|  |
| --- |
| ***Author: Andrea Baccolini: x18147518@student.ncirl.ie***  June 5, 2019 |



|  |  |
| --- | --- |
| **Higher Diploma in Science and Web technologies – National College of Ireland** | |
| Requirements Specification [RS] | |
| Project name: | rockonnect |

Requirements Specification (RS)

Document Control

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** |
|  |  |
|  |  |

**Table of Contents**

[Requirements Specification (RS) 1](#_Toc11399871)

[1 Introduction 4](#_Toc11399872)

[1.1 Purpose 4](#_Toc11399873)

[1.2 Project Scope 5](#_Toc11399874)

[1.2.1 What rockonnect will NOT do 5](#_Toc11399875)

[1.2.2 Assumptions 5](#_Toc11399876)

[1.3 Definitions, Acronyms, and Abbreviations 5](#_Toc11399877)

[2 User Requirements Definition 6](#_Toc11399878)

[2.1 Technologies used to meet the requirements 6](#_Toc11399879)

[3 Requirement specification 7](#_Toc11399880)

[3.1 Functional requirements 7](#_Toc11399881)

[3.1.1 Req 1: registration to the rockonnect community 8](#_Toc11399882)

[3.1.2 Req 2: sign in 10](#_Toc11399883)

[3.1.3 Req 3: user connects to other users 11](#_Toc11399884)

[3.1.4 Req 4: user reads/updates/creates threads 13](#_Toc11399885)

[3.1.5 Req 5: logout 14](#_Toc11399886)

[3.2 Non-Functional Requirements 15](#_Toc11399887)

[3.2.1 Performance/Response time requirement 15](#_Toc11399888)

[3.2.2 Easiness to use 15](#_Toc11399889)

[3.2.3 Availability requirement 15](#_Toc11399890)

[3.2.4 Robustness requirement 15](#_Toc11399891)

[3.2.5 Security requirement 16](#_Toc11399892)

[3.2.6 Reliability requirement 16](#_Toc11399893)

[3.2.7 Maintainability requirement 16](#_Toc11399894)

[3.2.8 Portability requirement 16](#_Toc11399895)

[3.2.9 Reusability requirement 16](#_Toc11399896)

[4 Interface requirements 16](#_Toc11399897)

[4.1 GUI 16](#_Toc11399898)

[4.1.1 Home page 17](#_Toc11399899)

[4.1.2 Sign up 17](#_Toc11399900)

[4.1.3 Sign in 18](#_Toc11399901)

[4.1.4 Users 19](#_Toc11399902)

[4.1.5 Chat/Threads 19](#_Toc11399903)

[4.2 Application Programming Interfaces [API] 20](#_Toc11399904)

[5 System Architecture 20](#_Toc11399905)

[6 System Evolution 21](#_Toc11399906)

[7 Appendix: list of figures 22](#_Toc11399907)

# 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 1 – The music galaxy” below:

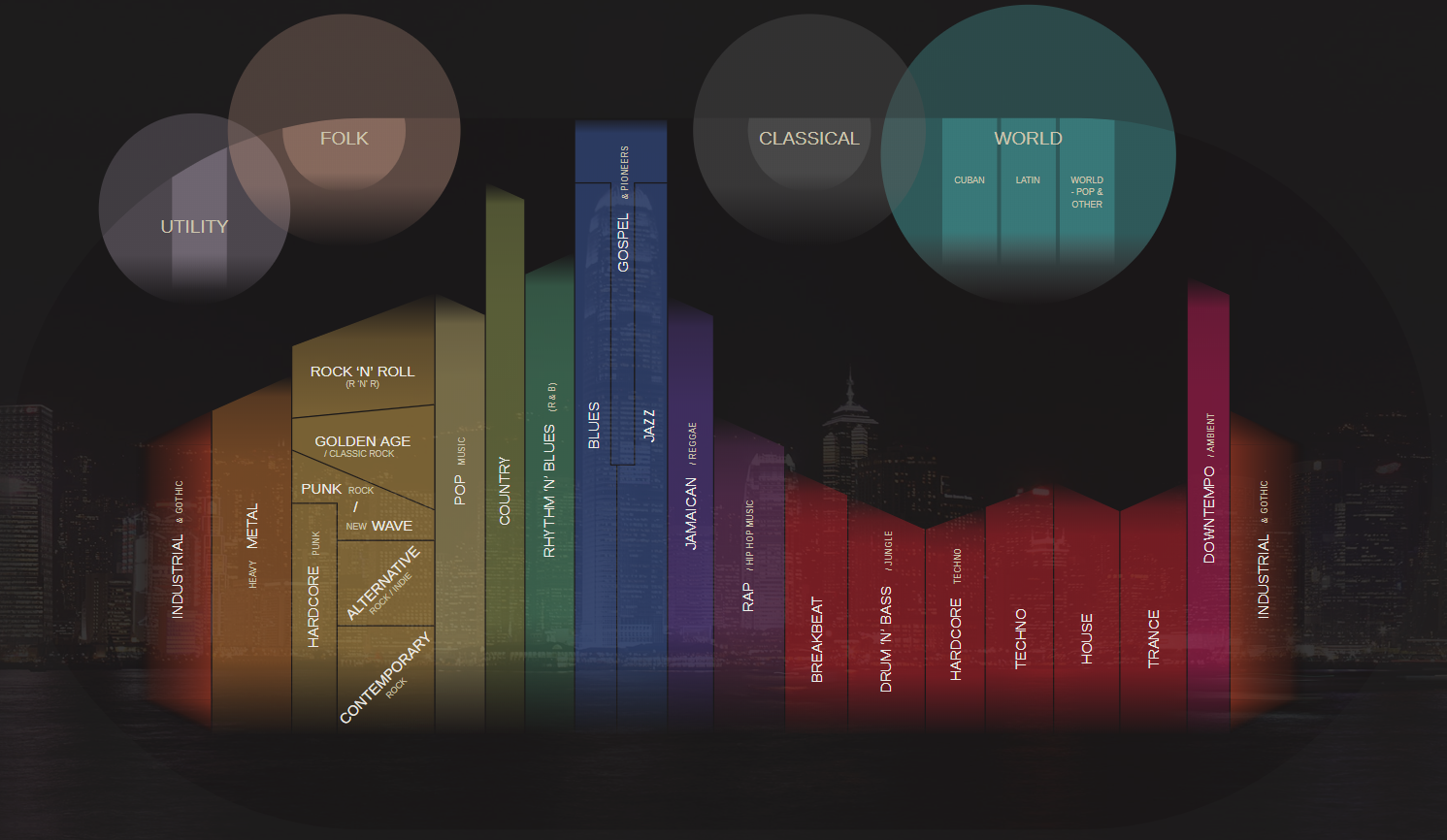


Figure 1 – The music galaxy

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.

## Definitions, Acronyms, and Abbreviations

webapp web application

GenX Generation X

GUI Graphic User Interface

TBD To Be Defined

UX User Experience

UI User Interface

……

# 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 2 - Use cases diagram for an understanding of the supported use cases.

A close up of a map

Description automatically generated

Figure 2 - 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 3 - 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

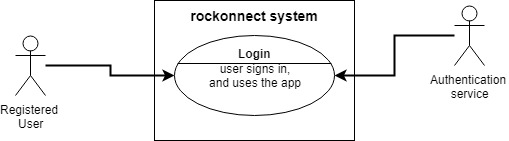


Figure 4 - Use case 2 - login

**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 5 - 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 6 - 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 7 - 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:

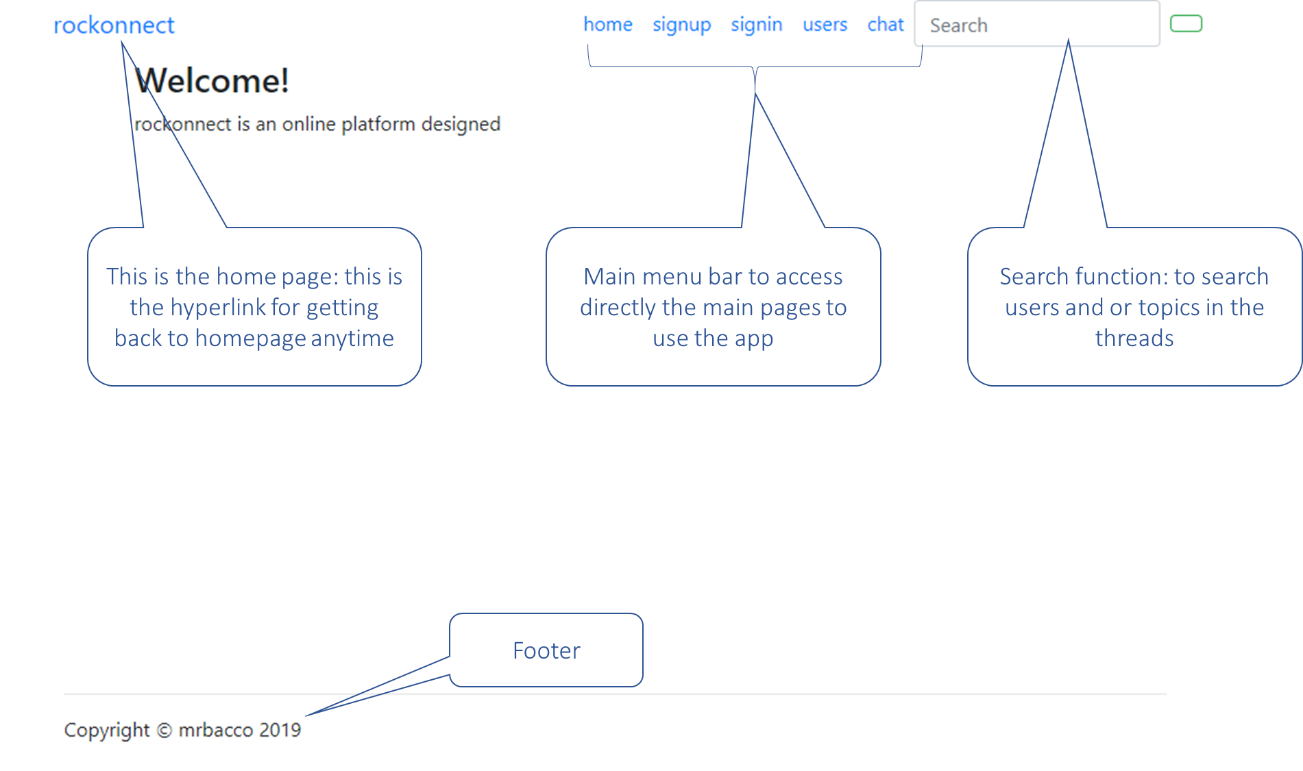


Figure 8 – homepage

### Sign up

The signup page looks like the following:

A screenshot of a cell phone

Description automatically generated

Figure 9 - 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 10 - 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 11 - 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 12 - 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 13 - 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 14 - 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.

# Appendix: list of figures

[Figure 1 – The music galaxy 4](#_Toc11399908)

[Figure 2 - Use cases diagram 8](#_Toc11399909)

[Figure 3 - Use Case 1 - signup 9](#_Toc11399910)

[Figure 4 - Use case 2 - login 10](#_Toc11399911)

[Figure 5 - Use case 3 – connection between users 12](#_Toc11399912)

[Figure 6 - Use case 4 – threads 13](#_Toc11399913)

[Figure 7 - Use case 5 – logout 14](#_Toc11399914)

[Figure 8 – homepage 17](#_Toc11399915)

[Figure 9 - Signup page 18](#_Toc11399916)

[Figure 10 - Sign in page 18](#_Toc11399917)

[Figure 11 - User list 19](#_Toc11399918)

[Figure 12 - User's profile 19](#_Toc11399919)

[Figure 13 - Chat/Threads 20](#_Toc11399920)

[Figure 14 - class diagram 21](#_Toc11399921)