trend of the score that kept increasing overtime.

# Conclusions

rockonnect is for some aspect another web application on the market, and this is surely a limitation. But, on the other side, the application has a specific goal and scope and it is different from what is available in the market already. There is a lot of room for improvement as highlighted in the test results and users’ comments, and that needs to be addressed. In addition to that there are features that you can see in the following section, that will be added in the product backlog as roadmap features and will be implemented.

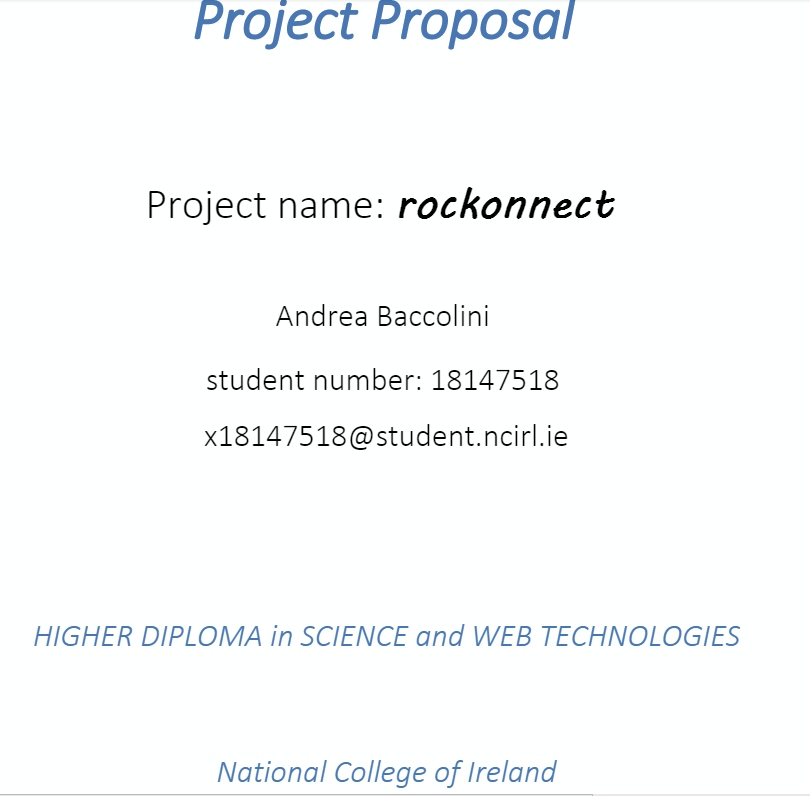
Overall the journey has led to a satisfied development team and a working application

# Further development or research

With more resources, the project could have supported the following features in order of priority:

1. controls on the same user added as friend more than one time: now there is no control if User A adds User B more than one time to his/her list of friends
2. An email sent for each user at the time of registration in order to validate that a real email is used
3. possibility to reply in the same thread and create a nested reply for a single thread
4. login with social media profiles [oAuth2 for Facebook, Twitter and/or Instagram] in addition to username and password
5. possibility for User A to be notified via email when User B and others are following him/her
6. ability to like and comment each post and notification sent to users
7. users can upload multiple images/files in the forum page when editing/creating a new post

# Appendix A: Project Proposal



## Objectives

The objective of the Project is to create an online platform which allows users to have a dedicated community focused on classical rock music. By “Classic Rock” is intended the music genre made of blues rock and hard rock music growing worldwide between the early 80s and the late 90s.

The online platform is a web application that can be accessed from any device having a browser and an internet connection. The web application is a way in between a social network app and a web forum. The name of the web app is “rockonnect”.

In more details rockonnect allows registered users to enjoy the following features:

[A] features pertaining to a traditional social media app: connect to/follow users that have an interesting profile and create / post user generated content like multimedia videos/pictures of classic rock events/instruments etc...

[B] features proper to a web forum: registered users can hold conversations [“threads”] in the form of posted messages. With this tool, users can exchange opinions and ideas about everything related to classic rock. Some examples are: threads on old published albums; concerts; rock bands performances, musicians; and musical instruments. The threads can be user even to organize and meet for jamming purposes. Anything that is related to classic rock can be discussed in the threads. There will be different user rights: seasoned users will also benefit of the admin rights which allow them to supervise the content and to ban users that might not comply with regulations. Regular users [most of the users of the app] will be able to connect with others but read and write their own content only.

rockonnect is not supposed to be used as a music file sharing system or for anything that can be subjected to copyright infringement and obscene/illegal material. It is responsibility of the users not to distribute such material through the web application.

Most of the current web applications are focused on the music streaming aspects of the classic rock music. There are plenty of these on the internet, and rockonnect is not going to provide any streaming radio functionality. Might collect a list of already available radio station streaming classic rocks songs, or users are free to exchange links to channels in the thread.

rockonnect content is going to be expanded by the community behind it, and the same community will drive the roadmap in term of features for the platform. The very first release of the web app might be very skinny in terms on content; the web app itself should be used as a framework and platform by the users who are then creating and expanding the community and the content behind it.

## Background

### The idea behind the application

Even thou there are a lot of existing web applications and website focused on music, the main driver for creating such an app is its goal to merge two of the most diffuse web application types: social media and web blogs/forums. rockonnect would like to offer the best aspects of both those type pf applications in a nice are like the classic rock where people like me, are sometimes struggling to share ideas or hints on musical instruments to use/techniques for playing some songs, for example, and having the possibility to connect with other users having the same struggles or a solution to those, or simply having shared more experience.

So, the “struggle” in finding a unique place as collector and booster of information specific to the genre had an important role in the decision behind the web app ideation. The first step in the research was to see what is available on the market today. I found a lot of applications that are music related mostly radio app broadcasting classic rock. I found also some social network applications, but not really focused on the genre. Most of the web applications on the internet are generic to music or rock and not specific to that genre. The initial release of the web app will look very “naked”. The more the community is growing the more the users of the web app will enrich the web app content.

### Target audience

The specific nature of rockonnect web application might be a good tool for the so called “generation X” [born between 1965 and 1980]. GenX could be the typical target market for the application. These people are usually commuting for work or working from home: most of the time a mix of the two. Their usage of the internet and the device that allow to access it, is different from the “millennial generation”, who are very focused on posting content and not only consuming it. According to some researches[[1]](#footnote-1) “GenX” use the internet for social network, but a lot also for video streaming applications. In addition to that, a personal analysis of the behaviours of sample friends/family that has been run already, has showed that having a tool like rockonnect would mitigate the commuting and having such a platform that mixes social interaction and content focused on a subject that is specific and demanded, would mitigate the day during work pauses [for example].

## Technical Approach

The proposed technical approach is the following:

[1] **Research**: research the problem/objective of the web application: look for what problem the web application would like to solve in first place.

[2] **Analyse** what could be the main 3 features that could be an innovation and start from there. By researching gathering as many as details in terms of requirements: find what is available in the market already what could be missing and fill the gap if it is consistent with the objective of the web application.

[3] **Requirements**: once having a clear understanding of what’s available, what is missing and what could be the main requirements, drafting a concise but very details product specifications. The product specification document must include the main features that the web application has to provide. Including the flows from a user accessing the web application for the first time. So, the flows must include the steps that user has to follow to be able to enjoy the web application to the fullest. So, for example: the registration process, which is mandatory for the users that want to connect to others, needs to be specified in each step and this will be done via the use case scenario description available in the product specification document. Having the clearly specified the objectives upfront, is really the foundation for having a good result and to face the development stage by limiting the issues to face. So, this step is very important for the success of the project.

[4] **Development** using **Scrum** Sprints: the development phase will be sliced into few Sprints [please see section Project Plan] for development and testing. Each sprint will carry a limited number of incremental software features that will be developed, but the expected outcome of each will be very clear from the beginning. The timing scheduled in the project plan for each sprint take into consideration also the aspect of researching for aspects that might be unknown to the developer, debugging and testing. In addition to that, comments in the developed code are important. The web application is not yet started, but somebody else might take over some tasks in the future according to the web app roadmap, therefore clear comments on the developed code are useful to understand the steps and phases of the software generated.

[5] **Reporting** and **issues tracking**: a great tool like the project plan allows the monitoring of issues and tracking of those. Reporting to the supervisor on a constant base allows the development to run a bit more smoothly since reporting allows fine tuning of the application itself.

## Special resources required

There should be no need of use of special resources, but there are few resources on internet that can be used for reference. One of the most supported in terms of community are the mozilla-dev and stackoverflow:

* <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs>: this is a brilliant resource that helped me in the past with great and clear content for building decent web applications. Has a good community behind it.
* Stackoverflow: will require it for any potential issue encountered during the development and the testing
* “Write Modern Web Apps with the MEAN STACK”: author Jeff Dickey, published by Peachpit Press: reading suggested by the supervisor
* Other resources: existing youtube videos on dedicated building web applications and node.js use

## Technical Details

The web application should be written in node.js for both server side and client side. The following is the list of the main frameworks, and libraries that are planned for the development:

* Node.js and npm: server and client side javascript language
* Express: framework for web applications
* Mongo db or firebase db or couchdb or mySQL or JSON for data creation manipulation and storage [CRUD]
* Mongoose [ORM Object Related Mapper] for db communication [in case of mongo db]
* Pug: template engine [for the views]
* Bower: package manager
* jQuery and AJAX: for DOM [Document Object Manipulation]
* bootstrap: for website framework
* bcrypts: for password encryption
* passport: for user registration/login/logoff features
* nodemailer: for email service to users [notifications, account/password retrieval, initial registration and email confirmation etc …]
* other frameworks and libraries as needed

## Evaluation

The platform, as a web application, will be evaluated in a similar way as it will be tested. The plan for evaluating is to create different account with fake names but real emails in order to evaluate the experience for the functional features. As the web application is focused on making real people connect to each other, some information needs to be stored in the web application db. Therefore, the minimum storage info will be also evaluated for the web application to be fully compliant with GDPR regulations. The system test will evaluate that basic functionalities are working [e.g. user registration, login, logoff, etc …]. A further step is evaluating the UX [User Experience] by allowing “friendly” potential users of the web app to use it for the first time. Feedback is important.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature of student and date

# Appendix B: Project Plan

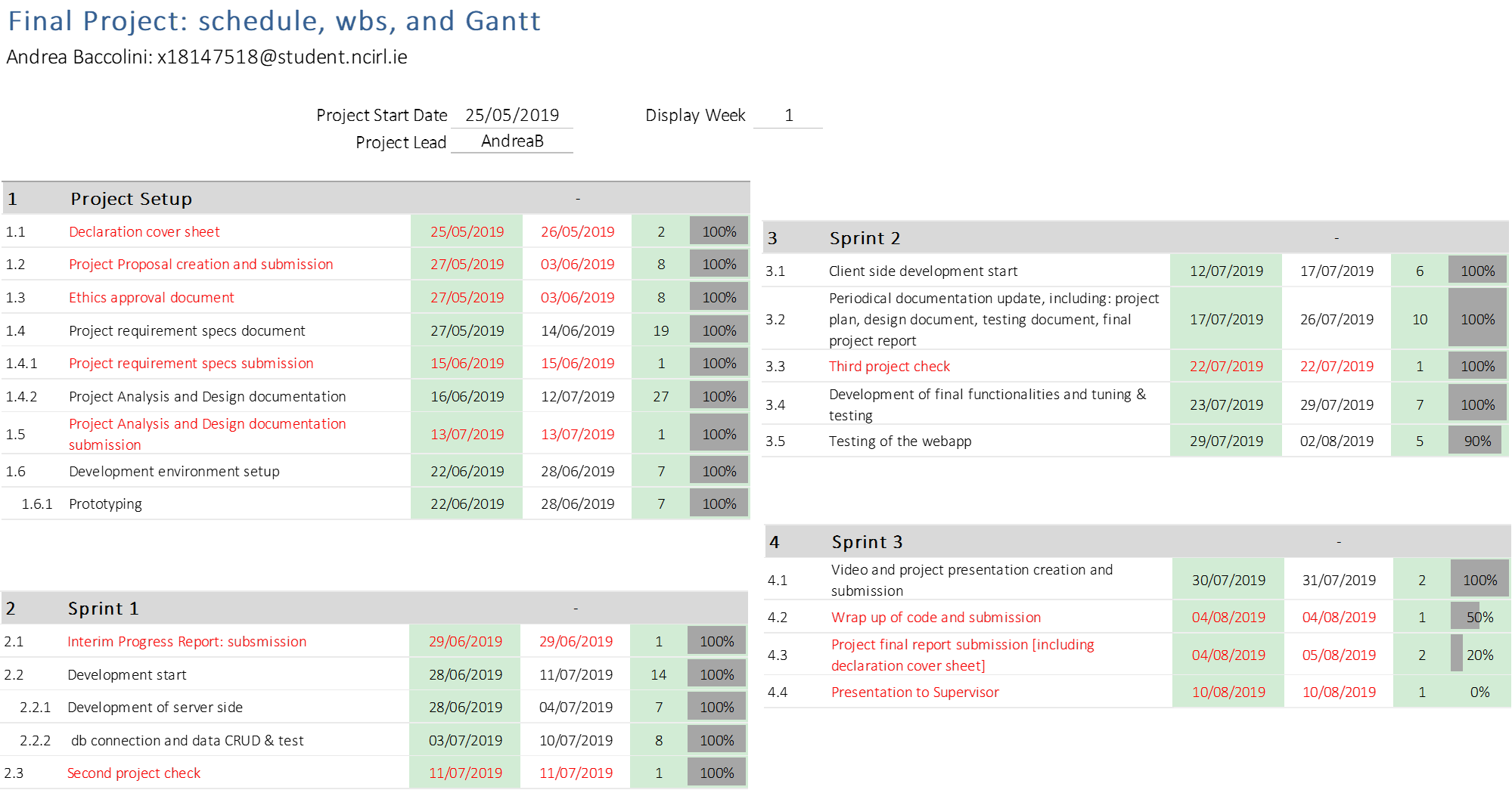
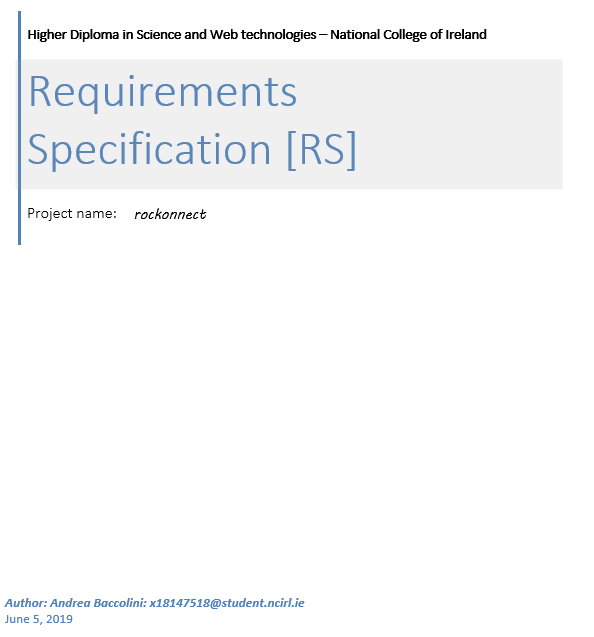


Figure 24 - project plan



Figure 25 – project plan: gant

# Appendix C: Requirement Specification document



Revision History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 04/06/2019 | 1.0 | Document created | A Baccolini | L. McCabe | √ |
| 14/6/2019 | 2.0 | Added section 2.1 and updated use case 3 [section 3.1.3] | A Baccolini |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Distribution List

|  |  |  |
| --- | --- | --- |
| **Name** | **Title** | **Version** |
| Liam McCabe | Supervisor |  |
|  |  |  |
|  |  |  |

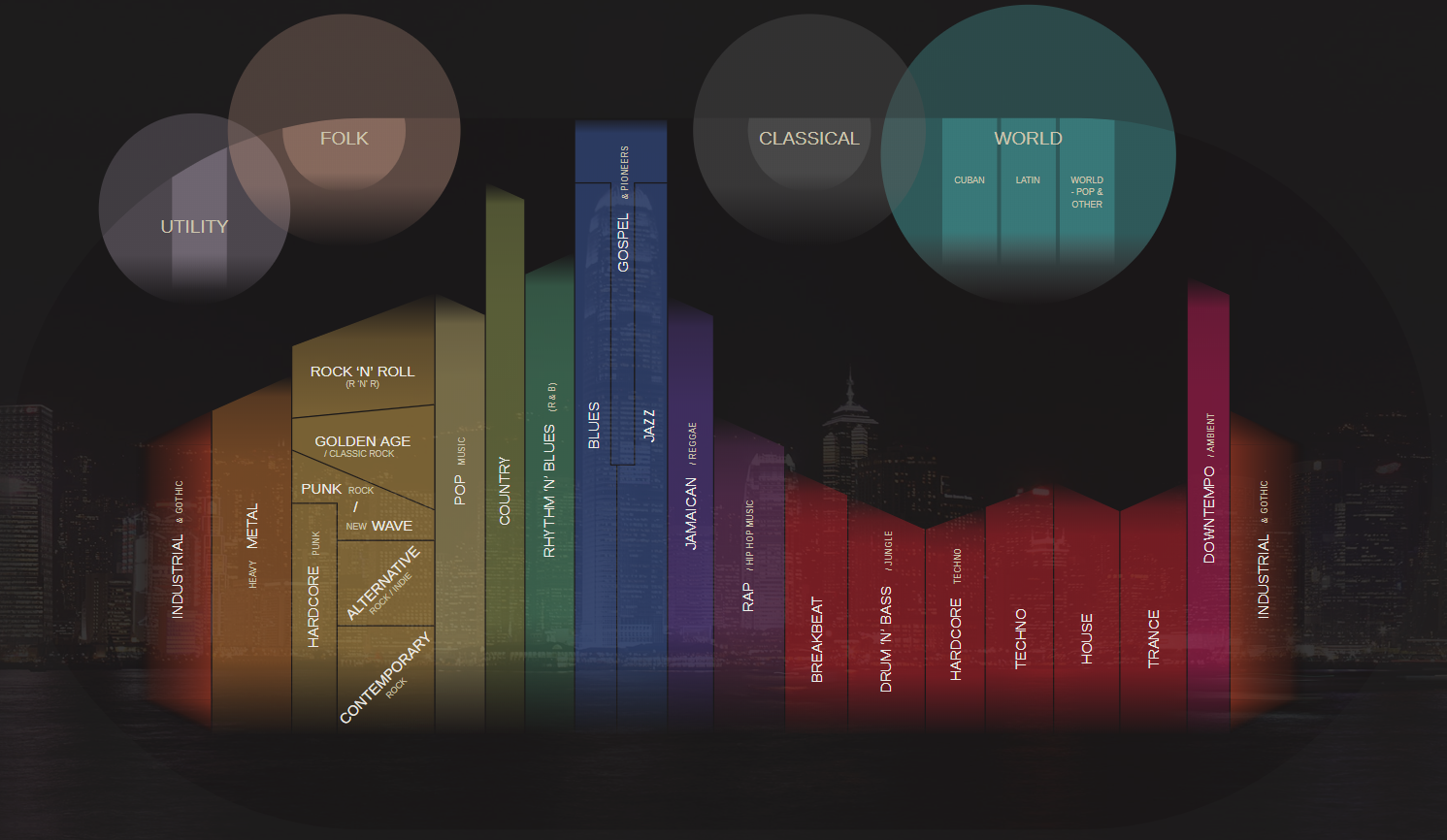
Related Documents

|  |  |
| --- | --- |
| **Title** | **Comments** |
|  |  |
|  |  |

## Introduction

### Purpose

Music and social media seem to be integral part of our daily routine, no matter where we are or what we do. Most of the time the two applications are separated from each because they are serving different goals for the users. Unless there is a specific interest associated with a subject, most of the times a single web application won’t carry social media features and music related features. Main reason would be the fact that music genres are so many that, bringing the two together is not adding much value. On the other side, it is a fact that different generations had experienced and enjoyed different music styles and genres: from classical music to experimental, the genre are too many, as shown in “Figure 26 – The music galaxy” below:



*Figure 26 – The music galaxy*

Therefore, in order to address this gap, this document wants to specify, in the clearest possible manner, the requirements, the layout, the usage flows and user experience of a web application called rockonnect, which is specifically designed to bring together the social media world and the music on a single and specific genre: “Classic Rock”. By “Classic Rock” is intended the music growing in western Europe and US between the early 80s and the late 90s which incorporates and has roots in elements of blues and rock ’n’ roll typical of the 60s, but also psychedelic music, hard rock, punk, grunge, and new wave. rockonnect is an online platform specifically designed for classic rock music lovers who would like to connect to each other in order to exchange ideas, information and any technical matters on the music genre called “Classic Rock”.

The web application is being intended for the so called “Generation X”, as the biggest catchment area, being “GenX” the people born between middle 60s and the early 80s, who were young adults during the “Classic Rock” decades.

### Project Scope

In more details rockonnect allows registered users to enjoy the following features:

[A] features pertaining to a traditional social media app: connect to/follow users that have an interesting profile and create / post user generated content like multimedia videos/pictures of classic rock events/instruments etc...

[B] features proper to a web forum: registered users can hold conversations [“threads”] in the form of posted messages. With this tool, users can exchange opinions and ideas about everything related to classic rock. Some examples are: threads on old published albums; concerts; rock bands performances, musicians; and musical instruments. The threads can be user even to organize and meet for jamming purposes. Anything that is related to classic rock can be discussed in the threads. There will be different user rights: seasoned users will also benefit of the admin rights which allow them to supervise the content and to ban users that might not comply with regulations. Regular users [most of the users of the app] will be able to connect with others but read and write their own content only. rockonnect content is going to be expanded by the community behind it, and the same community will drive the roadmap in term of features for the platform. The very first release of the web app might be very skinny in terms on content; the web app itself should be used as a framework and platform by the users who are then creating and expanding the community. In other words, rockonnect might be seen as a “crowdsourced” online platform as far as content is concerned.

#### What rockonnect will NOT do

What rockonnect will NOT do is to be used as a music file sharing system or for anything that can be subjected to copyright infringement and obscene/illegal material. It is responsibility of the users not to distribute such material through the web application.

Most of the current web applications are focused on the music streaming aspects of the classic rock music. There are plenty of these on the internet, and rockonnect is not going to provide any streaming radio functionality. Might collect a list of already available radio station streaming classic rocks songs, or users are free to exchange links to channels in the thread.

#### Assumptions

The web application needs an active internet connection to function properly, a proper device as client running a standard web browser on a standard operative system: users using different OS and different web browser might incur in a slightly different user experience, but functionally the web app will work in the same way. Different OS and browsers are not part of the final testing.

## User Requirements Definition

In order to give reasons for the requirements, let’s have a brief understanding of the target persona for this web application.

Persona: As defined in Section 1.1, the target for this application is the “GenX” person, who is in her/his 30s and/or 40s: has a job for which commuting might be required; is married and has a moderate usage of the internet, more in favor to getting contents [downloading, streaming or simply browsing] rather than uploading it. When in using internet and device that allows internet access, the persona’s objectives, might be slightly more focused on finding content on specific subjects, rather than uploading and sharing “selfies” or any other potential personal information. A brief interview carried out on a sample of people, belonging to GenX and classic rock music lovers, regarding [A] how they use social media and [B] whether they have an application that allows them to retrieve information [not songs] on their favorite music genre, has revealed that a platform combining A and B, would be greatly used.

Our persona can use multiple devices, such as phone, laptop or tablet, in different times of the day, so the web application needs to be responsive because of this reason.

rockonnect does not require any age verification: users, by signing up the service, they automatically accept the terms and conditions and declare they are 18+ at the time of registration.

Finally: the web application does not store any personal information except the email address. The users’ passwords are encrypted, and the username and name are the only information that are stored in the database together with email address. This info is supplied at the time of registration and is needed for account retrieval once logged in. an external administration service monitors the correct usage of the threads and has the power to delete users that are not comply with web application regulations.

### Technologies used to meet the requirements

A brief separate section on the technologies might be needed to clarify the decision of using them in view of the need in meeting the requirements described in section 3.

For example: as front-end technologies the application uses the most updated bootstraps frameworks together with technologies like “pug” which are making HTML5 easier to be updated and is very user friendly for anything that needs to be added to the webapp in the future. Also, pug is in-line with nodejs kind of syntax and allows create lots of functionalities in the client side, which is very important for social music app like rockonnect.

On a server perspective node.js is the server side [and client side] framework and language that is used. Reason is because not only allows the creation of whatever type of web application, but also it supported by a huge community behind. In addition to that, node.js exposes a lot of libraries and frameworks that can be included in the code for the different purposes; bcryptjs for password encryption/decryption, for example, or passport [within the express framework] for user authentication with standard username and password or with OAuth2 [login with social media credentials]. Another reason of choosing node.js as client/server-side language is that it works very well with JSON style data source such as JSON itself or non-SQL databases like mongodb, which are very popular recently and for the purpose of rockonnect, are a good fit in terms of storing data.

## Requirement specification

After having defined the scope, the target market/persona, and assumptions for the web application to work, the following is the list of functional and non-functional requirements

### Functional requirements

This section lists the functional requirements in **ranked order**.

The system must accomplish the following from a functional perspective:

1. Register to the community
   1. Create profile
2. Sign in the application
   1. Browse and connect to other users
   2. Read threads, comments on threads, like other user’s posts
   3. Initiate new threads
3. Reset password through email
4. Logout and go away

Please refer to *Figure 27 - Use cases diagram* for an understanding of the supported use cases.

A close up of a map

Description automatically generated

*Figure 27 - Use cases diagram*

#### Req 1: registration to the rockonnect community

As a user of the web app I want to be able to register [sign up] to the rockonnect community to enjoy the webapp features.

**Description & Priority**

Users of the webapp, must register before using the platform, registration process involves the creation of the user with username, password, name and email. There is also an about section that must be filled in order to be fully registered to use the application.

**Use Case**

**A close up of text on a white background

Description automatically generated**

*Figure 28 - Use Case 1 - signup*

**Scope**

The scope of this use case is to allow users to register to the online rockonnect community and create a brief profile.

**Description**

This use case describes the procedures to follow in the registration process, including successful registration and unsuccessful registration.

**Flow Description**

**Precondition**

The system is idle, and users can see the web application home page in the web browser.

**Activation**

This use case starts when users click on “signup” button

**Main flow**

1. User clicks on “signup” button
2. The system returns the signup page
3. The user enters signup info: name, username, password email and fill the about part with some personal info, then clicks the submit button
4. The authentication service verifies the details and hashes the password [A1]
5. The authentication service saves the user details [name, username, email, password and about] into the db [local db or remote db]
6. The system returns the sign in page to the user for login purposes

**Alternate flow [1]**

A1: username already present

1. The system alerts the user that the username is already present
2. The authentication service doesn’t save the users details in the db
3. The user is taken to sign up again
4. The use case continues at position 3 of the main flow

**Termination**

The use case terminates when users exit the signup page

**Post condition**

The system goes into a wait state, waiting for user to login

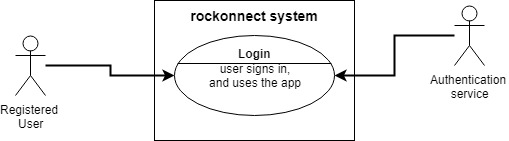
#### Req 2: sign in

**Description & Priority**

Once the user is registered in the community, he/she can sign in to use the features of the web application.

**Use Case**

The use case describes the login procedure for users that are registered to the web application



*Figure 29 - Use case 2 - login*

**Scope**

The scope of this use case is to provide the flow for user login.

**Description**

This use case describes the login procedure after the user is successfully registered.

**Flow Description**

**Precondition**

The system is initialised, and the user has already registered him/herself in the web application. The user is in the login page.

**Activation**

This use case starts when a user tries to login [username and password combination]

**Main flow**

1. The user, in the login page, enters the combination of username and password chosen at the time of registration
2. The authentication service checks the username is present in the database [A1]
3. The authentication service decrypts the password associated with that username
4. The authentication service compares the existing decrypted password with the one inserted by the user at the time of login [A2]
5. If passwords match, the system grants the login to the user

**Alternate flows**

A1: user not registered

1. The authentication service cannot find the user in the db because he/she is not previously registered
2. The system alerts the user of the failed login
3. The system redirects the users to the signup page for registration
4. The use case continues at position 2 of the main flow

A2: password is incorrect

1. The authentication service, by comparing the existing decrypted password with the one inserted by the user at the time of login, verifies they are not the same
2. The system alerts the user of the failed login warning username or password is incorrect
3. The system shows the sign in page for users to login again
4. The use case continues at position 5 of the main flow

**Termination**

The system redirects the logged in user to the welcome page of the webapp.

**Post condition**

The system allows the user to enjoy the features of the web application.

#### Req 3: user connects to other users

**Description & Priority**

For non-registered users, there is the option of seeing the list of users. Once the user is logged in the community, he/she can browse the full profiles of users and connect to others and share the rock experience.

**Use Case**

The use case describes the connection procedure between users that are logged in the web application.

A close up of text on a white background

Description automatically generated

*Figure 30 - Use case 3 – connection between users*

**Scope**

The scope of this use case is to provide the flow for logged in users to connect to other users.

**Description**

This use case describes the login procedure after the user is successfully registered.

**Flow Description**

**Precondition**

The system is initialised, and the user has already registered him/herself in the web application.

**Activation**

This use case starts when a user tries to login [username and password combination]

**Main flow**

1. The user A, browses the list of users
2. The user A clicks on user B profile to see his/her page details
3. The user A tries to connect to user B by adding him/her to the list
4. The system checks if user A is logged in [A1]
5. User A, when logged in, can create his/her own list of friends

**Alternate flows**

A1: user not registered

1. The system detects user A is NOT logged in
2. The system alerts the user with a message to login first before creating the list of “friends”
3. The system redirects the users to the singin page for login
4. The use case continues at position 4 of the main flow

**Termination**

The system presents the list of friends to user

**Post condition**

The system allows the user to enjoy the features of the web application.

#### Req 4: user reads/updates/creates threads

**Description & Priority**

Once the user is logged in the community, he/she can browse the list of users, and connect to others and share the rock experience.

**Use Case**

The use case describes the connection procedure between users that are logged in the web application.

A close up of text on a white background

Description automatically generated

*Figure 31 - Use case 4 – threads*

**Scope**

The scope of this use case is to provide the flow for users to view threads and, if logged in, edit or create new threads.

**Description**

This use case describes the steps that users must follow to view threads and to edit or create new one if they want.

**Flow Description**

**Precondition**

The system is initialised, and the user is on the threads page.

**Activation**

This use case starts when a user is browsing the threads page

**Main flow**

1. The user, under the threads list, browse the threads
2. The user tries to edit an existing thread or create a new one
3. The system checks if the user is logged in
4. The authentication service returns yes or no [A1]
5. If authentication service returns yes, then the system allows the user to edit or create a new thread
6. The thread is created and save in the db
7. The thread is visible in page and correctly updated/created

**Alternate flows**

A1: user not logged in

1. The authentication service returns no, and user is redirected to the login page
2. The system alerts the user of the imminent login procedure
3. Once logged in, the flow continues from number 5 above

**Termination**

The use case terminates when the user leaves the thread page

**Post condition**

The system returns in idle state

#### Req 5: logout

**Description & Priority**

With this requirement users can logout the platform before leaving the application.

**Use Case**

The use case describes the flow that logged in user must follow to correctly logout.

A picture containing text

Description automatically generated

*Figure 32 - Use case 5 – logout*

**Scope**

The scope of this use case is to provide the flow for users to view threads and, if logged in, edit or create new threads.

**Description**

This use case describes the steps that users must follow exit correctly the application before leaving the online platform.

**Flow Description**

**Precondition**

The system is initialised, and the user is logged in.

**Activation**

This use case starts when a logged in user clicks the logout button

**Main flow**

1. The logged in user browses the application pages
2. The user clicks on the logout button
3. The system ends the session of the user
4. The authentication service logs the user out [A1]
5. The user can leave the online platform by closing the browser or moving to another webpage

**Alternate flows**

A1: user not logged in

1. The authentication service cannot logout the user as he/she is not logged in
2. The system alerts the user logout is not possible
3. The use case resume at 5 above

**Termination**

The use case terminates when the user leaves the online platform

**Post condition**

The system returns in idle state

### Non-Functional Requirements

#### Performance/Response time requirement

Although the web application does not require real-time or near real-time responsiveness, the architecture and the technologies used, are cutting edge and designed for improving the responsiveness of the entire webapp. The GUI is developed in such a way to be lightly loadable on every device.

#### Easiness to use

This requirement is about how easy the application needs to be for users. The main reason behind this requirement is the fact that most of the users are music passionate and might not be “computer scientists”, so they would like to get to their objective in less than 3 clicks when possible. So, the UI [User Interface] of the webapp must be designed to satisfy this requirement. Please refer to section 4.1 for the mock-ups of the GUI.

#### Availability requirement

The webapp is available for users that have a device capable to run a browser: devices include laptop/desktops, and mobile devices such as phones and tablets. The other availability requirement is the underlining presence of a broadband internet connection which is properly working.

#### Robustness requirement

The application needs to scale up. There is no limit to the number of users that can register and there is no limit to the number of threads that each single user can create. Also, there is no limit to the number of connections a single user can have within the platform. rockonnect will be distributed in a second or third phase of development, depending on the number of traffic/users that will be supported.

#### Security requirement

Although rockonnect does not store any sensitive user information, the webapp is implemented for being fully secure. User data information are stored in a database and the passwords are encrypted. The database is in a secure location and under a secure cloud infrastructure. The webapp could be deployed in a secure cloud system as well. A roadmap feature for rockonnect is to support HTTPs: please refer to section 6 for a list of roadmap features of the webapp.

#### Reliability requirement

As a pure software product that does not interface any driver of any kernel or operative system, the system relies completely on the availability of the underlined hardware and OS. There is thou a requirement for the application to be on even when users are not using the platform properly, therefore, to test reliability before production, some part of the test needs to include misuse of the platform and negative tests [e.g. recurring login/logoff in short period of time, creation of multiple threads with no content etc…] in order to check the response of the application.

#### Maintainability requirement

The first production release of rockonnect is developed with the latest cutting-edge technologies, and widely used. There is a lot of community behind the technologies used, and a lot of ongoing development which helps the regular maintenance of the web application.

#### Portability requirement

As the target market is using different devices connected to the internet, then portability through responsiveness needs to be implemented, tested, and available from the first release.

#### Reusability requirement

There are few pieces in the software that can reuse for future or existing application. The registration/login/logoff managed by the authentication service is one. It could be reuse for any other project requiring authentication.

## Interface requirements

This section describes how the software interfaces with other software products or users for input or output. Examples of such interfaces include APIs, web services, shared memory, data streams, and so forth. Most systems would have a GUI. Add more subsections for other interfaces as reuired.

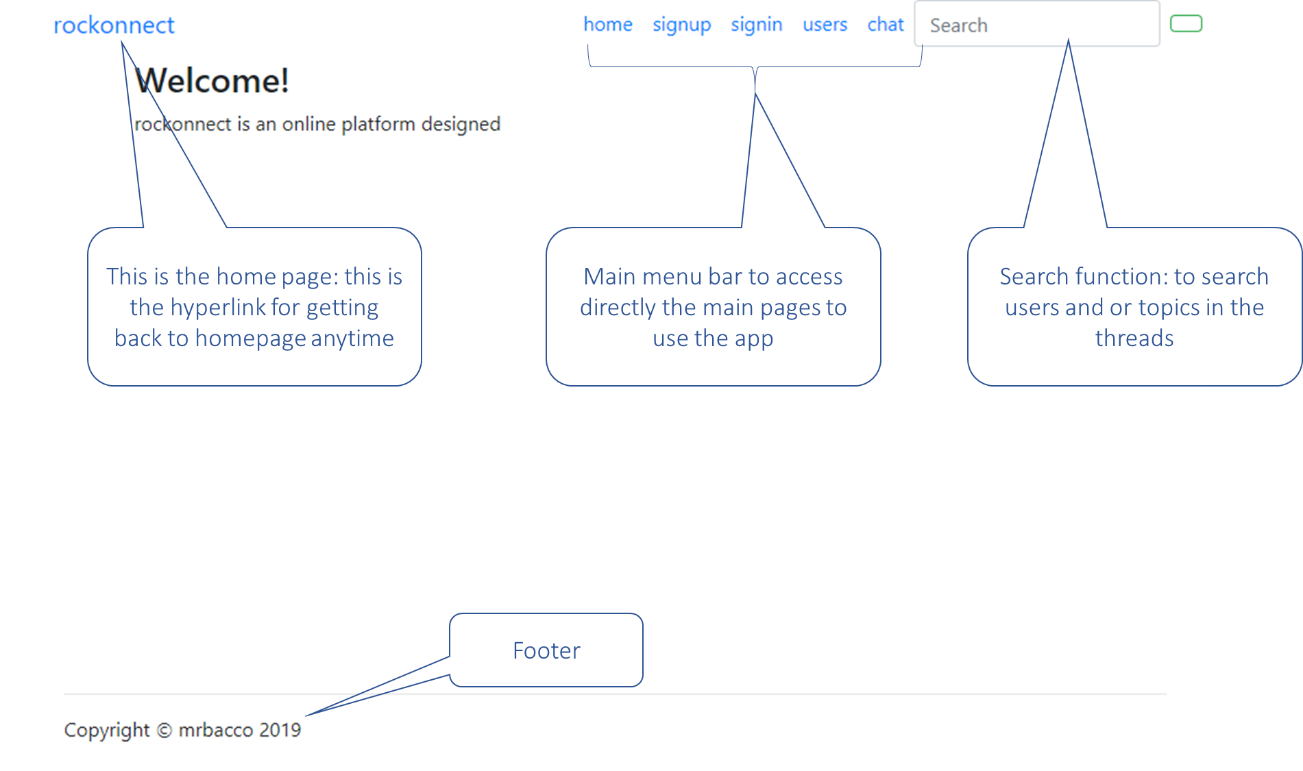
### GUI

This section would like to give more ideas on the GUI and the UX [User Experience] that users will be going through. rockonnect will look like a simple webapp, and very straight forward for user. As one of the non-functional requirements is easiness to use for the users, the GUI is designed in such a way to give the user the easiest user experience and “three clicks” experience.

*Please note that the following pages are mockups and the actual webapp might look a bit different from the mockups, because of different strategies that can be adopted during the development. The layout will not be changing thou.*

#### Home page

New users and registered users can access the webapp from its home page as following:



*Figure 33 – homepage*

#### Sign up

The signup page looks like the following:

A screenshot of a cell phone

Description automatically generated

*Figure 34 - Signup page*

All the fields under the signup page are mandatory. The “About Me” carries a text field for users to insert notes about themselves, so that other registered users can look at the profile. All the information under signup are stored in the database when saved [password encrypted].

#### Sign in

A screenshot of a cell phone

Description automatically generated

*Figure 35 - Sign in page*

Username and password are the fields that users will use to login the webapp.

#### Users

A screenshot of a cell phone

Description automatically generated

*Figure 36 - User list*

Under the “Users” page, users can see the list of users and their profile by clicking on the user’s name.

A screenshot of a cell phone

Description automatically generated

*Figure 37 - User's profile*

#### Chat/Threads

The chat page [threads] is carrying the list of threads that users can browse, and registered users can edit, as following:

A screenshot of a cell phone

Description automatically generated

*Figure 38 - Chat/Threads*

### Application Programming Interfaces [API]

In the first release of the webapp rockonnect, there will be no API towards external systems. In te second release planned for Q4 2019, there will be Oauth2 API for user signin using social media like Facebook, Twitter and Instagram. In the third release planned for Q1 2020 there will be API towards googlemaps for location service: technical details are TBD.

## System Architecture

The architecture is summarized in the following class diagram.

A close up of text on a white background

Description automatically generated

*Figure 39 - class diagram*

The system is designed to be developed as an object-oriented software-based application.

## System Evolution

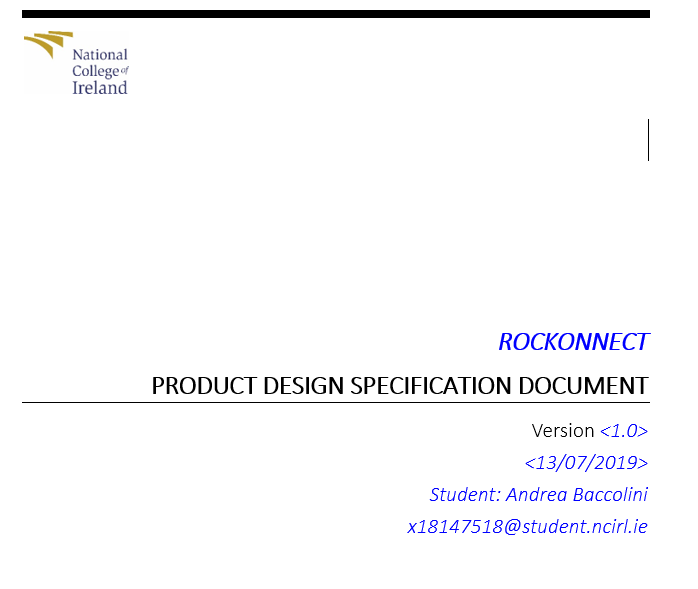
The potential short-term roadmap of the rockonnect features can be the following:

[A] – support of HTTPs: this feature is mandatory in production for the second release of the webapp in order to increase the security. Availability: Q4 2019.

[B] – OAuth2 authentication: users will be able to login using their social media [Facebook, twitter and Instagram] credentials through OAuth2 standard. Availability: Q4 2019.

[C] – location-based search: logged in users will be able to search for other community users by location: something like search for users near me. Availability: TBD.

# Project design and specification document



**VERSION HISTORY**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version #** | **Implemented**  **By** | **Revision**  **Date** | **Approved**  **By** | **Approval**  **Date** | **Reason** |
| 1.0 | *Andrea Baccolini* | *<14/7/2019>* | *Liam McCabe* | *<mm/dd/yy>* | Initial Design Definition draft |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**UP Template Version:** 12/31/07

## Introduction

### Purpose of the Technical Report

This document summarizes and tracks the necessary information required to effectively define architecture and system design in order to give, to the development team, the guidance on architecture of the system to be developed.

## General Overview and Design Guidelines/Approach

This section describes the principles and strategies to be used as guidelines when designing implementing and testing the system.

### Assumptions / Constraints / Standards

As this is a web application and it has distributed components, the first important assumption and dependency is to use the web application in the presence of a reliable underline broadband internet connection, to avoid latency during the HTTP transactions and for the purpose of keeping the page load time as shortest as possible. For this, the web GUI of the application is carrying the minimal information. A broadband internet connection is a requirement also because the users, threads/discussion, users list, passwords tables and all the data are stored in the mongo DB cloud database, so there is the need to access it via internet to be able to use the features of the web app.

There is no specific software requirement for the usage of the web application: the application can be used via the usage of a standard web browser. The responsiveness of the GUI allows the application to be used on desktop as well as tablet/mobile devices. The version of Chrome browser used for development and testing is the 75.

## Architecture Design

This section outlines the system architecture design of the rockonnect web application. The system works as described in ***Figure 40***.

A close up of a map

Description automatically generated

***Figure 40 - System Overview***

### Logical View

The design of rockonnect is based on a hierarchal design that allows users to visualize the content of the app before deciding to sign in and use all the features of the application. So anonymous users, those who are not yet registered/logged in in the app] can see the content but have no ability to edit or add anything. Users need to login to be able to add discussions/ or add friends to their list.

Please see a diagram in ***Figure 41***.

A screenshot of a cell phone

Description automatically generated

***Figure 41 - Hierarchal view of the webapp***

### Software Architecture

rockonnect is implemented following a layered architecture as described in ***Figure 42***.

A screenshot of a cell phone

Description automatically generated

***Figure 42 - layered architecture***

The system architecture is the following:

[A] GUI API: this level allows the users browsers to retrieve all the necessary script from bootstrap, pug, and jquery communities [ (bootstrap, 2018), (jquery, 2019), (pug, 2018)] for the GUI to run properly and render the page formatted as they should be.

[B] Application layer: this layer includes the views and the Pug templates [ (pug, 2018)] used to render and format the pages

[C] Front end engine layer: this is the client server-side code necessary to run functions and it is used as interpreter of the GUI command from the users

[D] Back end engine layer: this is the server side which hosts the server functionalities, like checking usernames existence when the users are registering [each user has a unique username], or granting users login, or encrypting the password before saving the data into the database.

[E] Database API: this layer is making sure that the data are correctly stored in the db and properly retrieved when necessary.

The technologies used for the development are the following:

Node.js for front and back end client and server development

Pug for HTML views templating

Mongodb cloud version for data storage

Vs code as IDE

For a detailed list of the software modules used for the development, please refer to the project proposal document (Baccolini, 2019), section 6.

### Security Architecture

The first release of the webapp might not contains HTTPS support for the underline transaction protocol, as described in ***Figure 40***. But the web app should cover this functionality soon in order to increase the level of security.

Additional security is provided by the password hashing before saving it into the database. In this case, the server side is hashing the password and only the encrypted token would be transferred on the HTTP transaction to the database, making sure the “man in the middle” cannot spoof it.

Another level of security is with registration and login. Users can edit or add content only if they are logged in. Anonymous users can only see the content and cannot edit anything.

On the data protection side, the database is stored in the secure mongodb cloud and it is accessible only by the administrator/development team through a solid HTTPs and a GUI that is not available for web app users. Web app users cannot edit other users’ details on the database nor through the application itself.

In addition to that, in order to avoid any sensitive information leakage and security breach, the application does not require any personal information: age, address, gender etc… are not part of the registration process.

To further improve the security aspect for the web app, the second version of it after the initial release, will consider getting rid of most of the npm libraries that are not vital for the functionalities of the webapp and can be replaced with more secure code.

### Communication Architecture

Communication is performed over the IP network using the HTTP [HTTPs] protocol to handle the transactional parts. Please refer to ***Figure 40*** for an understanding of the

rockonnect runs on the webserver and needs to be reachable from the user’s devices through a stable IP connection which, in most of the cases, is an internet connection. Therefore, a DNS [or FQDN] address needs to be implemented when hosting the application into a production system: this will mostly likely be performed by the hosting service and not by the web app itself. During the development phase a DNS or an FQDN is not needed as the webapp runs locally on the development device or can be reachable through an already provided DNS/FQDN on cloud development environment like CodeAnywhere.

An internet connection is also needed between the web server and the mongodb database, as this latter is hosted in the mongodb atlas cloud. In addition, the necessary hostname and credentials [username and password] information are coded in the application databse.js file and they can be accessed anytime if setting up a new development instance.

Another internet connection is needed for the API communication between the webapp application layer and bootstrap, jquery and pug API as defined in section 8.3.2.

### Performance

There are no specific requirements on high performance web page loading and low latency client-server communications: the only performance requirement is for rockonnect to be able to fully load the webpages when users browse them, therefore the reliability of the underline internet connection is a requirement since some of the user interface layouts are coming from bootstrap, jquery API.

## System Design

### Use-Cases

For the list of use cases, please refer to the Project Specification Document (Baccolini, 2019).

### Database Design

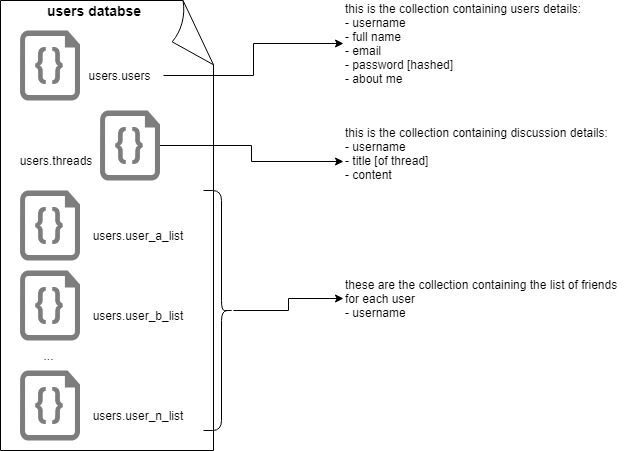
For the data storage the development team has chosen to use a NO SQL database, and the database of choice is mongodb. Major reasons for the usage of mongodb are the following benefits:

Schema less: during the development, the team does not have to care about the database schema anymore and they can dictate whichever schema needed just with the code.

Open source and free

Document based: support dynamic queries and ability to store complex data structures in a JSON or BSON format which is efficient in storage space as well.

The structure is defined in ***Figure 43***.



***Figure 43 - Database architecture***

In the structure used for rockonnect, there is a first database called users, which contains two separate “collections” [which corresponds to the tables in the SQL db syntax]; one for the users of the app, and one for the threads/discussion items. Please refer to ***Figure 43*** for a visual representation of the database structure.

In the NO SQL mongodb database syntax, each “table” is a collection: therefore, the main database used [called users], has three main collections:

* users [users of the webapp]
* threads [discussion forum items]
* user\_list [list of friends for each user]

### Data Conversions

There are few data conversion required for rockonnect to work.

The first and most important one is the conversion of users’ profiles and discussion threads into database collections in BSON format. These are stored in mongodb database called “users”.

The second data conversion is with regards to the password that user chose to use at the time of registration. The code is designed to hash and salt the password before storing it in the collection “users.users” in the mongodb.

### Application Program Interfaces

As shown in ***Figure 40***, rockonnect interfaces bootstrap, jquery and pug API for completing the views on the application layer. Reference for API are the following: (bootstrap, 2018), (jquery, 2019). Bootstrap is used for web pages frameworks and pug is used as HTML templating for editing the views and the code on the web client side.

### User Interface Design

For the user interface screenshots and design, please refer to the following documents:

Project requirement specifications (Baccolini, 2019) and interim progress report (Baccolini, 2019).

### Section 508 Compliance

Rockonnect is accessible to everyone that can have access to a portable device, or a pc or mac running a standard web browser and through a standard broadband internet connection.

# List of figures

[*Figure 1: use cases 9*](#_Toc15893005)

[*Figure 2: thread view requirement use case 10*](#_Toc15893006)

[*Figure 3 - rockonnect architecture diagram 12*](#_Toc15893007)

[*Figure 4 - environment setup 1 14*](#_Toc15893008)

[*Figure 5 - environment setup 2 15*](#_Toc15893009)

[*Figure 6 - routes development example 15*](#_Toc15893010)

[*Figure 7 – public and private routes 16*](#_Toc15893011)

[*Figure 8 - users registration 17*](#_Toc15893012)

[*Figure 9 - user login [passport] 18*](#_Toc15893013)

[*Figure 10 - multer upload 19*](#_Toc15893014)

[*Figure 11 – DB read 19*](#_Toc15893015)

[*Figure 12 - isLoggedIn 19*](#_Toc15893016)

[*Figure 13 - Users view 20*](#_Toc15893017)

[*Figure 14 - Friends db collection 20*](#_Toc15893018)

[*Figure 15 - test results 1 22*](#_Toc15893019)

[*Figure 16- test results 2 23*](#_Toc15893020)

[*Figure 17 - registration page 24*](#_Toc15893021)

[*Figure 18 - users list page 24*](#_Toc15893022)

[*Figure 19 - user profile page 25*](#_Toc15893023)

[*Figure 20 - Discussion forum 25*](#_Toc15893024)

[*Figure 21 - discussion item 25*](#_Toc15893025)

[*Figure 22 - post a thread 26*](#_Toc15893026)

[*Figure 23 - Mean Opinion Score 27*](#_Toc15893027)

[*Figure 24 - project plan 35*](#_Toc15893028)

[*Figure 25 – project plan: gant 36*](#_Toc15893029)

[*Figure 26 – The music galaxy 39*](#_Toc15893030)

[*Figure 27 - Use cases diagram 43*](#_Toc15893031)

[*Figure 28 - Use Case 1 - signup 44*](#_Toc15893032)

[*Figure 29 - Use case 2 - login 45*](#_Toc15893033)

[*Figure 30 - Use case 3 – connection between users 47*](#_Toc15893034)

[*Figure 31 - Use case 4 – threads 48*](#_Toc15893035)

[*Figure 32 - Use case 5 – logout 49*](#_Toc15893036)

[*Figure 33 – homepage 53*](#_Toc15893037)

[*Figure 34 - Signup page 53*](#_Toc15893038)

[*Figure 35 - Sign in page 54*](#_Toc15893039)

[*Figure 36 - User list 54*](#_Toc15893040)

[*Figure 37 - User's profile 55*](#_Toc15893041)

[*Figure 38 - Chat/Threads 55*](#_Toc15893042)

[*Figure 39 - class diagram 56*](#_Toc15893043)

[***Figure 40 - System Overview*** *59*](#_Toc15893044)

[***Figure 41 - Hierarchal view of the webapp*** *60*](#_Toc15893045)

[***Figure 42 - layered architecture*** *61*](#_Toc15893046)

[***Figure 43 - Database architecture*** *63*](#_Toc15893047)

# Other Material Used

Any other reference material used in the project for example evaluation surveys etc.

CD containing code should be glued to the technical report.

# Definitions, Acronyms, and Abbreviations

webapp web application

GenX Generation X

GUI Graphic User Interface

TBD To Be Defined

UX User Experience

UI User Interface

API Application Programming Interface

BSON Binary JSON

CRUD Create, Read, Update, Delete

DNS Domain Name System

FQDN Fully Qualified Domain Name

GUI Graphic User Interface

IDE Integrated Development Environment

MOS Mean Opinion Score

# Bibliography

Baccolini, A., 2019. *InterimReport\_ABACCOLINI\_summer\_2019,* Dublin: s.n.

Baccolini, A., 2019. *Project Requirements Specification Document,* Dublin: s.n.

Baccolini, A., 2019. *Project\_Proposal\_ABACCOLINI\_summer\_2019,* Dublin: s.n.

Baccolini, A., 2019. *Project\_Proposal\_ABACCOLINI\_summer\_2019.docx,* Dublin, Ireland: s.n.

Bootstrap, 2019. *Bootstraps themes.* [Online]   
Available at: https://getbootstrap.com/docs/4.3/examples/  
[Accessed 2019].

bootstrap, c., 2018. *bootstrap.* [Online]   
Available at: https://getbootstrap.com/docs/4.3/getting-started/introduction/  
[Accessed 2019].

Community Node.Js, n.d. *Bcryot.* [Online]   
Available at: https://www.npmjs.com/package/bcrypt

Copes, F., 2019. *The Pug Guide.* [Online]   
Available at: https://flaviocopes.com  
[Accessed 2019].

Dickey, J., 2015. *Write Modern Web Apps eith the MEAN stack.* s.l.:Peachpit press.

Erdeljac, A., 2018. *Learn how to handle authentication with Node using Passport.js.* [Online]   
Available at: https://www.freecodecamp.org/news/learn-how-to-handle-authentication-with-node-using-passport-js-4a56ed18e81e/  
[Accessed 2019].

Geek for Geeks, 2019. *Top 10 programming languages.* [Online]   
Available at: https://www.geeksforgeeks.org/top-10-programming-languages-of-the-world-2019-to-begin-with/

Gunda, D., 2018. *All you need to know about uploading and displaying pictures using Node.js,Express.js,Pug/Jade.* [Online]   
Available at: https://medium.com/@deepika.gunda/all-you-need-to-know-about-uploading-and-displaying-pictures-using-node-js-express-js-pug-jade-d89fbeb19947  
[Accessed 2019].

Hickey, W., 2014. *Classic Rock Started With The Beatles And Ended With Nirvana.* [Online]   
Available at: https://fivethirtyeight.com/features/classic-rock-started-with-the-beatles-and-ended-with-nirvana/  
[Accessed 2019].

Hickey, W., 2014. *The Biggest One-Hit Wonders of Classic Rock.* [Online]   
Available at: https://fivethirtyeight.com/features/the-biggest-one-hit-wonders-of-classic-rock/  
[Accessed 2019].

jquery, c., 2019. *Jquery API.* [Online]   
Available at: https://api.jquery.com/  
[Accessed 2019].

Passport, n.d. *http://www.passportjs.org/docs/downloads/html/.* [Online]   
Available at: http://www.passportjs.org/  
[Accessed 2019].

pug, c., 2018. *pug.* [Online]   
Available at: https://pugjs.org/api/reference.html  
[Accessed 2019].

Pug, C., n.d. *HTML to Pug converter.* [Online]   
Available at: https://html2jade.org/  
[Accessed 2019].

Pug, C., n.d. *Pug.* [Online]   
Available at: https://pugjs.org/api/getting-started.html  
[Accessed 2019].

Rosalie, C., 2019. *Top 10 programming languages from 2019.* [Online]   
Available at: https://www.businessinsider.com/the-10-most-popular-programming-languages-according-to-github-2018-10?r=US&IR=T#2-java-9  
[Accessed 22 1 2019].

Shahid, 2019. *How to Manage Session using Node and Express.* [Online]   
Available at: https://codeforgeek.com/manage-session-using-node-js-express-4/  
[Accessed 2019].

Stackoverflow, 2019. *Stackoverflow.* [Online]   
Available at: https://stackoverflow.com  
[Accessed 2019].

W3School, 1999-2019. *W3Schools learn node.js.* [Online]   
Available at: https://www.w3schools.com/nodejs/default.asp  
[Accessed 2019].

Baccolini, A., 2019. *InterimReport\_ABACCOLINI\_summer\_2019,* Dublin: s.n.

Baccolini, A., 2019. *Project Requirements Specification Document,* Dublin: s.n.

Baccolini, A., 2019. *Project\_Proposal\_ABACCOLINI\_summer\_2019,* Dublin: s.n.

Baccolini, A., 2019. *Project\_Proposal\_ABACCOLINI\_summer\_2019.docx,* Dublin, Ireland: s.n.

Bootstrap, 2019. *Bootstraps themes.* [Online]   
Available at: https://getbootstrap.com/docs/4.3/examples/  
[Accessed 2019].

bootstrap, c., 2018. *bootstrap.* [Online]   
Available at: https://getbootstrap.com/docs/4.3/getting-started/introduction/  
[Accessed 2019].

Community Node.Js, n.d. *Bcryot.* [Online]   
Available at: https://www.npmjs.com/package/bcrypt

Copes, F., 2019. *The Pug Guide.* [Online]   
Available at: https://flaviocopes.com  
[Accessed 2019].

Dickey, J., 2015. *Write Modern Web Apps eith the MEAN stack.* s.l.:Peachpit press.

Erdeljac, A., 2018. *Learn how to handle authentication with Node using Passport.js.* [Online]   
Available at: https://www.freecodecamp.org/news/learn-how-to-handle-authentication-with-node-using-passport-js-4a56ed18e81e/  
[Accessed 2019].

Geek for Geeks, 2019. *Top 10 programming languages.* [Online]   
Available at: https://www.geeksforgeeks.org/top-10-programming-languages-of-the-world-2019-to-begin-with/

Gunda, D., 2018. *All you need to know about uploading and displaying pictures using Node.js,Express.js,Pug/Jade.* [Online]   
Available at: https://medium.com/@deepika.gunda/all-you-need-to-know-about-uploading-and-displaying-pictures-using-node-js-express-js-pug-jade-d89fbeb19947  
[Accessed 2019].

Hickey, W., 2014. *Classic Rock Started With The Beatles And Ended With Nirvana.* [Online]   
Available at: https://fivethirtyeight.com/features/classic-rock-started-with-the-beatles-and-ended-with-nirvana/  
[Accessed 2019].

Hickey, W., 2014. *The Biggest One-Hit Wonders of Classic Rock.* [Online]   
Available at: https://fivethirtyeight.com/features/the-biggest-one-hit-wonders-of-classic-rock/  
[Accessed 2019].

jquery, c., 2019. *Jquery API.* [Online]   
Available at: https://api.jquery.com/  
[Accessed 2019].

Passport, n.d. *http://www.passportjs.org/docs/downloads/html/.* [Online]   
Available at: http://www.passportjs.org/  
[Accessed 2019].

pug, c., 2018. *pug.* [Online]   
Available at: https://pugjs.org/api/reference.html  
[Accessed 2019].

Pug, C., n.d. *HTML to Pug converter.* [Online]   
Available at: https://html2jade.org/  
[Accessed 2019].

Pug, C., n.d. *Pug.* [Online]   
Available at: https://pugjs.org/api/getting-started.html  
[Accessed 2019].

Rosalie, C., 2019. *Top 10 programming languages from 2019.* [Online]   
Available at: https://www.businessinsider.com/the-10-most-popular-programming-languages-according-to-github-2018-10?r=US&IR=T#2-java-9  
[Accessed 22 1 2019].

Shahid, 2019. *How to Manage Session using Node and Express.* [Online]   
Available at: https://codeforgeek.com/manage-session-using-node-js-express-4/  
[Accessed 2019].

Stackoverflow, 2019. *Stackoverflow.* [Online]   
Available at: https://stackoverflow.com  
[Accessed 2019].

W3School, 1999-2019. *W3Schools learn node.js.* [Online]   
Available at: https://www.w3schools.com/nodejs/default.asp  
[Accessed 2019].

1. https://www.social4retail.com/how-digital-behavior-differs-among-millennials-gen-xers-and-boomers.html# [↑](#footnote-ref-1)