

*rockonnect*

Product Design Specification Document

Version *<1.0>*

*<13/07/2019>*

*Student: Andrea Baccolini*

*x18147518@student.ncirl.ie*

VERSION HISTORY

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

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

TABLE OF CONTENTS

[1 Introduction 4](#_Toc14019200)

[1.1 Purpose of The Product Design Specification Document Document 4](#_Toc14019201)

[2 General Overview and Design Guidelines/Approach 4](#_Toc14019202)

[2.1 Assumptions / Constraints / Standards 4](#_Toc14019203)

[3 Architecture Design 4](#_Toc14019204)

[3.1 Logical View 5](#_Toc14019205)

[3.2 Software Architecture 5](#_Toc14019206)

[3.3 Security Architecture 6](#_Toc14019207)

[3.4 Communication Architecture 7](#_Toc14019208)

[3.5 Performance 7](#_Toc14019209)

[4 System Design 7](#_Toc14019210)

[4.1 Use-Cases 7](#_Toc14019211)

[4.2 Database Design 7](#_Toc14019212)

[4.3 Data Conversions 8](#_Toc14019213)

[4.4 Application Program Interfaces 8](#_Toc14019214)

[4.5 User Interface Design 8](#_Toc14019215)

[4.6 Section 508 Compliance 9](#_Toc14019216)

[5 List of figures 9](#_Toc14019217)

[6 Bibliography 9](#_Toc14019218)

[7 Acronyms 9](#_Toc14019219)

# Introduction

## Purpose of The Product Design Specification Document Document

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 1.

A close up of a map

Description automatically generated

Figure 1 - 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 2.

A screenshot of a cell phone

Description automatically generated

Figure 2 - Hierarchal view of the webapp

## Software Architecture

rockonnect is implemented following a layered architecture as described in Figure 3.

A screenshot of a cell phone

Description automatically generated

Figure 3 - 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 1. 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 1 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 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 4.

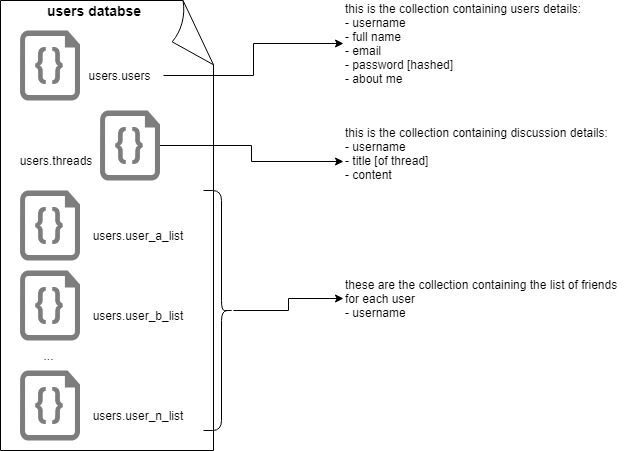


Figure 4 - Database architecture

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 4 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 1, 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 - System Overview 4](#_Toc14019220)

[Figure 2 - Hierarchal view of the webapp 5](#_Toc14019221)

[Figure 3 - layered architecture 6](#_Toc14019222)

[Figure 4 - Database architecture 8](#_Toc14019223)

# 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.

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

Dickey, J., 2014. *Write Modern Web Apps with the MEAN Stack.* s.l.:peachpit press.

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

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

# Acronyms

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

webapp Web Application