UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Master in Computer Engineering: Web Technology and Digital Business

Puzzle game online running on a bidirectional communication framework based on Node.js and Socket.io

Master's Final Project

by

Johanna Mesa Ramos
University of Las Palmas de Gran Canaria
June 2016

Supervisor

Cayetano Guerra Artal
Computer Science and Artificial Intelligence

"Everybody wanna be a bodybuilder, but don't nobody wanna lift no heavy ass weight."

- Ronnie Coleman

ACKNOWLEDGEMENTS

It was back in 2013 when I found myself writing the acknowledgements of my then end of career project. Today I am doing it again, but now for my master's degree. I have to thank the University of Las Palmas de Gran Canaria (ULPGC) for this, not only for their installment of the master's program, but also for bestowing upon me the scholarship that has allowed me to undertake such a fantastic opportunity. From the bottom of my heart, thank you.

There is also a thank you in store for my supervisor, Cayetano Guerra Artal. I do appreciate all the guidance you have provided along these past few months, helping me shape the idea I had into an actual master's final project.

I cannot go without showing my gratitude towards the university's administration staff. They have always lent a helping hand since I started out my career path back in 2009, they are an absolutely amazing team, thank you so much.

Last but not least, I want to thank my family and my other half. They are the ones who keep me sane and who encourage me to finish strongly whatever project I happen to have undertaken. I love you mami, I love you dad, I love you witcher.

[This page is intentionally left blank.]

TABLE OF CONTENTS

TABLE OF ILLUSTRATIONS

TABLE OF UML DIAGRAMS

TABLE OF RESULTS

SUMMARY

The goal was to dive into the JavaScript runtime that is Node.js, to do so the following idea was procured: build a framework that offers a real-time bidirectional communication service, without clients having to install Node.js in their servers. In addition, a basic JavaScript game, that uses the service, was programmed following the book "Building JavaScript Games: for Phones, Tablets and Desktop" [3].

The framework was built through the phases of analysis, design, and implementation. Once it was well established, the game became the focus of attention for which the following points were carried out: game idea conception, design and production.

Finally, a basic rough draft of a business plan was generated for the service that the developed framework offers.

RESUMEN

El objetivo era adentrarse en el sistema en tiempo de ejecución de JavasScript que es Node.js, para ello se manifestó la siguiente idea: implementar un framework que otorgue servicio de comunicación bidireccional en tiempo real, sin que los clientes tengan que instalar Node.js en sus servidores. Además, se desarrolló un juego en JavaScript que usa dicho servicio, siguiendo el libro "Building JavaScript Games: for Phones, Tablets and Desktop" [3].

El framework se desarrolló siguiendo las fases de análisis, diseño e implementación. Cuando quedó consolidado, el juego pasó a ser el foco de atención, realizando los siguientes puntos: concepción de la idea del juego, diseño y producción.

Finalmente, un plan de negocio básico, que describe el servicio de comunicación, fue generado.

CHAPTER 1 - INTRODUCTION

Chapter 1 presents the following: document structure, current state and goals, justification of the specific skills covered, contributions to both technical and personal environment, and a brief mention of the outcomes.

During the course of the master's degree Node.js was briefly looked into. This new technology offers a whole plethora of possibilities. Once the program finished, it was decided that the master's final project would delve into Node.js further to expand the student's knowledge about the subject. The idea was to create a framework that offers a real-time bidirectional communication service, making it possible to use some of the powerful open source libraries available in Node.js' package ecosystem, npm. Moreover, because the student was also eager to learn about game development, it was decided that a basic JavaScript game that utilizes the built framework could be developed as well. This document encompasses the student's work in developing both the framework and the game.

1.1 DOCUMENT STRUCTURE

The document is structured in chapters, made up of sections, so as to portray the information lineally, as follows:

- Chapter 1
  This chapter's main focus is the project's introduction through the state of the art, goals, justification of the specific skills covered, contributions to both technical and personal environment, and a brief mention of the outcomes.

- Chapter 2
  The second chapter delves into development of both the framework and the game.

- Chapter 3
  The final results of both the game and Bicomm (the framework) are portrayed through images in this chapter.

- Chapter 4
  The business plan that looks to identify the profitable activity that is Bicomm's service is found in this chapter.

- Chapter 5
  The last chapter encloses the student's conclusions and future of the project.

1.2 STATE OF THE ART

FRAMEWORK

Upon building web applications, one often faces the fact that HTTP (Hypertext Transfer Protocol) functions as a request-response protocol in the client-server model. On one hand, a client issues a request, on the other hand the server receives and processes said request, sending back the response. If the client carries out future requests, they will not be related to the previous ones. A server cannot send the client anything unless the client first sends a request. If the goal is for the server to communicate with the client without them sending a request first then it must be implemented by one of the following methods:

- Polling: client constantly asks the server if there is any data.
  The problem with this is that the server still relies on the client initiating the connection. If the server were to have data during the polling interval (time the client waits between making polls) it would still have to wait for the client's poll, ruining the 'real-time' requirement.

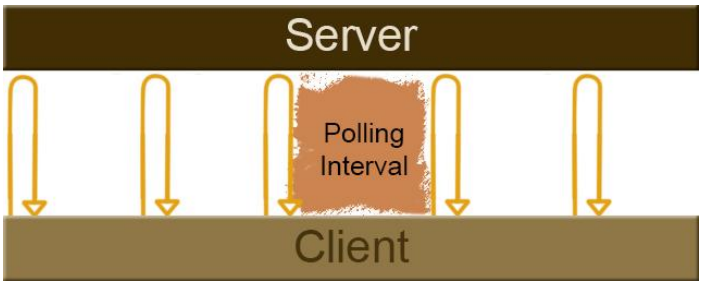


Illustration 1. Client-Server polling.

One way to fix that would be to lower the polling interval, but that would create an overhead on the server side.

Another way is to use the technique called "long polling". A client opens an Ajax-based connection to the server, but the server does not reply until it has data. While this sounds like the sure way, it presents its own set of problems. The amount of requests the server can handle lessens with every new request because they remain open until the server has a response for it.

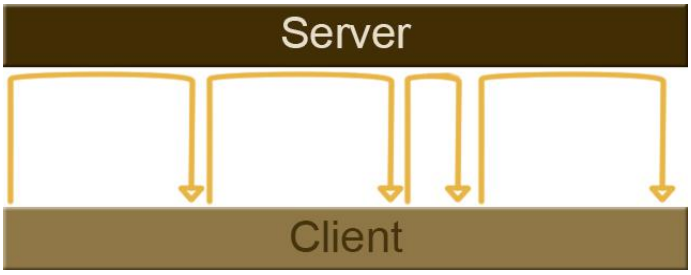


Illustration 2. Client-Server long polling.

- HTML5 WebSockets.

  The aforementioned techniques only simulate full duplex communication, with each having their share of disadvantages. Along with HTML5 came the solution, WebSockets. They offer a real bidirectional TCP (Transmission Control Protocol) connection between the server and the client. In short, to establish a WebSocket connection the client sends a WebSocket handshake request over HTTP, the server responds with the details on how to open the TCP connection. As simple as that. Why is this the solution? Because TCP is a connection-oriented protocol, in other words, a connection is first established and maintained until both ends of said connection have finished exchanging as many messages as they need. In essence, TCP is a stateful protocol.

  It is a promising way out, almost all major browsers support this (starting at a certain version) as of today[1]. There are issues with some proxy servers though. If the WebSocket protocol client implementation detects that the user agent is configured to use a proxy when connecting to its destination it should use the HTTP CONNECT method to set up a persistent tunnel, using an encrypted WebSocket connection. It is the feature of an HTTP-compatible handshake that allows an HTTP server to share its default HTTP and HTTPS ports (80 and 443) with a WebSocket gateway or server. Nonetheless, some proxies need to upgrade to support WebSockets, or need further configuration to not cause the connections to fail, especially if said connection is unencrypted. The client side implementation of the WebSocket protocol is done via JavaScript and the server side implementation depends on the language used (Java, Ruby on Rails, C++, .Net, JavaScript ... etc).

  The framework in this project is built using an HTML5 WebSockets implementation.

[1]" Web Sockets", Can I use, http://caniuse.com/#feat=websockets (June 01, 2016)

These are the techniques available to carry out real-time bidirectional communication. Anyone who would want to develop a service that requires this kind of communication would also have to implement one of these techniques or all of them (to allow a graceful downgrade for clients who have browser versions that do not support WebSocket connection).

GAME

Thanks to browsers and web technologies thousands of online games are playable as of today. A browser game is a computer game that is meant to be played on a browser, over the internet. The genres of said games range from puzzle all the way to adventure, appealing to both regular and casual players. Some famous free to play game sites are: Kongregate[2], Armor Games[3], and Addicting Games[4].

Illustration 4. Kongregate.

Illustration 3. Armor Games.

Illustration 5. Addicting Games.

[2] Kongregate , http://www.kongregate.com/ , (June 01, 2016)
[3] Armor Games, http://armorgames.com/ , (June 01, 2016)
[4] Addicting Games, http://www.addictinggames.com/ , (June 01, 2016)

1.3 GOALS

FRAMEWORK

Even if a service to be developed only needs basic communication, one of the techniques explained in section 1.2 would still have to be used. Seems like a lot to implement just to pass some messages between a server and a client. That is where this project's framework goal comes in:

> Set up a system in JavaScript, using Node.js, that will offer real-time bidirectional communication without the client having to implement these protocols nor use anything other than simple JavaScript on the client side. The client will sign up in the system, download the necessary JavaScript library, hook it up to their service, initialize basic parameters and they will be well on their way to real-time full duplex communication with no effort at all.

GAME

This project's game objective is as follows:

> Develop a browser game where the player may choose whether they want their controls to be within the game (touch screen or PC mouse) or within another device that has access to internet and a browser. The option of using controllers on a different device will be possible thanks to the framework built in this assignment. In order to make the most of said framework, multiplayer mode will also be implemented. The student will learn about OOP (Object Oriented Programming) in JavaScript and how to correctly separate classes to develop a reusable basic game engine.

The idea of a game with the controllers on different devices already exists and its fantastic show case can be found in "Chrome Experiments: Mobile Controller Experiments" [5]. Before coming up with the idea, the student did not know that Google was already experimenting on the matter.

[5] "Mobile Controller Experiments", Chrome Experiments, https://www.chromeexperiments.com/mobile-controller , (June 1, 2018)

1.4 SPECIFIC SKILLS COVERED

The master's degree that requires this thesis has a pool of specific skills that the graduating student must acquire. In this section only a few of said skills will be exposed, explaining how the thesis covers them. These skills' definition have been obtained from the official page of the Computer Engineering School of Las Palmas de Gran Canaria[6].

TI01

Definition:

> Ability to model, design, define an architecture, implement, administer, operate, manage and maintain applications, networks, systems, services and software content.

The TI01 skill has been adequately covered due to the student's work in:

- ✓ Designing, defining the architecture, and implementing the framework that offers a distinct communication service.
- ✓ Modeling, designing, and implementing the use of the framework in a JavaScript game by manipulating the developed game engine.

TI02

Definition:

> Ability to understand and know how to apply the functioning and organization of the Internet, the technologies and network protocols of the new generation, the component models, middleware and services.

This skill has been sufficiently covered because of the framework's nature:

- ✓ The framework's service is offered through the Internet, using an open source library that manages both HTML5 WebSockets and long polling to set up a two-way communication line between server and client.

[6] "Máster en Ingeniería Informática. Escuela de Ingeniería Informática Universidad de Las Palmas de Gran Canaria" EII, http://eii.ulpgc.es/archivos/MasterenIngenieriaInformatica2014.pdf, (June 1, 2016)

- ✓ For the service to work properly one must understand what a cross-domain solution is and its restrictions.

TI04

Definition:

> Ability to design, develop, manage and evaluate certification mechanisms and guarantee the security in treatment and access of information in a local processing or distributed system.

This skill is met during the development of the framework:

- ✓ For the clients (other developers) to access the framework's service, it is necessary to sign into the system and download the necessary libraries. The sign in process was developed from scratch, understanding the importance of encrypting sensible information and setting cookies only in appropriate domain areas to sustain sessions.

1.5 CONTRIBUTIONS

TECHNICAL ENVIRONMENT

At the time the project was developed it was not possible to find a service that would allow a client to have real-time bidirectional communication without having to install specific technology such as Node.js. A quick Google search will show how to carry out said communication, but not a service that already provides it without having to install any additional software.

The developed framework opens up doors to any client, independently of whatever server side scripting technique they are using (PHP, ASP, Groovy, Java, Phyton...etc), who wants real-time bidirectional communication. The client need only sign into the system, download the necessary JavaScript libraries and set up a few parameters to be well on their way. The framework lets the client concentrate on their personal project rather than use up precious time figuring out how to manage the communication.

PERSONAL

This project has given me the opportunity to delve into Node.js and the wondrous open source library that is Socket.io. I was also able to dabble some into a document-

oriented database, MongoDB. I am still amazed at the fact that JavaScript can now be used in a server. Not only that, thanks to knowing about the technology through the master's degree I was able to soon apply it practically in my job. I have also gone on to use Electron, a framework that lets you use Node.js and its open source libraries to build native desktop applications. Essentially Electron takes care of the hard parts while letting you focus on the core of the application.

I was also very interested in building a game from scratch in order to understand its process and structure. The I book I used as a guide was extremely helpful in providing adequate direction, building games iteratively, adding more functionality within each iteration. The book also promoted good coding practices and principles such as refactoring and proper encapsulation to give way to reusability.

My JavaScript skills have immensely improved thanks to this experience, I am proud to know such a powerful and versatile programming language.

1.6 RESULTS

In this section a brief mention of the project's results will be presented.

The framework, named Bicomm (allusion to bidirectional communication) was successfully built and the service setup was less complicated than initially expected. There is an important problem in the business plan that will be discussed in the last chapter. Bicomm is accessible through the following web address until August 30th of 2016: http://bicomm.noip.me:8888/signup . To access the library one must sign up using one of the following invitations: hellowehopeyoustay123, welcomenewcomer456, ahoymatey987, ahbutyouhaveheardofme2, and butwhyistherumgone0. If you find that all invitations have been used up, then feel free to use the following user:

username: iamterry
password: 123

The finished game can be played in the following address until August 30th of 2016: http://woad.es/ . To experience the part of the game that uses the Bicomm service you must choose for the controls to be on "Phone". There you may choose either single or multi player mode.

The basic game engine developed for this game will be kept for future personal references.

CHAPTER 2 - DEVELOPMENT

In this chapter the development of both the framework and the game will be presented through development methodology, requirements specification, design and the user manual.

2.1 METHODOLOGY

It is of utmost importance to rely on a proved working methodology while carrying out a project. A methodology helps carry out the project with the expected results, making sure the software not only does things properly but also does what it is meant to do. This section will briefly present the methodology used in this thesis and with what tools were the requirements and architecture defined.

INCREMENTAL MODEL

In short this model divides the project development into various builds, where every build grows and expands the one before it after it has been validated. Each build is meant to be a manageable module, where a series of requirements are picked, developed, tested and validated before moving on to the next module. This model was picked because both framework and game could be well defined as a whole, so they could be broken down and built incrementally. Another driving factor was the need to get a working prototype of the framework early on so that the game could start to interact with it.

UNIFIED MODELING LANGUAGE

The Unified Modeling Language (UML), born in 1994 to the hands of Grady Booch, James Rumbaugh and Ivar Jacobson, is a modeling language used for general purposes in software engineering. Through UML one can, in a standard way, provide a way to visualize the intended system. In this thesis UML's structure diagrams are used to define the architecture design of both Bicomm and the game.

REQUIREMENTS THROUGH USE CASES

Bicomm's and the game's requirements were obtained through The Process Group's modifications and variations of "In Search of Excellent Requirements" by Karl E. Wiegers [18]. The Use Case Approach is task-centric, revealing requirements by explaining actions that need to be completed.

2.2 FRAMEWORK

REQUIREMENTS

Only the principal use cases will be exposed in this section through the Use Case Approach. To complement each use case, a rough drawn interface was carried out with the tool Pencil Project[7].

Sign up

Name	Sign up
Actor	User.
Description	The User uses one of the available invitations to become a Registered User.
Trigger	User selects Sign up action.
Preconditions	User is not a Registered User.
Post conditions	User becomes a Registered User.
Priority	High.

Normal Flow	
1. Select Sign up action.	2. System verifies invitation. 3. Invitation is valid, user becomes Registered User and Sign in action is called.

Alternate Flow	
	3.a Invitation is invalid, error message is prompted to the user. 3.b Invitation is already in use, error message is prompted to the user.

[7] "Pencil Project", pencil, http://pencil.evolus.vn/, (June 3, 201)

Derived Functional Requirements:

- Verify invitation
- Register User
- Invalidate invitation after successful sign up

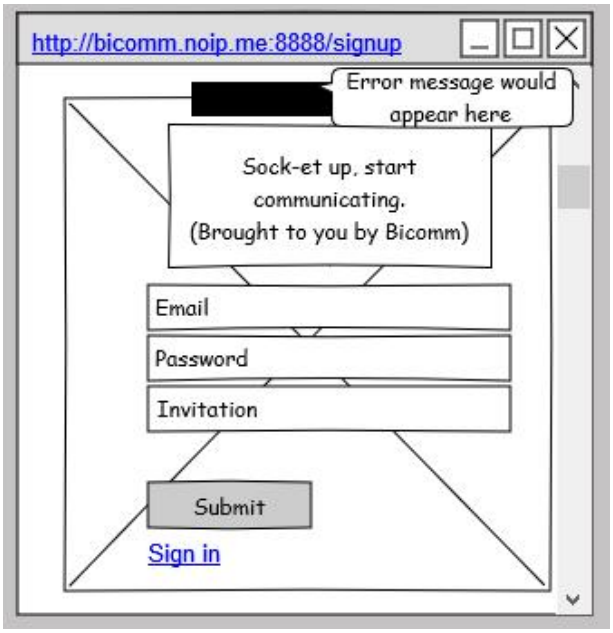


Illustration 6. Top part of Sign up page mock up.

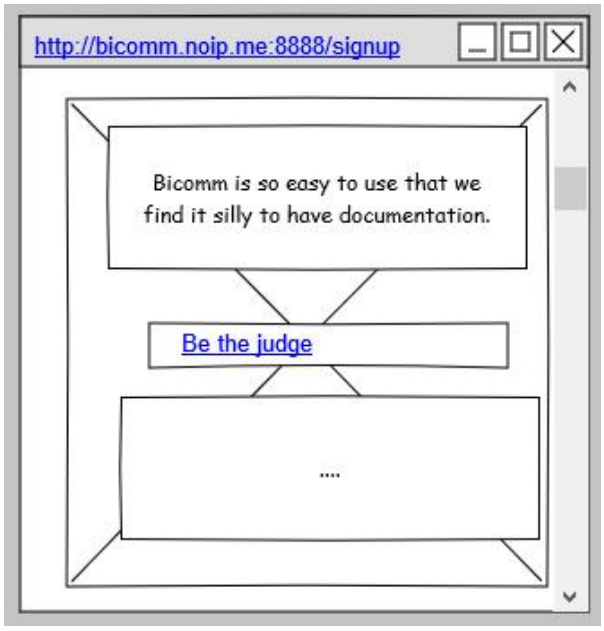


Illustration 7. Bottom part of Sign up page mock up.

Sign in

Name	Sign in
Actor	Registered User.
Description	The Registered User logs into the system.
Trigger	User selects Sign in action.
Preconditions	User is a Registered User.
Post conditions	User is signed into the system (has a cookie in browser that allows them access to URLs under /auth_api).
Priority	High.

Normal Flow	
1. Select Sign in action.	2. System verifies Registered User. 3. Registered User is valid, a cookie is placed in Registered User's browser, establishing session. 4. Registered User is redirected to Dashboard.

Alternate Flow	
	3. Registered User details are not valid, an error message is shown.

Derived Functional Requirements:

- Verify Registered User
- Place cookie in Registered User's browser

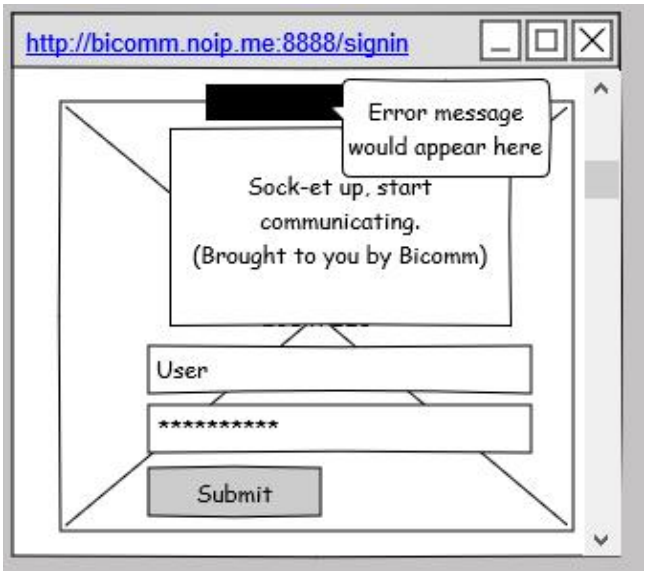


Illustration 8. Sign in mock up.

Download configuration file and library

Name	Download configuration file and library
Actor	Registered User.
Description	The user downloads configuration file and library from dashboard.
Trigger	Registered User selects Download action.
Preconditions	Registered User has logged in within the system.
Post conditions	Configuration file and library has been downloaded.
Priority	High.

Normal Flow	
1. Select Download action.	2. System serves configuration file and library in a .zip file.

Derived Functional Requirements:

- Identify User
- Verify User Session

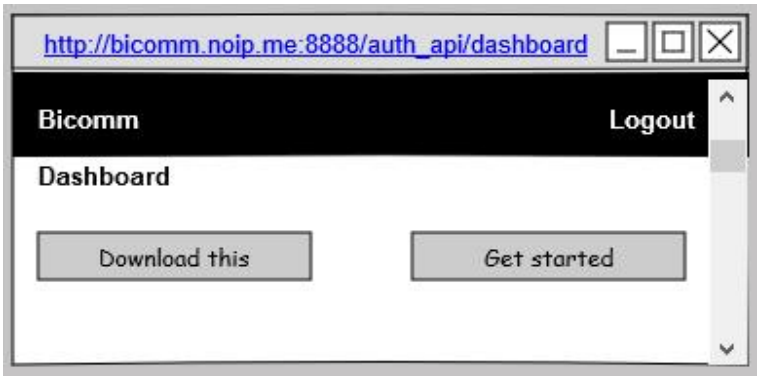


Illustration 9. Dashboard mock up.

DESIGN

This section will briefly introduce Node.js and then move on with the framework's architecture, which will be presented through UML's structure diagrams supported with explicatory paragraphs.

Node.js

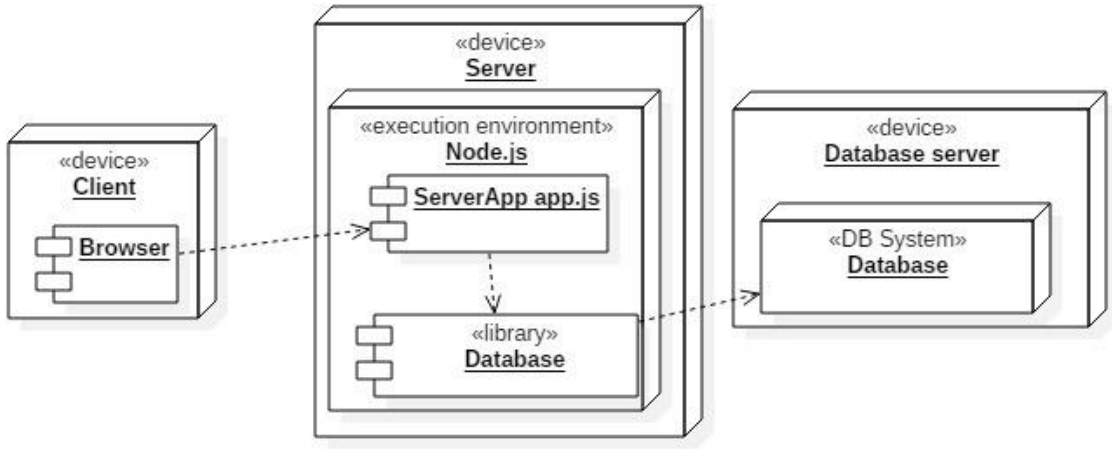
Node.js was first released in 2009 thanks to its creator, Ryan Dahl. As per the documentation[8]:

> "Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world."

Node.js gave the developers of the world a way to create web applications that can establish real-time, two-way connections. An interesting detail to mention about Node.js is the fact that it is single threaded, using non-blocking I/O calls, which allow it to support tens of thousands of concurrent connections without incurring the cost of thread context switching. Its goal is to allow highly concurrent applications. This is achieved by having any I/O performance use a callback. A direct negative of this set up is that Node.js does not allow vertical scaling through CPU core number increase without using additional modules.

[8] Nodejs.org Node.js, https://nodejs.org/en, (June 5, 2016)

Node.js on its own does not provide a web server out of the box, one has to add required modules, create the server and accept and return responses. All of this is done in the main JavaScript file that is run with the command `$node app.js`.



UML Diagram 1. Sample deployment diagram of a Node.js system.

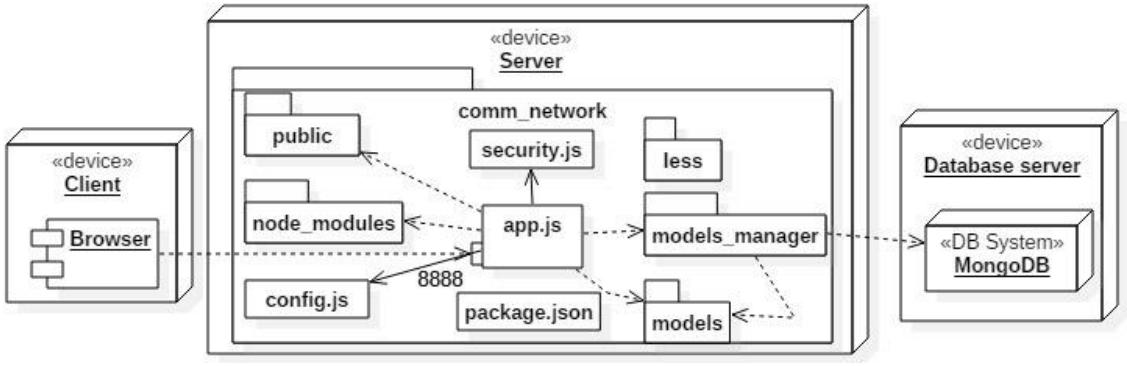
Developers coming from other server side scripting languages might find it quite different, and a bit daunting, to have to program, or configure npm modules, all those things that a server like Apache or Nginx does automatically. But that is implied whenever one uses new technology, it is just a matter of learning how to use it.

To complete Bicomm the subsequent technologies were used:

- Node.js
- Node.js modules
- MongoDB
- Bootstrap framework 3.3.1
- Less
- AES 256 encryption - CTR (Counter) mode
- jQuery 1.11.2
- HTML
- CSS
- JavaScript

Framework architecture

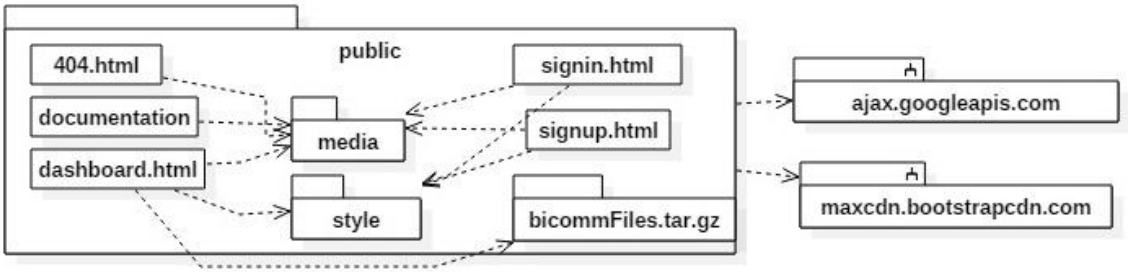
Bicomm's architecture is resumed as follows.



UML Diagram 2. Bicomm's architecture.

The main file of the framework is *app.js*, it is in charge of creating the server, setting request and response manipulation, and routing the real-time bidirectional communication. The created server, as shown in the diagram, is set to listen on port 8888. The database used is document-oriented MongoDB. Each package in the diagram will be introduced briefly to further understand the components of Bicomm.

public



UML Diagram 3. public folder.

The *public* folder contains all the HTML that is served in response to client requests by the server created in *app.js* . When clients access the dashboard the downloaded content is the *bicommFiles.tar.gz* , which contains the needed JavaScript library and configuration file: *socket.io.js* and *socket.io-config.js* .

An interesting aspect to mention is the fact that one can program route shortcuts in the created server in *app.js* . For example, image files are in the route *media/assets/* but if one inspects the source code of the HTML pages, they will see that image routes are

images/ , which is the shortcut that was created. It is a quick, easy way of hiding the server's real folder structure to prying eyes.

The responsive style of the site was carried out thanks to the Bootstrap framework's 12 column grid system. As can be seen from the diagram, Bootstrap is accessed through a CDN (Content Delivery Network) and its style customization is achieved by manipulating the framework's Less files and adding the resulting CSS (Cascading Style Sheet) file last. Less extends the CSS language by adding features that allow variables, mixins and functions, as well as other techniques that are not available with plain CSS. Less is a CSS pre-processor that manages to make CSS theme-able and extendable. Less sheets will not work in browsers as substitutes for CSS sheets, it is necessary to compile Less sheets to obtain the resulting CSS rules that all browsers understand. There are two options for this, either compile Less files once and serve the CSS result, or serve Less files directly along with the file *less.js* , in charge of compiling in real-time. The latter is great for development but a poor choice for production when performance and reliability are important. The CSS files in the *style* folder are the result of compiling the appropriate LESS files in the *comm_network* folder. LESS files' compilation and compression is carried out with the modules *less* and *less-plugin-clean-css*, which can be called from the command line.

node_modules

Node.js modules are managed by the pre-installed package manager that is npm. From the npm registry one can install any number of modules through the command line. The modules found in npm range from simple helping libraries to complicated task runners. As per npm site's definition, npm is:

> "… a way to reuse code from other developers, and also a way to share your code with them, and it makes it easy to manage the different versions of code."[9]

A detail to be remarked upon is the fact that whenever a module is installed one can mark it as a project dependency by adding the option *--save,* which instructs npm to include the package inside of the dependencies section of the project's *package.json* file. Additionally, one can go further and use `--save-dev` to save packages that are only needed during development, like the *morgan* module (an HTTP request logger middleware). If the project needs to be moved, only the project's core files are needed thanks to npm's `install` command, which will install all the project's dependencies written in its *package.json* file. The node modules used in this project were:

[9] "What is npm?" npm, https://docs.npmjs.com/getting-started/what-is-npm, (June 6, 2016)

- *body-parser*

  This middleware allows client's form input to be read by storing it as a JavaScript object that is accessible through `req.body` where `req` represents the request that the server receives.

- *express*

  This module allows rapid web development by providing a robust set of features that give the developer the capability to set up middleware that can respond to HTTP requests, define routing tables that are used to carry out different actions depending on the HTTP Method and URL, and allow dynamic HTML page rendering depending on the arguments passed to the templates.

- *forever*

  This module is used to keep the Node.js server developed running perpetually. A Node.js server can be started through the `node` command in the current shell, but the process will stop running when the shell is closed, crashes or it is forced to exit through `Ctrl-C`. With forever the child process (the Node.js web server) will run as a daemon continuously, and if the server crashes for any reason it will also restart it automatically.

- *jsonwebtoken*

  The module is used to create and verify JSON (JavaScript Object Notation) Web Tokens.

- *less and less-plugin-clean-css*

  These modules are used to compile and compress LESS files.

- *mongoose*

  The mongoose module is a MongoDB object modeling tool that allows model definition through the Schema interface.

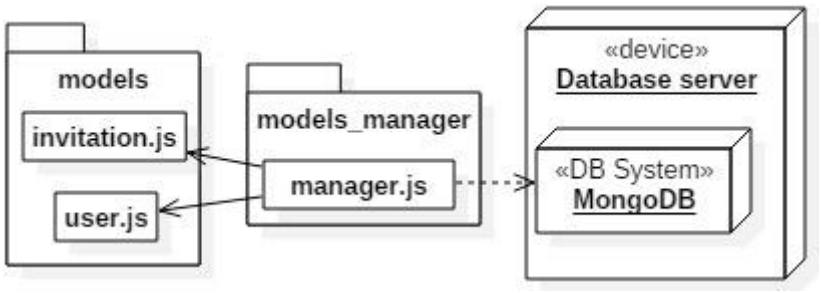
- *morgan*

  This module was only used during development to help log HTTP requests directly to the shell.

- *socket.io*

  This module is key to the framework's service. The socket.io module allows WebSocket connections to provide bidirectional communication in real-time.

models_manager and models



UML Diagram 4. DAO (Data Access Object) design pattern applied to a document-oriented database.

The diagram shown encapsulates how MongoDB is accessed by the system. The DAO (Data Access Object) design pattern was taken as reference and applied to the *manager.js* file. A DAO is an object or an interface that provides access to an underlying database or a persistence mechanism, ultimately isolating the Single Responsibility Principle, which states:

> "A class should have only one reason to change."[10]

With the help of the mongoose module the documents in the database were represented through the Schema interface. It is through *manager.js* that the server can access the database without worrying about specific code interaction. Because all documents are accessed the same way, there is no need to create model-specific classes that access MongoDB, it can all be done with just the *manager.js* file, which is made available to the server as a local module through `module.exports`. The *manager.js* uses the models (*invitation.js* and *user.js)* to know what document in the database it is supposed to query. All that needs to be done before querying *manager.js* is to set the appropriate model.

security.js

It is necessary for the framework to be able to authenticate signed in users in order to allow them access to the dashboard. The traditional way of authenticating is through

[10] "SRP: The Single Responsibility Principle" drive.google.com, https://drive.google.com/file/d/0ByOwmqah_nuGNHEtcU5OekdDMkk/view?pli=1, (June 6, 2016)

server based authentication, where the server stores the user's logged in information locally (either memory or disk). This implies a series problems:

- For every authenticated user the server creates a record somewhere locally. The more authenticated users there are the more overhead the server experiences.
- If scalability is part of the plan then sessions stored in memory will present a problem when the cloud provider starts to replicate servers to handle increased application load.
- CORS (cross-origin resource sharing) could become a problem, such as forbidden requests, when the application wants to share the data across multiple devices.

Taking into account the listed problems the token based authentication was used in Bicomm. This authentication method is stateless, information about the user is not stored in the server nor in a session. This instantly solves the scalability problem, now the application can scale and add as many machines as it needs to without having to worry about where a user is logged in (also referred to as session affinity). The usual flow goes as follows:

1. User requests login with username and password
2. Server validates credentials
3. Credentials are valid so the server provides a signed token
4. Client stores signed token and sends it along with every request
5. Server verifies token and responds with the appropriate data

A big security pro about using token based authentication is preventing CSRF (cross-site request forgery) attacks. This is because no cookie is sent with each request, the token is sent as an HTTP header. More often than not though, the token is stored in a cookie, but said cookie would only be a storage mechanism, not an authentication one. The industry standard in token based authentication is JWT (JSON Web Tokens).[11]

JWT work across different programming languages and are self-contained, which is why they can be transmitted through an HTTP header or through the URL.

[11] "JSON Web Token (JWT)" Internet Engineering Task Force (IETF), https://tools.ietf.org/html/rfc7519, (June 6, 2016)

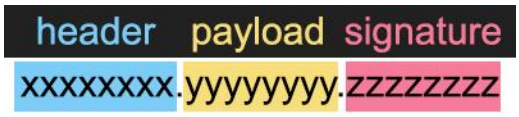


Illustration 10. JWT anatomy.

A JWT is made up of three parts separated by a dot, and they are the header, the payload and the signature. The header itself is composed of two parts, the declaring type (which is JWT) and the hashing algorithm used for the signature component, the resulting JSON object is encoded with Base64, and the result is the header. The payload will carry the information that the server wants to transmit plus information about the token itself, also as a JSON object and Base64 encoded. The third and last part of a JWT is the signature, which includes the header, the payload and a secret, all encrypted with the hash algorithm specified in the header part of the JWT.

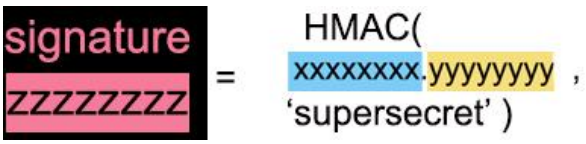


Illustration 11. JWT signature component.

The module *jsonwebtoken* was used to implement JWT authentication in Bicomm. Thanks to the module the following was achieved:

- Existence of protected and unprotected routes (everything under */auth_api/* is protected and requires a JWT) thanks to JWT validation.
- User authentication through a form, passing the username and password and receiving a JWT in return. This token is stored in a cookie and passed along whenever an */auth_api/* domain request is carried out by the client.

In the JWT Claims (the payload part of the JWT), the user's name and plan is registered to avoid having the server look that information up every time the user carries out a request. Saving user information as JWT Claims is a common practice, but one must be aware of the fact that a JWT is easily decodable because its encoding is Base64, in other words, it merely represents data in an ASCII (American Standard Code for Information Interchange) string format. This is best described with an

example. Using the tool Wireshark[12] the sign in HTTP conversation was monitored to extract the JWT.

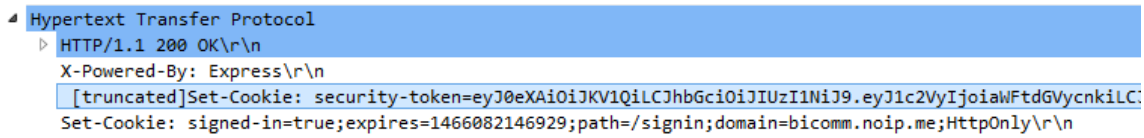


Illustration 12. HTTP monitoring.

The secret token the server returned is the following:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoiaWFtdGVycnkiLCJwbGFuIjoicmVnIiwiaWF0IjoxNDY1NjUwMTQ2LCJleHAiOjE4OTc2NTAxNDZ9.SmfNfaq0G50MgAmhDYY9SMdED7H5hNdopXFeMaUnsno

When introduced in jwt.io's debugger[13] (making sure the algorithm is set to HS256 and the signature's secret is *mastertecwebnegdigitaltfmulpgc*), the following decoded information is obtained:

Illustration 13. JWT decoded.

As can be seen from the image, the JWT Claims are easily obtained. The signature's secret is only necessary to validate the JWT, without it one can still decode the JWT's payload, which contains the claims. To add a layer of security towards the information in the claims the entire JWT is encrypted with the help of the file *security.js*, which uses

[12] "WireShark" https://www.wireshark.org/ (June 6, 2016)
[13] "Debugger" JWT, https://jwt.io/, (June 6, 2016)

the *crypto* module, employing the Advanced Encryption Standard encryption algorithm with 256 bits key length in CTR mode. With this in place the JWT looks as follows:

8b2596226a422bc97d15f7e8dd405ed965b7377d4713b51a65f5f4d7838e4fb13f6885d2176fd0c6014395e42db265ed1da94aee535705a111c216cd75f88834d57bf27ad419d823f0d44edc98743d9ce3b3b5dbe81085447475baacf3b3c703988d17f4386f0f15f6dc34a180de883de9cd421ee11b45fd2fd388b355727101cfc94bc4136660d9186902c0aa66e3e05a65379412305fb9526be9f8f5bab0335b849f78e25fe65bdf

The downside of this is that with every request the JWT will have to be decrypted, not to mention there still is a big flow in security with this design since there is a way for someone to hijack the JWT stored in the cookie. With Wireshark one can still sniff out the packets communicated in the network and attain the new encrypted data stored in a cookie. Once that is obtained, all it takes is to add that same cookie to one's browser through a JavaScript code as simple as:

```
document.cookie="security-
token=8b2596226a422bc97d15f7e8dd405ed965b7377d4713b51a65f5f4d783
8e4fb13f6885d2176fd0c6014395e42db265ed1da94aee535705a111c216cd75
f88834d57bf27ad419d823f0d44edc98743d9ce3b3b5dbe81085447475baacf3
b3c70fd38d07d23b6f0f15f6dc34a180de883de9cd421ee11b7bdb2cd1888555
721540c1d55af4084418c8147065d4946e918463604ea31c3137a6631aa3f9fb
93f004699ea26d9f6deb6384;path=/auth_api";
```

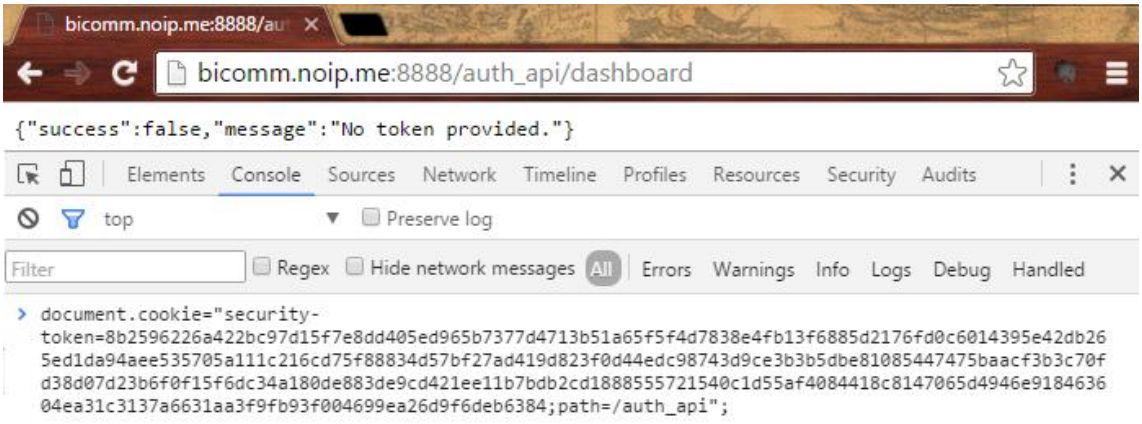
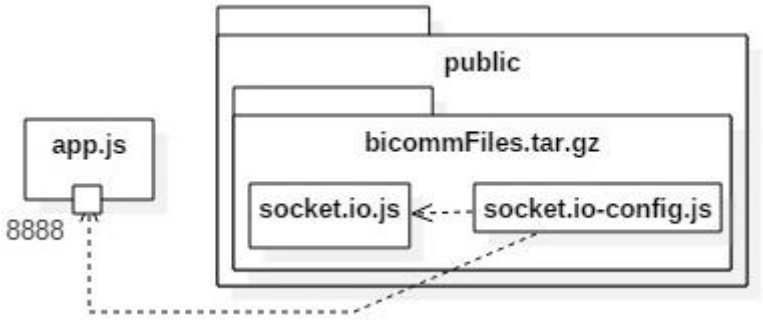


Illustration 14. Hijacking JWT inside cookie.

Once the command is executed and the page refreshed, access to all pages under */auth_api/* is possible. The one way to fix these problems is to use a safe connection through HTTPS, where all the requests are encrypted, including the headers and

cookies. That is why it is extremely important to use HTTPS whenever session tokens or cookies with sensitive information are shared.

app.js and bicommFiles.tar.gz



UML Diagram 5. Server and configuration files.

This is the main file where the core code of the framework is. Both the code for the HTTP server and the bidirectional communication is here. The compressed directory contains the client side script from the module *socket.io* and the necessary configuration file that the client must set up accordingly to start using the service.

For the bidirectional communication service to work two options are valid[14]. On one hand, the socket's connection can be forced to use only WebSockets, which are inherently cross domain. On the other hand, CORS must be enabled server side by having the following headers: `Access-Control-Allow-Origin=*, and Access-Control-Allow-Headers= X-Requested-With`. The latter option was used in Bicomm because it allows the Socket.io library to handle what best connection method is suitable for the client, starting off with long polling and upgrading gradually to WebSockets if the client supports them.

The full duplex conversation is managed in the following steps:

1. Client connects (either long polling or WebSockets) with server and passes a room id.
2. Server adds connected client to a room labeled as the room id passed.
3. Whenever client emits a 'controlAction' message to the server, the server will broadcast the message to everyone that belongs to that socket's room.

[14] With the knowledge acquired the student answered the following question in stackoverflow.com under the username JMR. "Cross-domain connection in Socket.IO" stackoverflow.com, http://stackoverflow.com/questions/8970880/cross-domain-connection-in-socket-io/33912510#33912510, (June 6, 2016)

4. Any client in the same room as the one who emitted the 'controlAction' message will receive the message. The client who emitted it will not receive it.

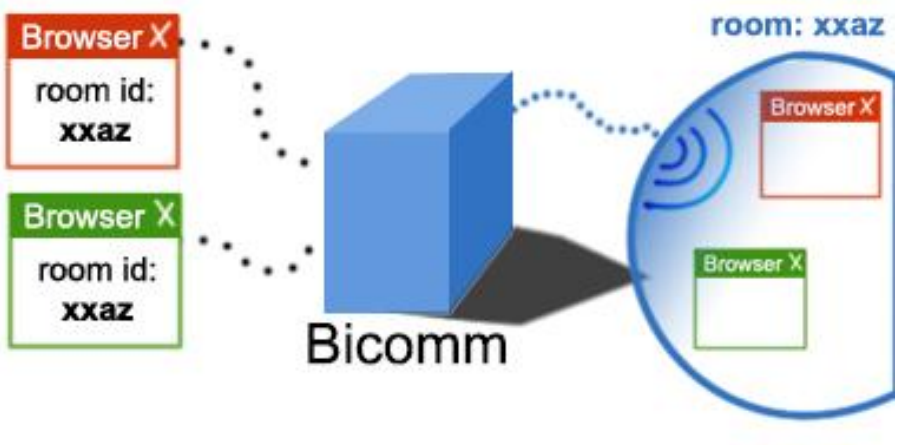
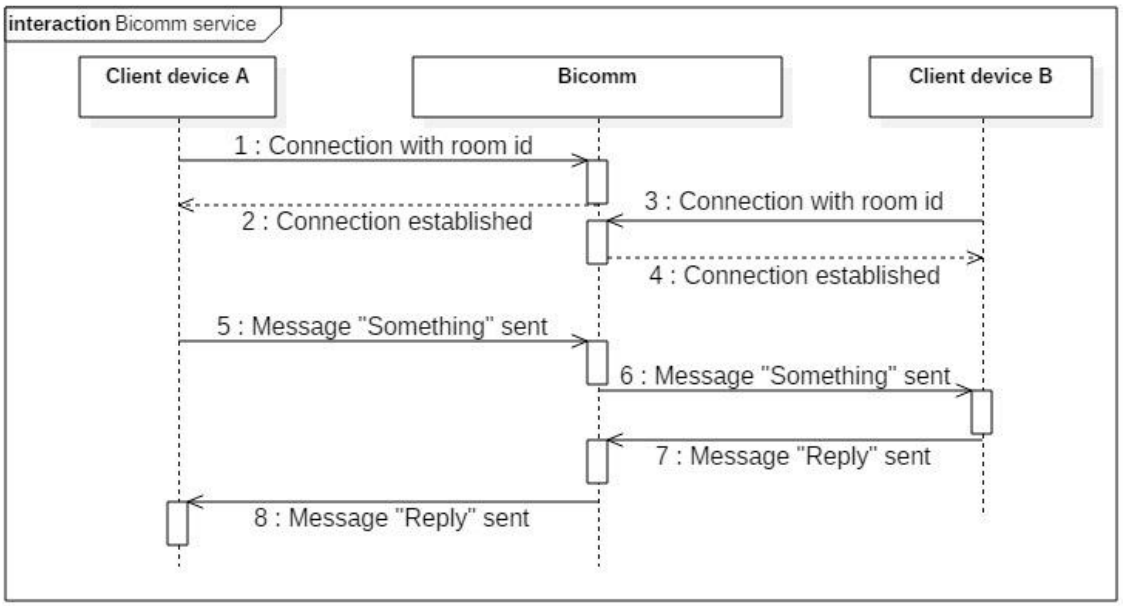


Illustration 15. Full-duplex scheme.



UML Diagram 6. Interaction diagram of Bicomm's service.

DOCUMENTATION

This section presents the "get started" documentation in Bicomm's site.[15] The briefness of said documentation is due to how simple it is to set up. Following are the steps one must carry out to be well on their way to using Bicomm's service.

[15] "Documentation" Bicomm, http://bicomm.noip.me:8888/documentation, (June 8, 2016)

- Download the library and configuration file from your dashboard. The downloaded files are required by your clients.
- A room id must be set up for every group of clients you want communicating. You can't call `IO.init()` without setting the room id first. If you need different groups of clients communicating, be sure to set appropriate room ids. Clients with the same room id will communicate with each other.

```
IO.room_id = room_id;
IO.init();
```

- To set the action to be performed when the client receives a message:

```
IO.onReceiveAction = function(msg) {
   /*The message is contained in the msg variable, this is
the full duplex communication (receiving).*/
};
```

- To send a message to fellow clients with the same room id:

```
IO.sendAction("message to be sent as a string");
```

Practical Example

ClientA and ClientB are to communicate with each other. While ClientC and ClientD are to communicate with each other and not with ClientA nor ClientB.

ClientA and ClientB's code:

```
IO.room_id = 'secretforchatroomONE';
IO.init();
IO.onReceiveAction = function(msg) {
   console.log(msg);
};
```

Then ClientA does:

```
IO.sendAction("I, ClientA, won't see this message.");
```

ClientB would see in their JavaScript console the message sent.

ClientC and ClientD's code:

```
IO.room_id = 'secretforchatroomTWO';
IO.init();
IO.onReceiveAction = function(msg) {
   console.log(msg);
};
```

Then ClientC does:

```
IO.sendAction("I, ClientC, won't see this message.");
```

ClientD would see in their JavaScript console the message sent.

2.3 GAME

REQUIREMENTS

Just like it was done for the framework, only the principal use cases will be exposed in this section through the Use Case Approach. There is a rough drawn interface complementing each use case.

Choose Controls

Name	Choose Controls
Actor	Player.
Description	The Player chooses where they want the controls of the game to be.
Trigger	Player enters the game site.
Preconditions	None.
Post conditions	Player is taken to the game's main menu.
Priority	High.

Normal Flow	
1. Player selects PC .	2. System redirects player to game.

Alternate Flow	
1. Player selects Phone. 3. Player selects either One or Two Players.	2. System makes Choose Player Mode action visible. 4. System redirects Player to game with the appropriate controls' URL.

Derived Functional Requirements:

- Make it possible for user to pick where they want the game controls to be
- Generate appropriate URL(s) when player chooses controls on phone

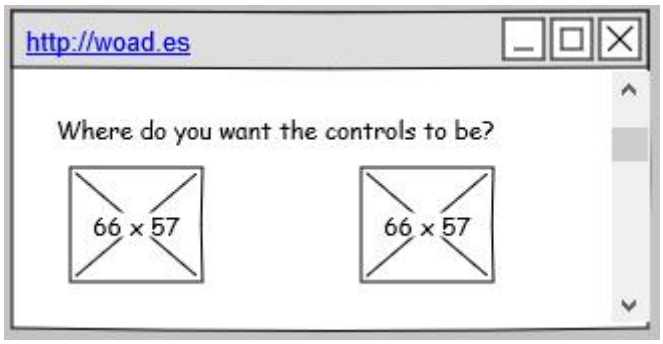


Illustration 16. Normal flow Choose Controls mockup.

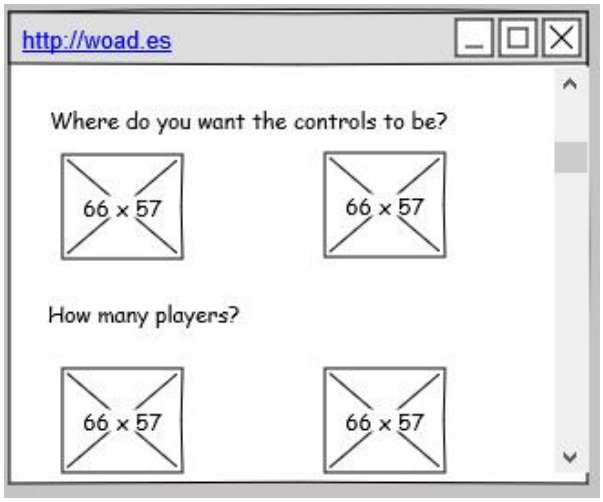


Illustration 17. Alternate flow Choose Controls mockup.

Illustration 18. Post condition mockup of alternate flow Choose Controls.

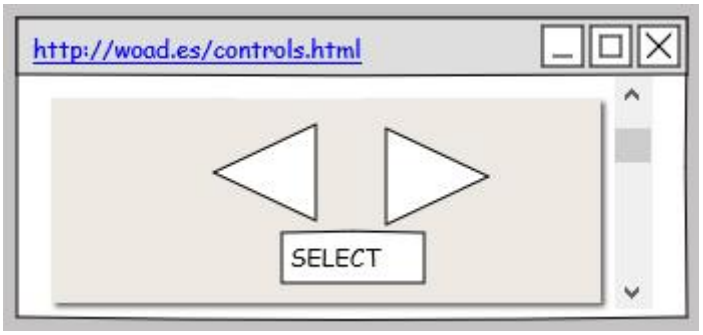


Illustration 19. Controls mock up.

Access Options' Menu

Name	Access Options' Menu
Actor	Player.
Description	The Player accesses the game's options menu.
Trigger	Player selects Access Options' Menu action.
Preconditions	Player has chosen where they want the controls to be.
Post conditions	Player is presented with the options' menu.
Priority	Medium.

Normal Flow	
1. Player selects Options' Menu . 3. Player changes options available.	2. Game presents options menu. 4. Game saves changes.

Derived Functional Requirements:

- Player can change game options such as music.

Choose Level

Name	Choose Level
Actor	Player.
Description	The Player chooses a level to play.
Trigger	Player selects Choose Level action.
Preconditions	Player has chosen where they want the controls to be.
Post conditions	Player is playing the level.
Priority	High.

Normal Flow	
1. Player selects a level.	2. Level is unlocked. 3. Game loads the level and the player plays.

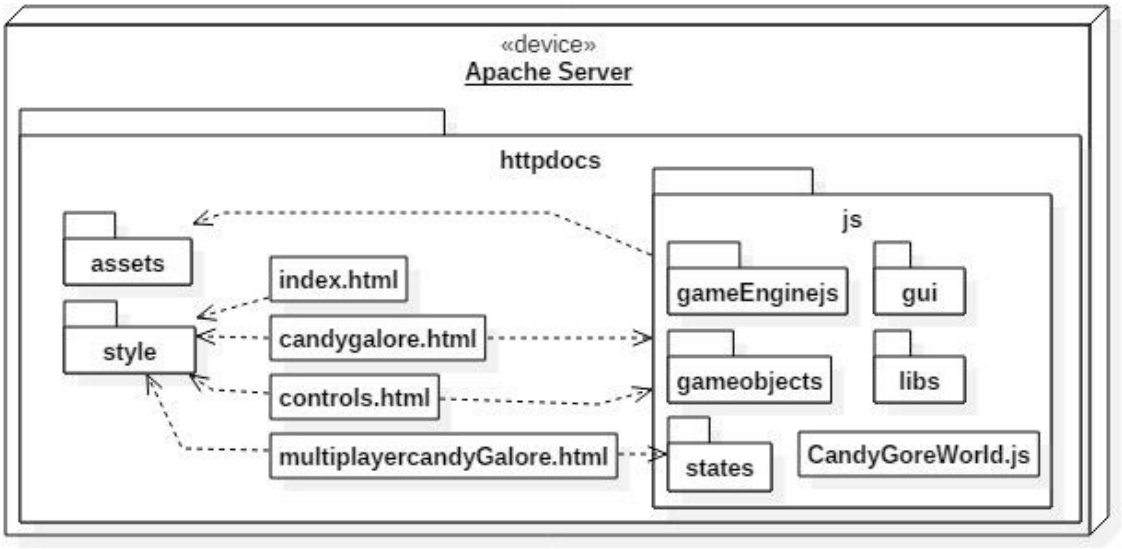
Alternate Flow	
	2. Level is locked. 3. Game does not load the level.

Derived Functional Requirements:

- Player can choose what level to play.

DESIGN

The game's architecture will be described in this section through UML's structure diagrams supported with explicatory paragraphs.

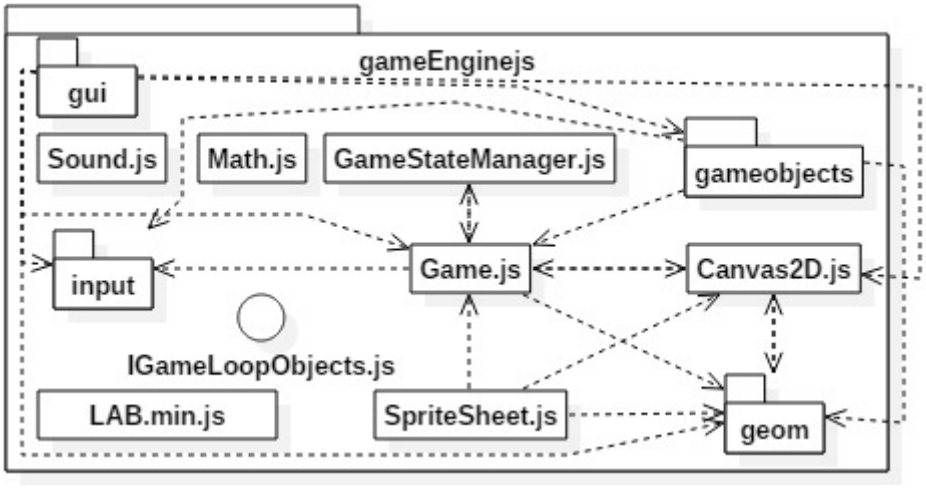


UML Diagram 7. Game architecture.

This is the game's general architecture. The server where the game is hosted does not have any CMS (Content Management System) installed. The game is served simply through an HTML file and interacts thanks to JavaScript. In the following sections each part will be explained with more detail.

gameEnginejs

The *gameEnginejs* holds the game engine that was developed while implementing the game. It consists of a series of classes that take care of various responsibilities such as geometry and user input. The rest of the game is built upon this base, and it can easily be reused for another game that might require its need.



UML Diagram 8. gameEnginejs contents.

The contents of the game engine are made available through a set namespace. A namespace is a technique often used in JavaScript to avoid collisions with other objects or variables in the global namespace. The technique is also used to organize modules (blocks of functionality) in manageable groups, like a game engine. The only set back is that JavaScript does not provide a built-in support for namespaces as other languages do. Nonetheless, it does provide objects and closures, which are used to achieve the namespace design through a JavaScript mechanism that allows the developer to define and call a function at the same time. Eric Miraglia[16], engineering manager for the YUI project at Yahoo, first talked about this technique as a "JavaScript Module Pattern", back in 2007.

```
"use strict";

var gameEnginejs = (function (gameEnginejs){
      ... Define module
      gameEnginejs.myModule = MyModule;
      return gameEnginejs;
})( gameEnginejs || {} );
```

What is essentially been done here is creating an anonymous function and executing it immediately, that way all the code inside the function will live in a closure, providing privacy and state throughout the lifetime of the application. By returning the specified module created within the closure, the basic Module Pattern is completed. Taking the pattern to an advanced level, loose augmentation allows JavaScript applications to

[16] "A JavaScript Module Pattern" YUI, http://yuiblog.com/blog/2007/06/12/module-pattern/, (June 8 2016)

load scripts asynchronously, where each module can reside in its own file. Ultimately what the expression `gameEnginejs || {}` does is pass the variable `gameEnginejs` if it is defined or an empty object literal to start constructing upon. Thanks to this, it will not matter the order in which the modules are added to the namespace, the first time it happens with an empty object literal, and afterwards it will be the `gameEnginejs` variable that has been defined.

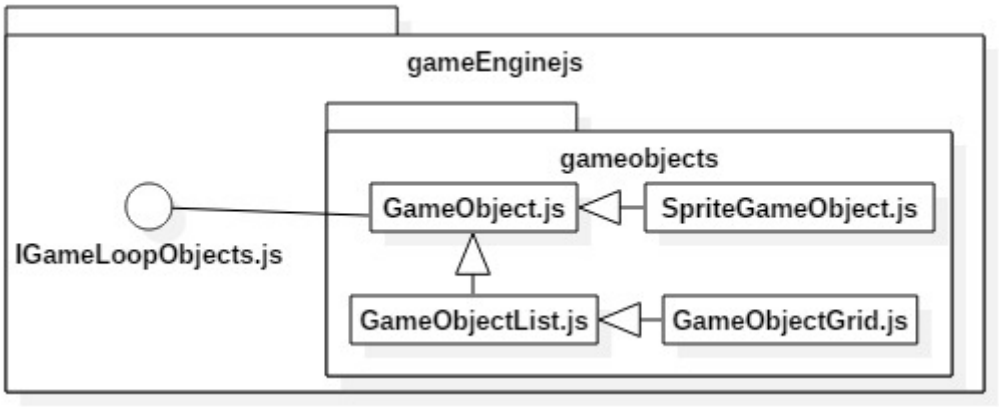
A brief description of each component in the game engine follows.

IGameLoopObjects.js

This file is meant to define an interface that other classes will implement in order to be considered as game loop objects. The methods defined here are needed for the game loop requirements, such as *handleInput, update, draw* and *reset*.

gameobjects

The *gameobjects* folder inside of *gameEnginejs* contains all the base classes that a game might use to further develop other classes.



UML Diagram 9. gameobjects contents.

The *GameObject* class is the base class of the game engine, it contains basic properties (like world position) and functions (like the ones set in the *IGameLoopObject* interface). The world position property of the *GameObject* class calculates the object's absolute x and y coordinates in the game world. It also contains the local position, which is the object's position in respect to its parent's position. The world position is not stored as a fixed number, but it is rather calculated upon called, by adding the object's local position to its parent's world position. If the object has no parent, then its local position is the same as its world position. All other classes in the game engine extend from this base class. The *GameObjectList* class is meant to represent a list of

GameObjects , with all the functionalities of a list (add, sort, remove, at, find, clear...etc). It also overrides some of the *GameObject's* functions to better suit its purpose. The *GameObjectGrid* class extends from the *GameObjectList* and its goal is to represent a matrix of columns and rows. Even though the *GameObjectGrid* is a *GameObjectList* it manages to be a grid through the following math equations to determine row and column positions in a single dimensional list:

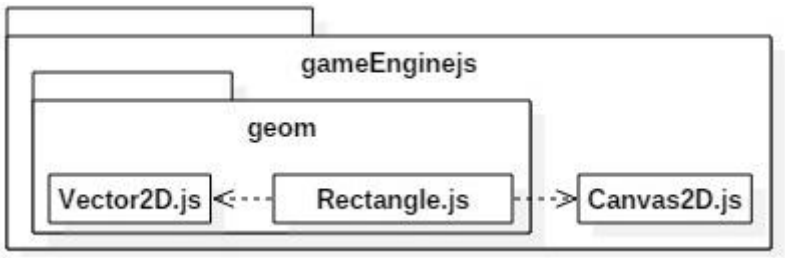
```
var row = Math.floor(this._gameObjects.length / this._columns);
var col = this._gameObjects.length % this._columns;
```

Last, the *SpriteGameObject* handles any *GameObject* that has a sprite of its own to draw, or is represented by a sprite (an image).

Math.js and Sound.js

As the name of the *Sound.js* file suggests, this module is used to add sound to the game, carrying out basic functions such as play or having properties such as volume. The *Math.js* file is not an exported module to the *gameEnginejs* namespace, but it's rather used to add functionality to JavaScript's Math built-in object.

geom



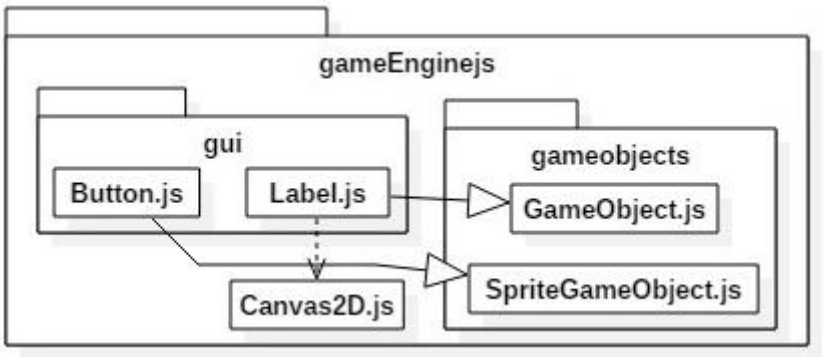
UML Diagram 10. geom contents.

The *geom* folder contains exported modules that are meant to aid in geometrical needs. The *Vector2D* module represents a two dimensional vector, and provides several functions to manipulate such a vector (addition, subtraction, multiplication...etc). The *Rectangle* module, as its name states, represents a rectangle with the help of the *Vector2D* module. Being able to represent a rectangle in an HTML5 Canvas is absolutely basic in terms of pin pointing where an object is and what area it delimits within the Canvas' point grid. It helps to define portions of the screen, letting the application know if a certain area is being touched, for example.

SpriteSheet.js

If a game uses many sprites it becomes inefficient to load them one by one, especially if the data is to be transmitted over the internet. It is not the same to have a single request and download 30 sprites within a single sheet, than to have 30 individual requests for each sprite. This technique does not apply only to game development, it is often used to improve a site's speed, since CSS allows background positioning of images, letting it use sprite sheets of icons. As it would be implied, the *SpriteSheet* module allows the game to use sprite sheets instead of single sprites. The dimensions of the sprite sheet (how many columns and rows) does not have to be set dynamically, instead it is the name of the sprite sheet that must contain its proportions: spr_name@COLSxROWS.png.

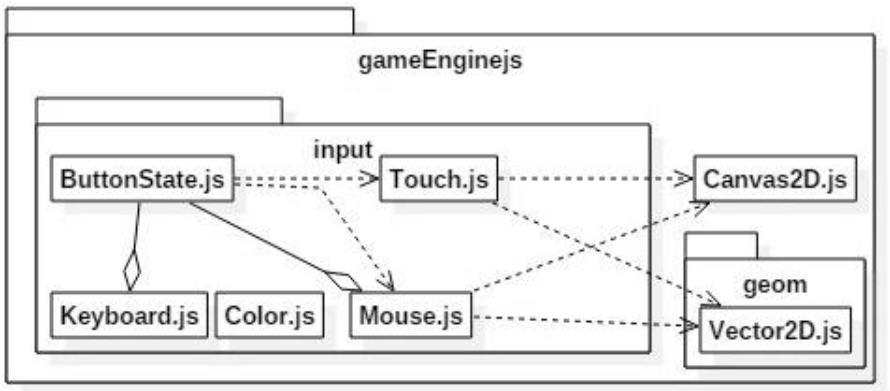
gui



UML Diagram 11. gameEnginejs/gui contents.

This folder contains commonly used GUI (Graphic User Interface) elements, such as a Label or a Button. The modules are meant to be used on an HTML5 Canvas.

input



UML Diagram 12. input contents.

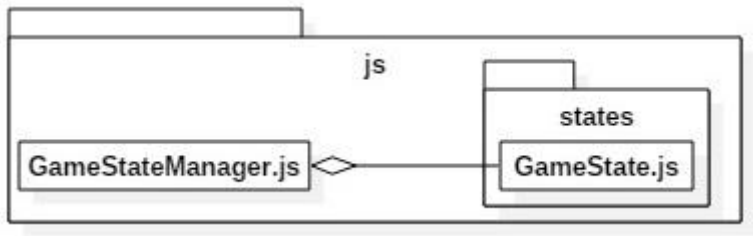
The *input* folder handles all the basic ways a user could input information towards the game: mouse, touch (touch devices), and keyboard. The *Color.js* file is merely a way of accessing hexadecimal colors through their human readable name. A big difference when developing with an HTML5 Canvas is that detection of user input through common events such as *mousedown* or *keydown* are no longer enough. It is also necessary to determine if the position where the event occurred forms part of a *GameObject* such as a button. The game objects that tend to be clicked are those that are drawn on the Canvas, those would be the *SpriteGameObjects*. Hence why that class has a very useful property called *boundingBox,* which calculates the rectangle it encloses within the Canvas. This property is then used to determine if the user input event is destined for said *SpriteGameObject.*

Another design pattern is introduced in this folder, the Singleton Design Pattern[17]. Summarized, the Singleton pattern is meant for classes that need only have one instance and provide a global point of access to its information, along with lazy initialization (initialize when it is called for the first time). The way to achieve this is by exporting the module through an instance of the class rather than the class definition.

```
var gameEnginejs = ( function(gameEnginejs) {
      function Mouse_Singleton() { ... }
      ...
      gameEnginejs.Mouse = new Mouse_Singleton();
      return gameEnginejs;
}) (gameEnginejs || {} );
```

The classes that use this pattern are: *Mouse.js, Keyboard.js* and *Touch.js*.

GameStateManager.js



UML Diagram 13. GameStateManager.js

[17] "Singleton Design Pattern" SourcMaking, https://sourcemaking.com/design_patterns/singleton, (June 8 2016)

This class is in charge of managing the game's states, which also implies switching between them when the need arises. This class also implements the Singleton Design Pattern.

Canvas2D.js

The *Canvas2D* singleton class represents the HTML5 canvas used in the game, offering various functionalities such as resize, clear, initialize, scale or drawImage.

Game.js and LAB.min.js

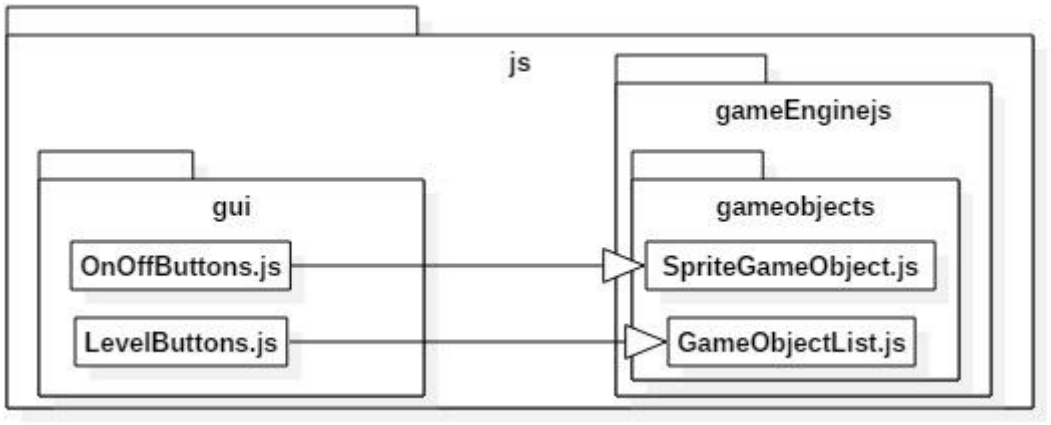
Last but not least are the *Game.js* and *LAB.min.js*. The traditional way of adding JavaScript files (sequentially with the <script> tag) will not work because JavaScript files are retrieved from a server, being unable to pin point the order in which they will be loaded. The browser will not be able to interpret the code if the files are loaded in an inappropriate order (because of the existing architecture, classes like *GameObject* need to be loaded before *GameObjectGrid*). If a file is loaded with code that points to another file that has not been loaded yet, the code will not work. To load JavaScript dynamically and in a predefined order the script-loading tool LABjs[18] is used, and that is the *LAB.min.js* file. The *Game.js* file contains generic game code to start up a game based on the developed game engine, some of its methods are: size, totalTime, loadAssets, start...etc. The idea is for another class to use these methods to get a specific game ready. An important inner property within this class is the *requestAnimationFrame*, in charge of telling the browser that the application needs to perform an animation and requests for it to call a specific function to update said animation before the next repaint. The property finds out if the function *requestAnimationFrame* is available in the current browser, if it is not it returns a function that will call another function (a callback) after 16.6 milliseconds have passed, in an attempt to simulate a basic *requestAnimationFrame*. *Game.js* also uses the Singleton pattern.

```
var requestAnimationFrame = (function () {
          return  window.requestAnimationFrame ||
                window.webkitRequestAnimationFrame ||
                window.mozRequestAnimationFrame ||
                window.oRequestAnimationFrame ||
                window.msRequestAnimationFrame ||
                function (callback) {
                      window.setTimeout(callback, 1000 / 60);
```

[18] "Loading And Blocking JavaScript" LABjs, http://labjs.com/ , (June 15 2016)

```
                    };
    })();
```

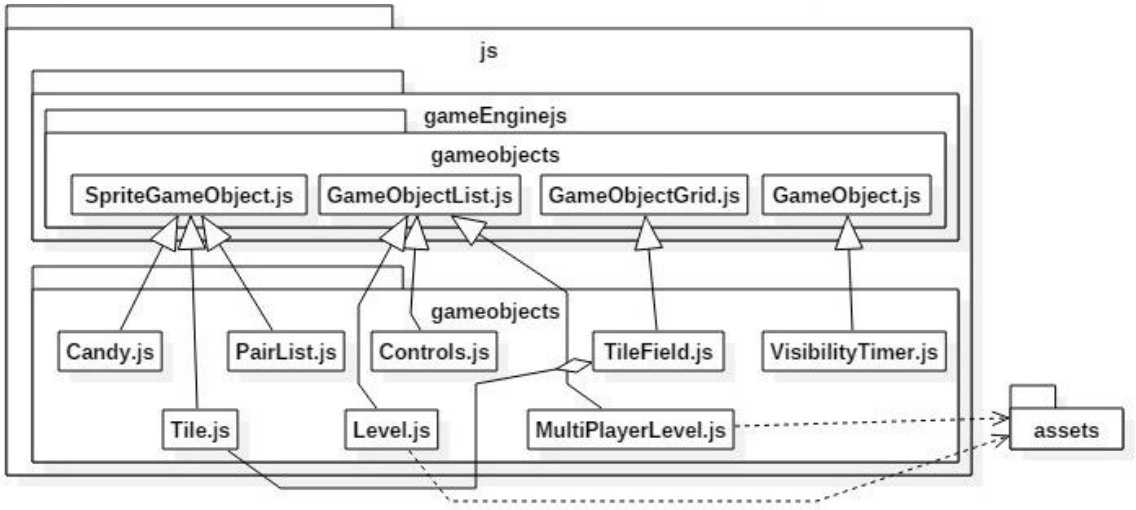
gui



UML Diagram 14. js/gui contents.

The *OnOffButton.js* represents a button that has two states: on or off. It manages to be a single *SpriteGameObject* because it uses a sprite sheet (one row, two columns) to represent both of its states. Meanwhile, the *LevelButton.js* represents the buttons the user clicks to play a level in the game. It has to handle three states: solved, unsolved or locked.

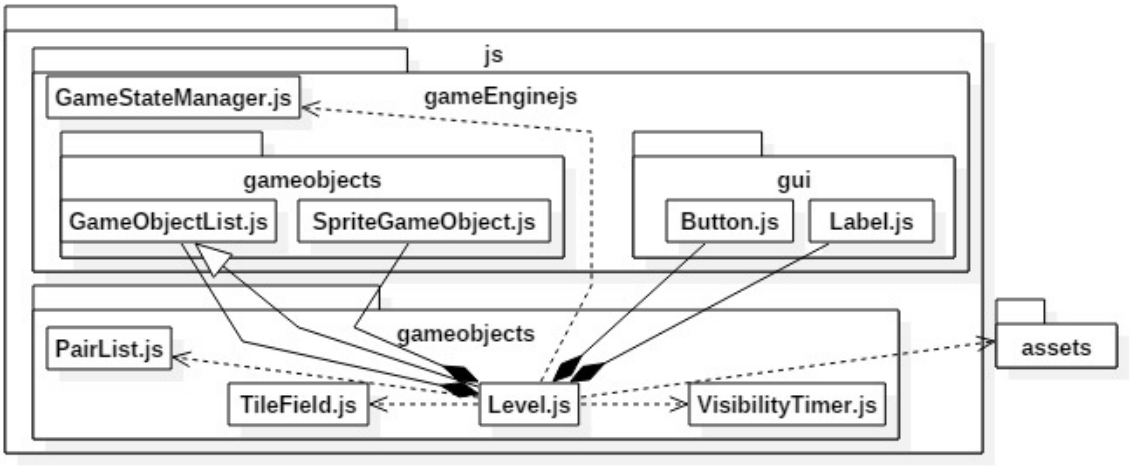
gameobjects



UML Diagram 15. Game specific game objects, js/gameobjects.

The *VisibilityTimer.js* is used as a timer during the game, the time is decreased with every update called from the game's main loop from *Game.js* .

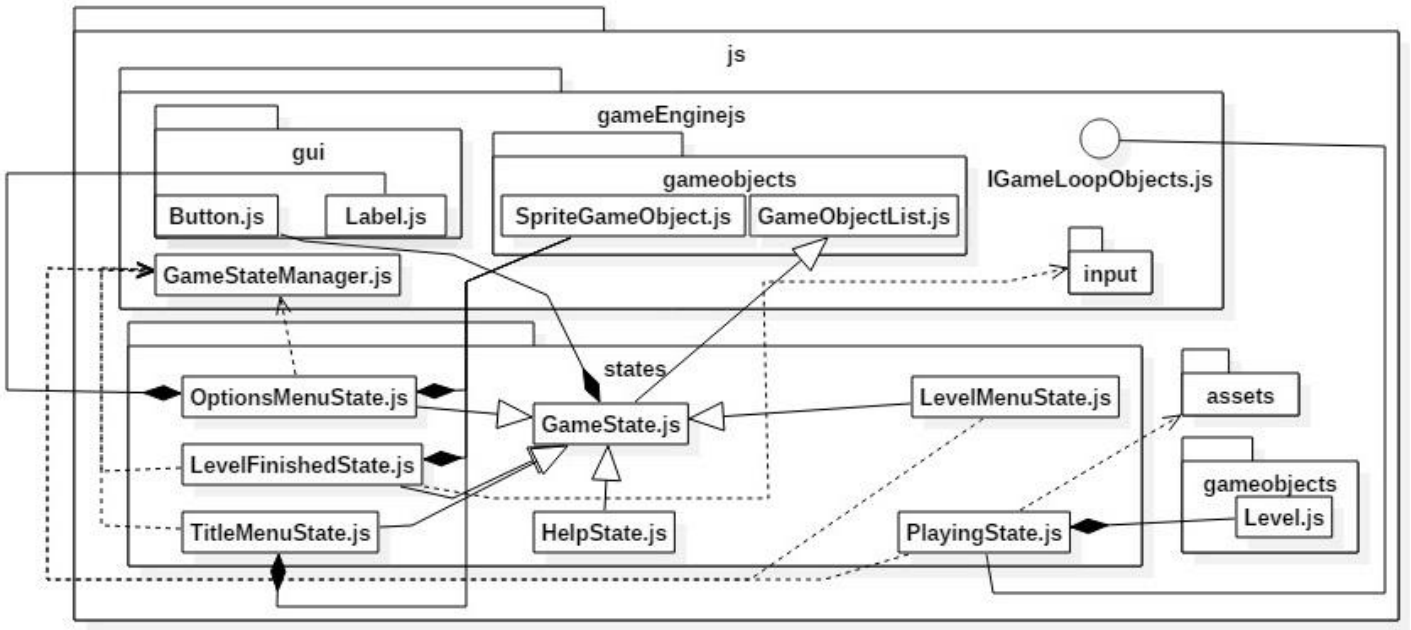
The *PairList.js* is used to graphically represent how many candy pairs the player has obtained. The *Candy.js* represents the candy objects in the game, and the *Tile.js* represents what type of tile (normal, background, wall) will make up the *TileField.js* (the board) in the game. The *Controls.js* represent the game controls within the game.



UML Diagram 16. js/gamebojects/Level.js .

Finally, the *Level.js* and *MultiPlayerLevel.js* (same dependencies) represent the levels played in the game. Their responsibility is to build the level from the level designer's plan and handle the user interaction.

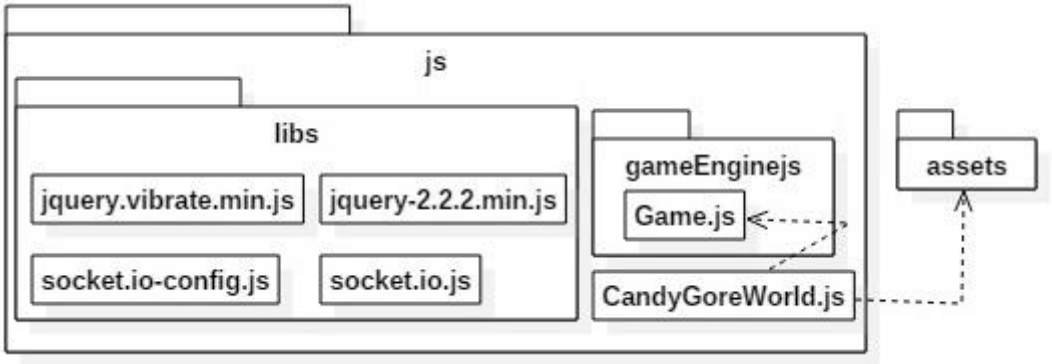
states



UML Diagram 17. js/states contents.

The states of the game are represented by the classes inside the *js/states* folder. There is a basic class from which almost all extend, the *GameSate* class. The game was developed with the mind that it is composed of different states (or scenes) that interact with the player in different ways. The user navigates through these states throughout the course of the game, spending most of their time at the *PlayingState.js*, in charge of the actual game playing. It is the *GameStateManager.js*' job to deal with the different states of the game, switch from one to the other and find them. The game states were designed to run completely independently from each other.

CandyGoreWorld.js and libs



UML Diagram 18. js/libs and js/CandyGoreWorld.js

The *libs* folders holds the necessary Bicomm files needed for the full duplex real-time communication. The jQuery library is used to enhance the control's experience (when the user chooses the controls to be outside of the game); if it is a phone, and it allows it, it lightly vibrates with every tap on the controls.

The *CandyGoreWorld.js* is in charge of initializing the game and loading the appropriate sprites associated with the game.

style and assets

The *styles* folders holds the game site's images and CSS style. Because the style of the site that presents the game was so basic, vanilla CSS with the help of the Bootstrap framework, was employed.

The *assets* folder holds all the sprites and sprite sheets the game employs, as well as the sound files. The graphics carried out for this game were done in Adobe Photoshop

(thanks to the license provided by the University) after studying various lessons from Bert Monroy in Lynda.com.[19]

HTML files

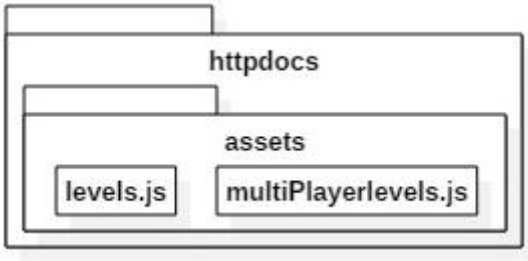
The entry point to the game is through *index.html*, in this page the player chooses where they want the controls to be:

- On the PC (that means the controls are within the game)
- On phone

  This choice is the one that uses Bicomm's service
  - Single Player
  - Two Players

The *candygalore.html* is the file that responds to the first choice and the *multiplayercandyGalore.html* is for the second choice. The *controls.html* are the controls outside of the game (when the player chooses controls to be on phone).

Game Level Design



UML Diagram 19. Game Level Design.

Part of developing a game is putting thought into the game level design, which in big games is a job of its own. The ideal set up is for the game level designer to design whatever kind of level they have in mind without knowing a lot about code. This allows non-programmers to excel at their level design task, helping effectively in the development of the game. To achieve this, the levels in this game are created through characters arranged in a grid:

```
window.LEVELS.push({
    ...
    tiles :                 ["#########",
                             "#.......#",
                             "#...b...#",
```

[19] "Bert Monroy", Lynda.com, http://www.lynda.com/Bert-Monroy/27-1.html, (June 9, 2016)

```
                                "#.......#",
                                "#.......#",
                                "#...b...#",
                                "#.......#",
                                "#########"]
});
```

Each character represents one of the classes previously presented. It is the *Level.js* and *MultiPlayerLevel.js*'s responsibility to read these levels in the form of characters and translate them to game objects.

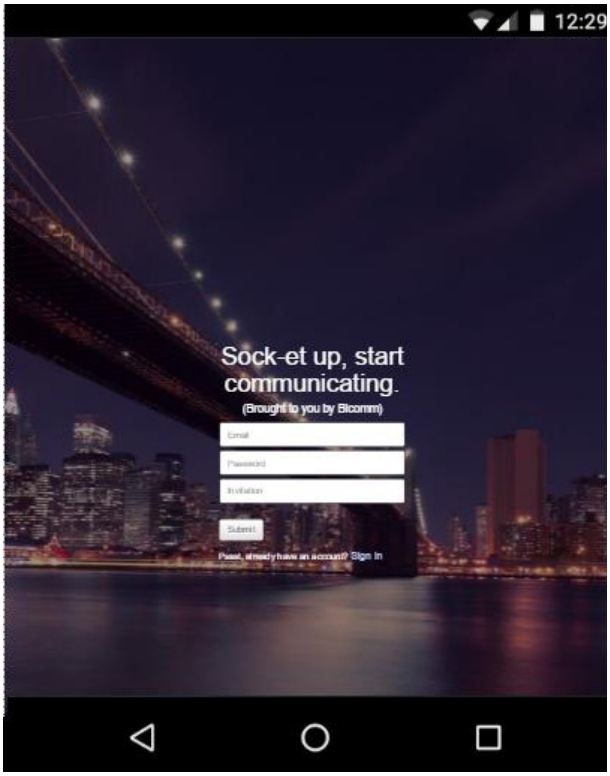
- '.' is the *Tile.js*
- ' ' is the *Tile.js* (background type)
- 'r,b,g,o,p...' are the *Candy.js* (the letter represents the color, if the letter is capitalized then the candy is initially boxed). When designing the level for two players, to distinguish between which candy is for which player, the numbers 1 and 2 have to appear before these characters ('1b, 2r').
- '@' is a *SpriteGameObject.js* representing the kid that eats the candy
- Anything else is a *Tile.js* (wall type)

This makes it real easy to change existing levels or add new ones.

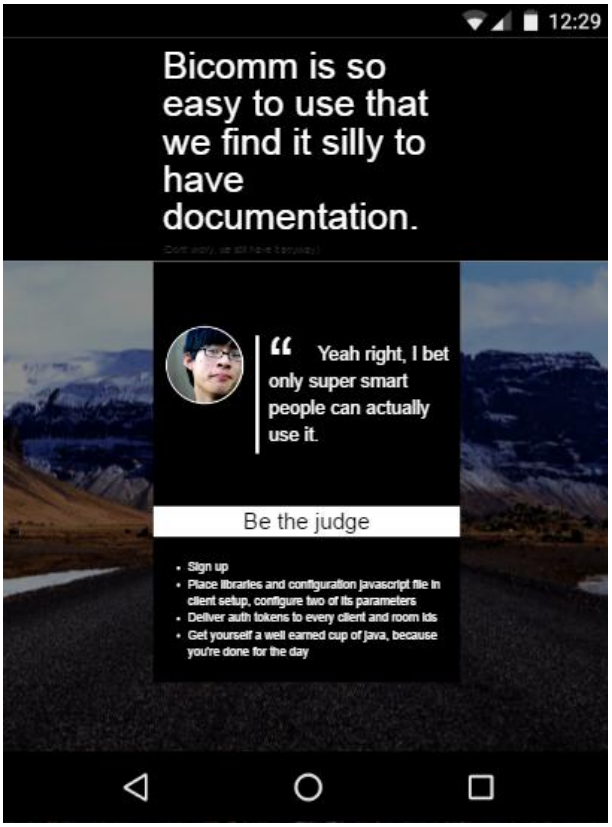
CHAPTER 3 - RESULTS

This chapter will present the results of both the framework and the game through images of the final products.

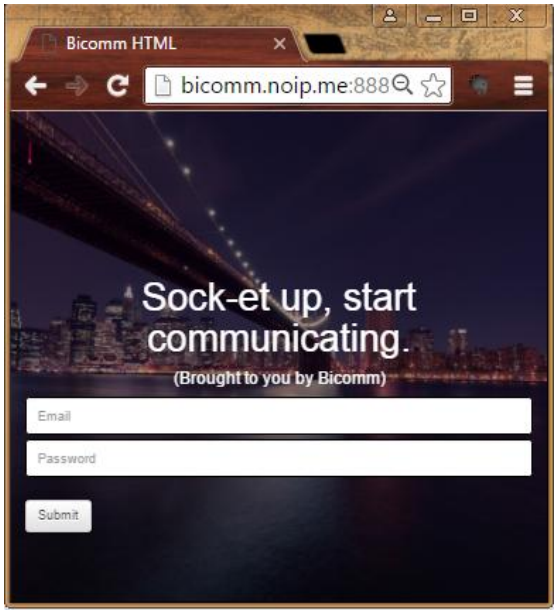
3.1 Framework



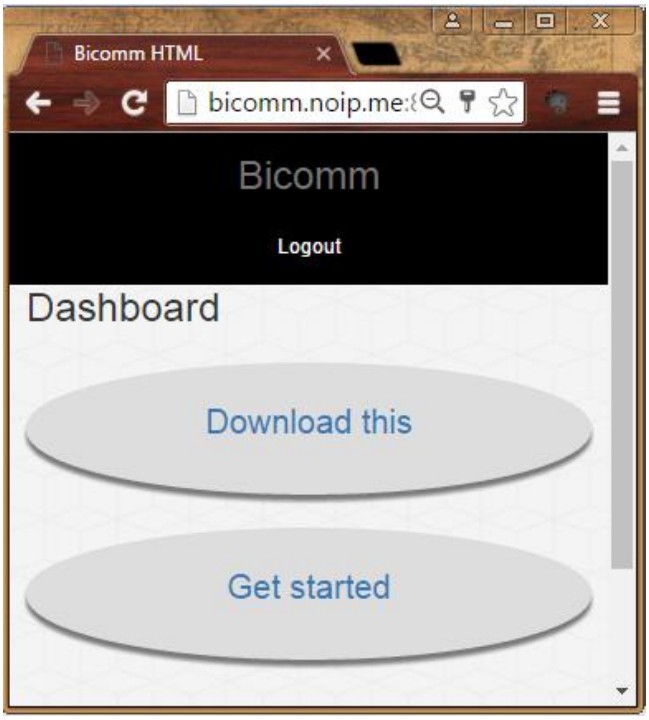
Results 2. Bicomm sign up (Nexus).



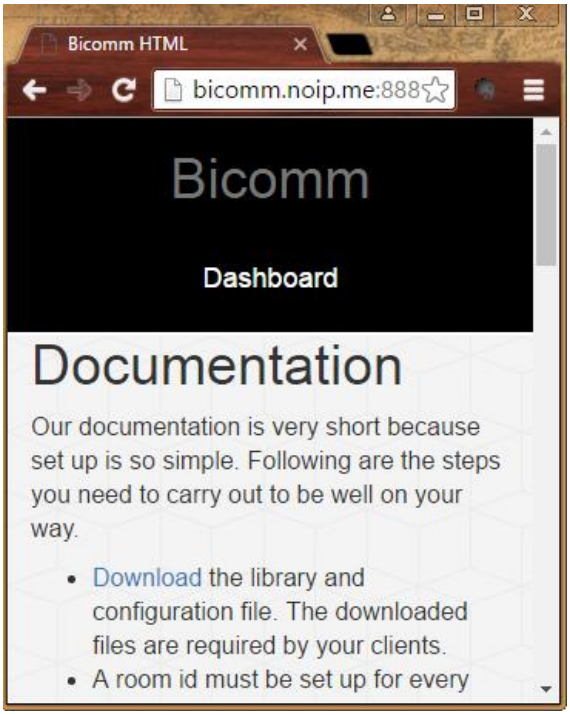
Results 1. Bicomm sign up (Nexus).



Results 3. Bicomm sign in (PC).



Results 5. Dashboard (PC).

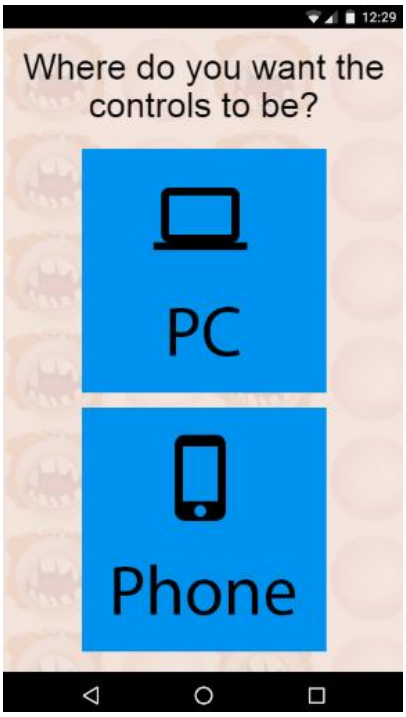


Results 4. Documentation (PC).

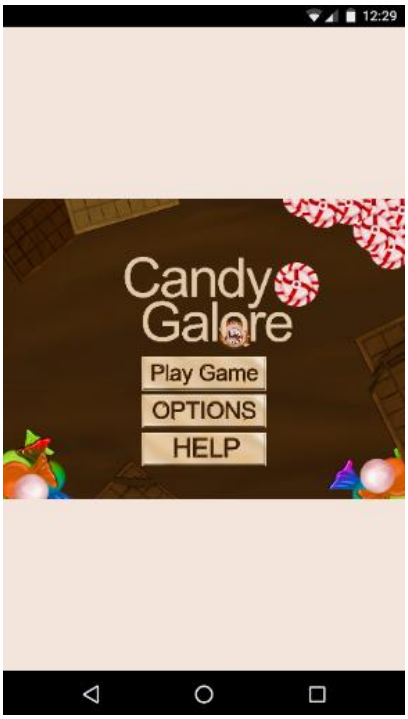
```
db.users.find()
"_id" : ObjectId("56a8d095941cf74c0e74aaa1"), "name" : "iamterry", "password" : "123", "plan" : "reg", "__v" : 0 }
db.invitations.find()
"_id" : ObjectId("567288f37be632740b2c3892"), "number" : "hellowehopeyoustay123", "taken" : true, "taken_by" : "iamterry"
"_id" : ObjectId("567289237be632740b2c3893"), "number" : "welcomenewcomer456", "taken" : false, "taken_by" : "" }
"_id" : ObjectId("5672893b7be632740b2c3894"), "number" : "ahoymatey987", "taken" : false, "taken_by" : "" }
"_id" : ObjectId("567289737be632740b2c3896"), "number" : "ahbutyouhaveheardofme2", "taken" : false, "taken_by" : "" }
"_id" : ObjectId("569cfc6ee55a10576bcee9d7"), "number" : "butwhyistherumgone0", "taken" : false, "taken_by" : "" }
```

Results 6. MongoDB.

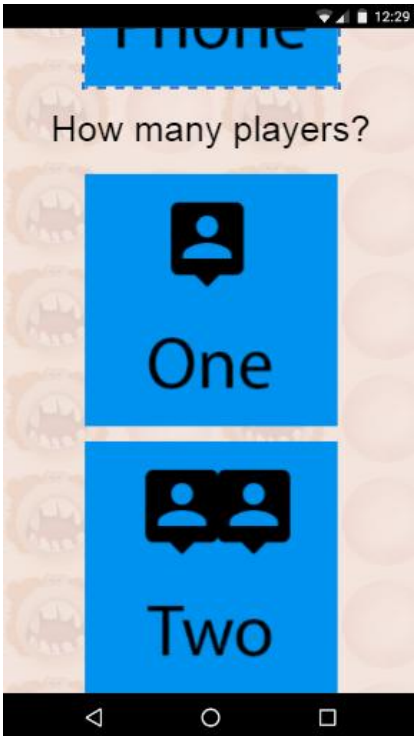
3.2 Game



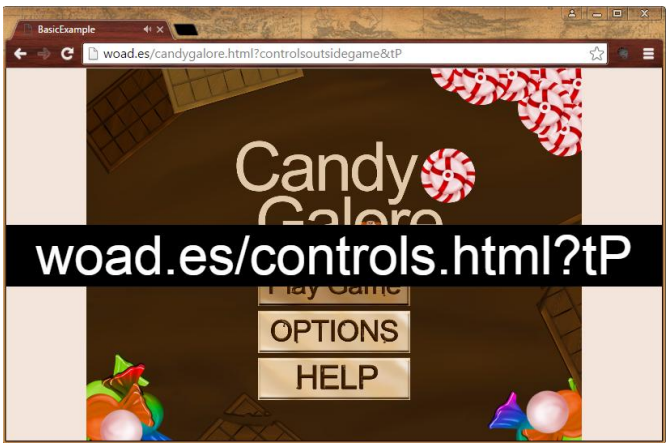
Results 8. Choosing controls. (Phone)



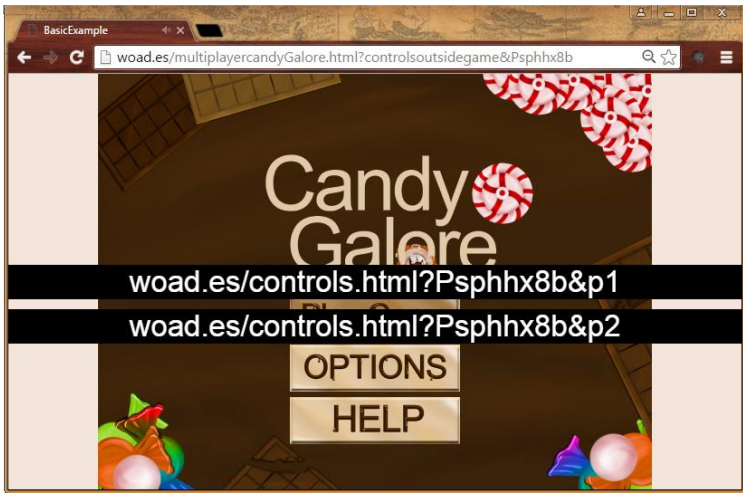
Results 7. Game entry. (Phone)



Results 9. Choosing number of players. (Phone)

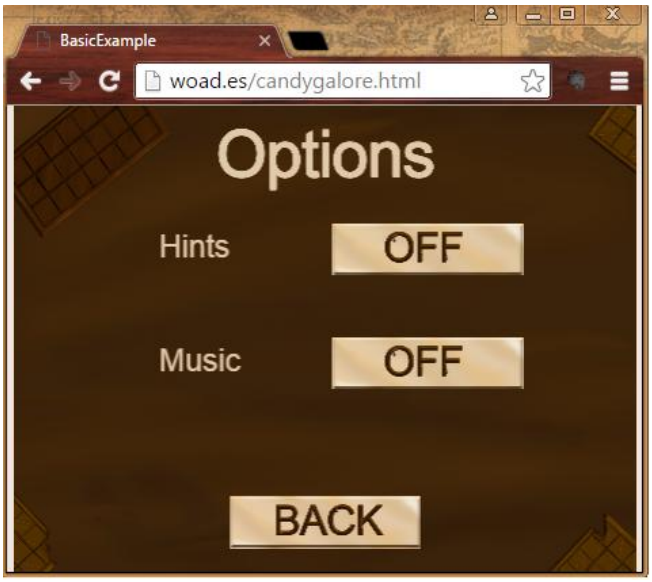


Results 10. Single player game entry. (PC)

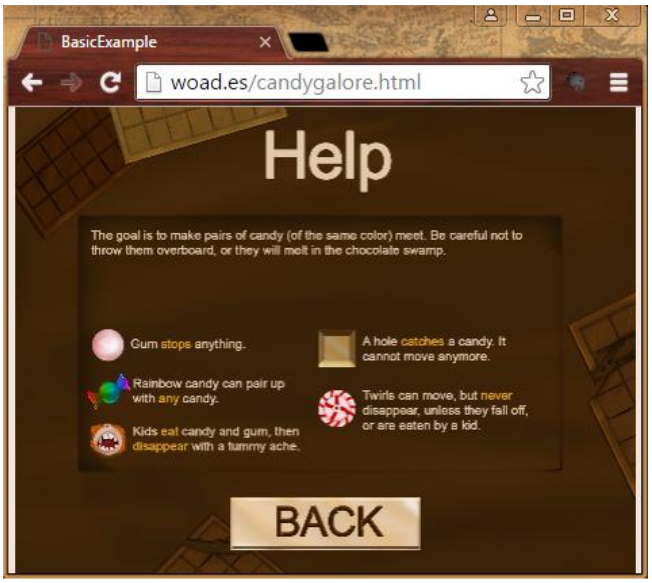


Results 12. Two player game entry. (PC)

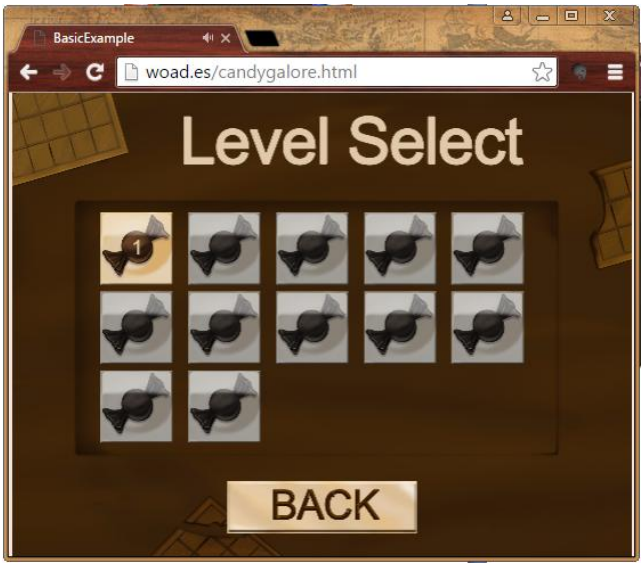
Results 11. Player controls. (PC)



Results 13. Options Menu. (PC)

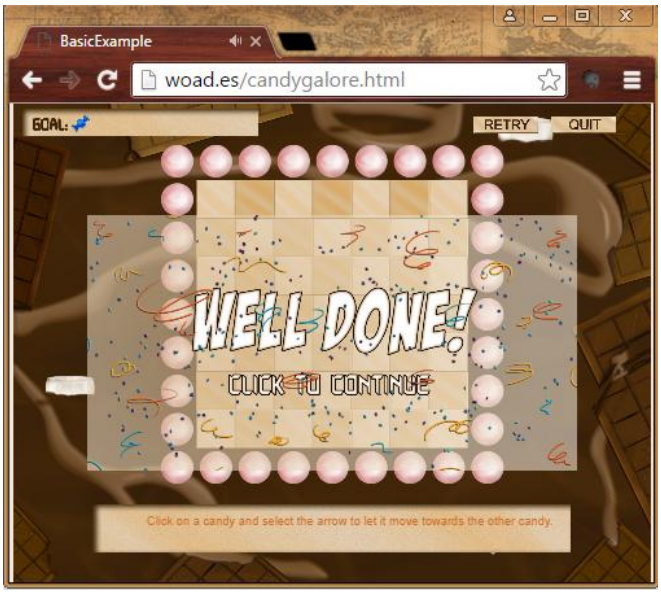


Results 14. Help Menu. (PC)

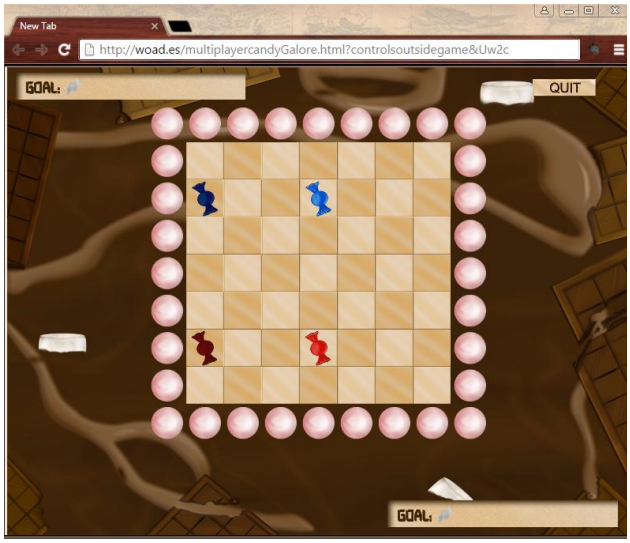


Results 15. Level Select Menu. (PC)

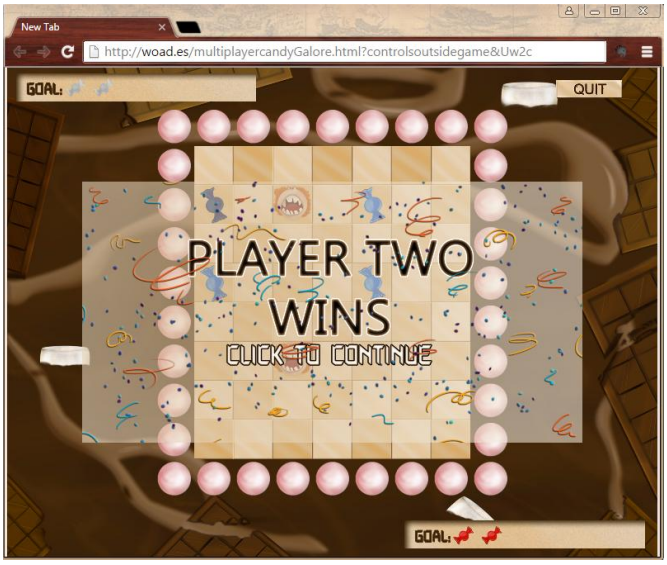
Results 18. Single player playing state. (PC)



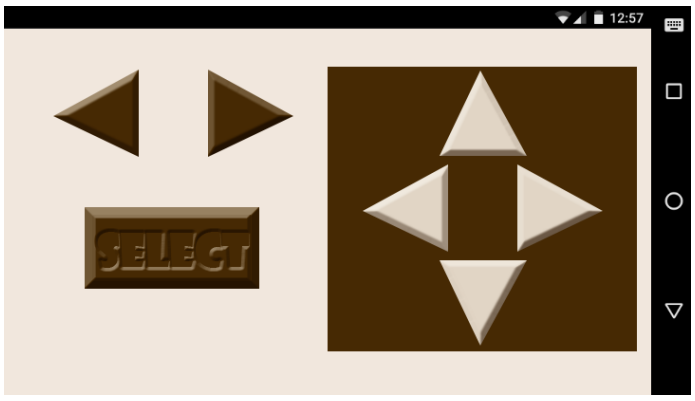
Results 17. Single player game won state. (PC)



Results 16. Two players playing state. (PC)



Results 19. Two players game won state. (PC)



Results 20. Playing controls. (Phone)

CHAPTER 4 - BUSINESS PLAN

A formal business plan is a statement of business goals, targeting areas such as the market, the product, the problem, capital and resource requirements, so on so forth. This chapter presents a first draft of Bicomm's business plan defined through an elevator pitch and the Business Model Canvas.

4.1 Elevator Pitch

The elevator pitch concept is born from the idea of being able to impress a possible investor during a brief ride in an elevator. It is meant to quickly provide an overview of a product. It first became popular and referenced to pitches made by entrepreneurs to Silicon Valley capitalists. The characteristics that make up an elevator pitch are:

- Not longer than 30 seconds
- Approximately 80-90 words
- Answers some or all of the following: who you are, what you offer, what the benefits are and how do you do it.

This business plan's elevator pitch is:

> Ever asked one of your developers to conjure a real-time chat compatible with all clients? Did they make you feel like if you were asking for the moon? Not any more, not with Bicomm in play. Our company offers just that within minutes of configuration, real-time communication for developers to set up in their system without having to worry about technology compatibility. It's easy, stress-free and powerful thanks to the now popular Node.js. Your developers will be happy and they will make you happy, all thanks to Bicomm.

4.2 Business Model Canvas

The Business Model Canvas (BMC) was presented in the class "Entrepreneurship in IT" for the Master in Computer Engineering: Web Technology and Digital Business[20].

[20] "Máster en Ingeniería Informática: Tecnología Web y Negocio Digital", http://eii.ulpgc.es/blogs/estudios-eii-mii/master-en-tecnologia-web-y-negocio-digital/ , (June 10, 2016)

BMC

The BMC is meant to give a coherent structure and view to a business plan through its nine elements:

- Customers
  This element describes the customers. Who are the customers? What do they think? What do they do? In this segment the dimensions of the market are also defined. A business can have a single or multi-sided market. A multi-sided market has different types of clients, like a newspaper (the clients are the readers on one hand, and the advertisers on the other).

- Value Proposition
  The business's value must be presented in this element. What is it about the business that will compel others to buy its service? The ideal link would be to clearly explain how the customers defined in the Customers section have their problems solved through the Value Proposition. If current alternatives exist, then the company's outstanding differences must be explained, explaining how they are better over the existing options.

- Channels
  The business must have a way to reach their propositions. How are they promoted, sold and delivered?

- Customer Relationships
  The way in which a customer interacts with the company throughout the sales and product lifecycle is defined as Customer Relationships.

- Revenue Streams
  In this segment the ways in which the business earns revenue have to be specified.

- Key Activities
  The Key Activities are the principal actions the business carries out to deliver its Value Proposition.

- Key Resources
  There are a certain set of assets the business needs to carry out its service, those are the Key Resources.

- Key Partnerships
  At times businesses establish partnerships to relieve unimportant activities onto them, allowing the business to employ all of its attention towards the Key Activities.

- Cost Structure
  This element defines what the business spends money on, linking it to the Revenue Streams.

Bicomm's BMC

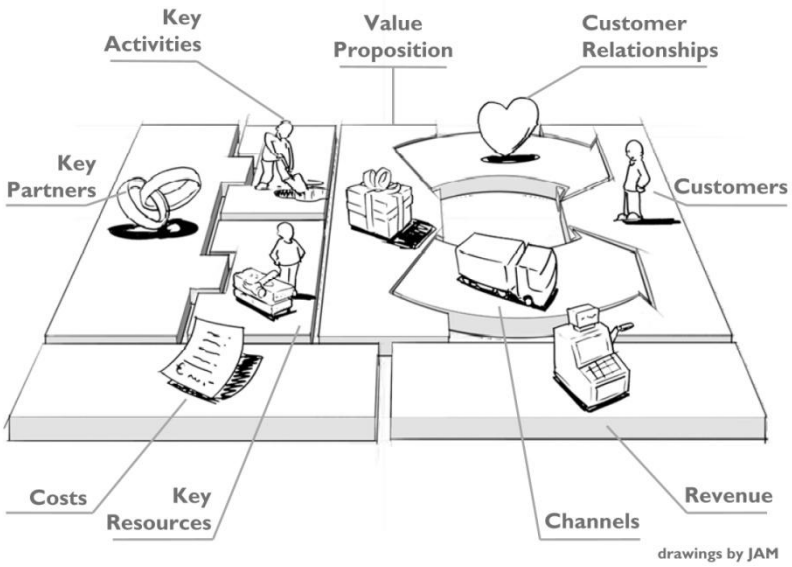


Illustration 20. BMC.[21]

Customers

Bicomm's service is directed towards a single side market, online projects that need to set up simple full duplex communication in real-time. In order to use the service the customers will need backend and frontend knowledge. This definition encloses all sort of web development projects such as games, chats...etc.

[21] "Business Model Canvas", Espirex, http://espriex.co/business-model-canvas , (June 13 2016)

Value Proposition

Bicomm offers a simple set up to get full duplex communication going within minutes. The current alternatives to Bicomm are frameworks or libraries that are technology dependent. This means the customers not only need to install the technology but also know how to program in that specific language, setting up the library or framework server side. They are a lot of setbacks if the customer wants to exchange a few messages between their clients in real-time. That is where Bicomm comes in, the service offered can be used in any kind of server-client (be it PHP, Ruby on Rails, Java EE...) set up. The only technology it uses is the client side running JavaScript. The configuration can be completed within minutes. In other words, it is fast and easy to use.

Channels

The channels define how Bicomm communicates its proposition to the customers previously defined. One of the business's main channels is its site. It is where customers can sign up and hire the service, download the configuration files and read the manual to know how to get started.

Another important channel is through Google AdWords. It is important for Bicomm to become known to the Customer Segments it aims for, and Google AdWords allows this through special keyword searches. When these keywords are used in Google's engine then a link to the Bicomm site will be offered as an advertisement. This is quite lucrative since Bicomm will be able to directly tackle the Customers segment it is defined for, the customers whose problems can be solved by Bicomm's services. Some of the keywords could be: communication, real-time, chat, online...etc.

Customer Relationships

This could be Bicomm's weakest point in regards to its Business Plan. The customer does not directly interact with any person in order to start using Bicomm's service. Both the sales and product lifecycle are all done through the web. Because one of Bicomm's Value Propositions is the ease with which the service can be used it becomes necessary for the documentation to have no faults and be understood even by entry level developers. It would be advisable, in the future, for Bicomm to offer premium post-sale service for those who require it.

Revenue Streams

Bicomm's Revenue Streams are directly mapped to the Customers. Bicomm offers a service where the client will set up the given library and use Bicomm's server as a middleman for the communication between their clients and server. This implies that each customer will be using different amounts of server-client traffic at any given time. Due to this, a permanent license cannot be applied and a subscription model is better suited instead. The customers would sign up for a subscription much similar to those offered by web hosting services. The price would vary depending on the amount of bandwidth desired.

Key Activities

The Key Activities are meant to list the crucial activities the business must carry out in order to deliver its propositions. Bicomm is a service-driven business, making sure clients' needs are met with the most up to date technology is one of the key activities that it carries out. To be able to do this, things like maintaining expertise of Node.js knowledge, and constant learning about users and new techniques to improve the existing product are some of the Key Activities that need to be done.

Key Resources

As mentioned before, Bicomm is a service-driven business, offering one product: full duplex communication in real-time. This helps filter out the assets that need to be in place in order to fulfill the Key Activities. Bicomm's principal Key Resources are Node.js and Socket.io expertise, as well as web traffic optimization (needed to satisfy all clients equally, independently of their geographical location) and a good server infrastructure.

Key Partnerships

Bicomm's service relies on the internet, offered by web hosting companies. It is easy then to see that Bicomm's Key Partnership is their web hosting company, one that should be able to satisfy their need in order to provide their Value Propositions to the Customers.

Cost Structure

The Key Activities, Resources and Partnerships drive costs to maintain them. The cost in terms of the Key Partnership would need to be able to cover the expected traffic that is to be given to the Customers, so it could be variable as the business tests different

business models. Of course, the Revenue Stream should cover those costs when enough customers have signed up with Bicomm.

CHAPTER 5 - CONCLUSION AND PROJECT'S FUTURE

5.1 Conclusion

This master's thesis concludes having achieved all the goals envisioned and something more along the way. Bicomm was successfully established, allowing anyone to implement full duplex communication in real-time within minutes. The framework's ease of use was tested while developing an online game that creatively used the service to add something more to the playing experience by providing controls through a phone. Finally, establishing a business plan changed my view from developer to business woman, having to think about how to turn an idea into a profitable scheme. It was eye opening comprehending how important it is to have a business model well defined to understand where the idea might be weak and what needs more development.

Carrying out this project from initial concept on paper to live site and game have provided me with in depth knowledge about JavaScript, online game architecture, game development and Photoshop. I have also dipped into Node.js and Socket.io, but there is still a lot more to learn from those great technologies.

It was quite satisfactory to apply what I learned through my master's degree and this thesis into my day job. Thanks to my knowledge in Node.js two projects in my work have been carried out with this technology, shortening development time and achieving applications that have worked smoothly since set in production. While developing Bicomm, one of the problems I had trouble overcoming was the CORS implementation, I could not quite find the information I needed. In the end, I managed to solve it and helped others along the way, by posting my solution in the well known site that is StackOverflow[22].

The knowledge I harnessed in JavaScript made me comfortable enough to play in this year's Tuenti Challenge 6[23], solving the problems in JavaScript (executing them with Node.js) and managing to become number 136 out of 781 even though I only got to problem 5. Developing an architecture with OOP (Object Oriented Programming) applied with JavaScript has opened my eyes to how powerful, and delightful, this language truly is. I was very happy to develop the game, understanding the

[22] "Cross-domain connection in Socket.IO" , stackoverflow, http://stackoverflow.com/questions/8970880/cross-domain-connection-in-socket-io/33912510#33912510 , (June 15, 2016)
[23] "Phase 1: Final Ranking", Tuenti Challenge 6, https://contest.tuenti.net/Stats , (June 15, 2016)

requirements needed and how to implement them. I will be keeping the game engine I developed for reference or use in future projects.

While Photoshop and illustration was not part of the goals set in this master's thesis, I ended up learning how to use it in order to provide the game with decent images. The site Lynda.com was extremely helpful, as well as the opportunity to use Photoshop thanks to the license provided by the ULPGC. The comprehension I gathered was enough to help others by answering questions in the site, as a subdomain of StackExchange, GraphicDesign[24]. When the ULPGC offered a chance to take the test to qualify as an "Adobe Certified Associate in Visual Communication Using Adobe Photoshop CC", at no cost to the student, I was quick to grab the chance, and proudly earned the certificate.

I conclude that this thesis as well as the master's degree have greatly helped me develop my knowledge in online technology, enhancing my full stack developer profile, as well as opened a door to game development and illustration. As I look to the future I think about delving into C++, virtual reality and artificial intelligence to pave my path towards game development.

5.2 Project's future

The working results of this project are but a first draft to what could be. Bicomm essentially has many improvements to be made to become a full blown business:

- Identify the client through the traffic received. At the moment Bicomm simply offers the service to the petitions received, not knowing if the petition is from a subscribed client or not.
- Measure and limit how much traffic a client produces. Without this feature the Revenue Streams are nonexistent.
- Research how to protect the client from having his subscription hijacked. This dilemma was looked into during the development and discussed but no answer was attainable. The problem relies in that anybody can download the client's JavaScript files to play the game, when one downloads the files they are easily viewable through any browser console. Once inspected, they can easily find the room id they use and obtain Bicomm's configuration file.
- Connect Bicomm with a paying gateway so clients can carry out a subscription.

[24] "JMR", graphicdesign, http://graphicdesign.stackexchange.com/users/39730/jmr , (June 15, 2016)

- Improve time response between the user sending the message and the server receiving it (improve the time it takes the message to travel to Bicomm server and then for it to be broadcasted).

As can be appreciated, Bicomm's project is quite young in its development, there's plenty of room for future students to develop it further.

BIBLIOGRAPHY

[1] Avin. Avin's Blog. "Adding CSRF token to jQueyr AJAX requests." 3 February 2013. 9 January 2016. <http://avinmathew.com/adding-csrf-token-to-jquery-ajax-requests/>

[2] Balliauw, Maarten. Maartenballiauw. "Techniques for real-time client-server communication on the web (signalR to the rescue)." 29 November 2011. 3 December 2011. <http://blog.maartenballiauw.be/post/2011/11/29/Techniques-for-real-time-client-server-communication.aspx>

[3] Egges, Arjan. *Building JavaScript Games for Phones, Tablets, and Desktop.* Apress, 29 September 2014.

[4] Express. "Serving static files in Express." 8 December 2015. <http://expressjs.com/en/starter/static-files.html>

[5] Express. "Installing." 9 November 2015. <http://expressjs.com/en/starter/installing.html>

[6] Hartmann, Christoph. Lollyrock. "Encrypt and decrypt content with Nodejs." 10 January 2015. <http://lollyrock.com/articles/nodejs-encryption/>

[7] Holbrook, Josh. Nodejitsu. "How to use crypto module." 20 August 2011. 9 December 2015. <https://docs.nodejitsu.com/articles/cryptography/how-to-use-crypto-module>

[8] Kiessling, Manuel. "The Node Beginner Book." The Node Beginner Book. 9 November 2015 <http://nodebeginner.org/>

[9] King, Kelly. Codeplanet. " How to Make a Single Page Website." 18 October 2015. 15 December 2015. <https://codeplanet.io/how-to-make-a-single-page-website/>

[10] Kitamura, Eiji. Ubl, Malte. Html5rocks. "Introducing WebSockets: Bringing Sockets to the Web." 20 October 2010. 19 December 2015. <http://www.html5rocks.com/en/tutorials/websockets/basics/>

[11] Leggetter, Phil. Leggetter. "Making cross domain JavaScript requests using XMLHttpRequest or XDomainRequest." 12 Match 2010. 13 November 2015. <http://www.leggetter.co.uk/2010/03/12/making-cross-domain-javascript-requests-using-xmlhttprequest-or-xdomainrequest.html>

[12] mongoDB. "Install MongoDB on Red Hat Enterprise, Cent0S, Fedora, or Amazon Linux". 4 December 2015. <https://docs.mongodb.org/v2.4/tutorial/install-mongodb-on-red-hat-centos-or-fedora-linux/ >

[13] mongoDB. "UNIX ulimit Settings". 4 December 2015. <https://docs.mongodb.com/manual/reference/ulimit/>

[14] Mongoose. "Finding documents." 8 December 2015. <http://mongoosejs.com/docs/2.7.x/docs/finding-documents.html>

[15] Node.js. "Node.js v6.2.2 Documentation." 9 December 2015. <https://nodejs.org/api/>

[16] Petkov, Petko. Websecurify. "Hacking Nodejs and MongoDB." 11 August 2014. 10 January 2015. <http://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html >

[17] Robbins, Charlie. Nodejitsu. "Keep a node.js server up with Forever." 30 November 2010. 12 November 2015 <http://blog.nodejitsu.com/keep-a-nodejs-server-up-with-forever/>

[18] Sakry, Mary. Potter, Neil. Census. "Clearer Software Requirements Using a Concise Template." 1 November 2015. <http://www.census.gov/cspi/pdf/Clearer_Software_Requirements_Neil_Potter.pdf>

[19] Sevilleja, Chris. Scotch.io. "Learn to Use the New Router in ExpressJS 4.0." 14 April 2014. 2 December 2015. <https://scotch.io/tutorials/learn-to-use-the-new-router-in-expressjs-4#our-sample-application>

[20] Socket.io. "Get Started: Chat application." 9 November 2015. <http://socket.io/get-started/chat/>

SOURCES OF IMAGES USED

[1] Anders Jildén. Unsplash. 19 January 2016. <https://unsplash.com/@andersjilden>

[2] Exasperation. Freepik. 20 January 2016. <http://www.freepik.com/free-photo/exasperation_41662.htm#term=annoying&page=1&position=6>

[3] Subtle Patterns. 15 January 2016. <http://subtlepatterns.com/>