Escola Universitària
Politècnica de Mataró

Centre adscrit a:

UNIVERSITAT POLITÈCNICA
DE CATALUNYA

GRAU EN ENGINYERIA INFORMÀTICA

YOURTALKS.COM

Frontend i Real Time

Memòria

FRANCESC GIL CARDONA

PONENT: JOSEP ROURE ALCOBÉ

PRIMAVERA 2014

TecnoCampus
Mataró-Maresme

Dedicatòria

A la meva família i amics.

Resum

L'objectiu del projecte YourTalks es crear una plataforma web per a la realització de conferències/classes o qualsevol altre tipus de xerrada en temps real. YourTalks vol oferir una experiència el més semblant possible a l'assistència física, a la conferència i complementar-ho amb serveis addicionals que aportin un valor afegit tant als oients com al ponent. Aquests serveis facilitaran que els locutors facin les seves explicacions més interactives i també inclouran la possibilitat d'interactuació entre tots els assistents un cop acabada la conferència.

La plataforma serà modular de manera que es pugui configurar la sala amb diferents funcionalitats i serveis segons la voluntat del ponent i dels assistents. A més ha de permetre la creació de nous serveis a mesura que vagi apareixent noves necessitats i vagi madurant la pròpia plataforma. Per a la creació técnica del projecte s'ha realitzat una extensa cerca i aprenentatge de diverses tecnologies.

Resumen

El objetivo del proyecto YourTalks es crear una plataforma web para la realización de conferencias/clases o cualquier otro tipo de charla en tiempo real. YourTalks quiere ofrecer una experiencia lo más parecida posible a la asistencia física, a la conferencia y complementarlo con servicios adicionales que aporten valor añadido tanto a los oyentes como al ponente. Estos servicios facilitarán que los locutores hagan sus explicaciones más interactivas y también incluirían la posibilidad de interactuación entre todos los asistentes una vez terminada la conferencia.

La plataforma será modular de manera que se pueda configurar la sala con diferentes funcionalidades y servicios según la voluntad del ponente y los asistentes. Además debe permitir la creación de nuevos servicios a medida que apaezcan necesidades y vaya madurando la propia plataforma. Para la realización técnica del proyecto se realiza una extensa búsqueda y aprendizaje de diversas tecnologías.

Abstract

The main objective of the project YourTalks is to create a web platform to make conferences, classes or any other type of talk in real time. YourTalks wants to offer the same experience as attending to a real conference, and complement it with extra services that add value to the attenders and the announcer. This service will help the presentations of the announcers to be more interactives, and also, after the conference, there's the possibility to interact with any of the attenders.

The platform will be modular, so the announcer can add or remove services (modules) of the conference room. Also it should allow the creation of new services for the new necessities we may found, as the platform grows up. There was an extensive research and learning of the new technologies, for the project technical development.

Índex

Índex de figures.

Índex de taules.

1. Objectius

1.1 Propòsit

Disseny i implementació de l'estructura base de la plataforma YourTalks. YourTalks és una aplicació web que ajuda a donar conferències/classes o qualsevol altre tipus de xerrada en temps real a través d'internet. Permet l'organització de les conferències en sales amb la possibilitat de complementar-les amb diverses funcionalitats o serveis, com per exemple, un xat, videotrucades, llistat de preguntes, presentacions i enquestes, el qual permetrà una interacció entre oient-ponent i oient-oient. D'aquesta manera es vol donar un valor afegit, permetent la interactuació entre oient i ponent més enllà de la duració de la conferència i oferir una xarxa social entre ells. També es vol que sigui un punt de connexió entre tots els ponents del món.

1.2 Finalitat

Aconseguir una plataforma web per donar conferències/classes o qualsevol altre tipus de xerrada, a través d'internet en temps real. Ajudar als locutors a realitzar les seves explicacions més interactives i amenes per al oients donant un valor afegit, permetent la interactuació entre oient i ponent més enllà de la duració de la conferència i oferir una xarxa socials entre ells.

1.3 Objecte

Un servei web per realitzar presentacions en temps real, en model SaaS (Software as a Service) de distribució. SaaS és interessant perquè evita a l'usuari de qualsevol tipus de manteniment sobre la plataforma, tot ho gestiona el proveïdor.

1.4 Abast

Tota la idea explicada anteriorment requereix un llarg desenvolupament, recursos i temps. S'ha de dedicar molt de temps a la investigació i estudi de noves tecnologies, perquè no s'han treballat a classe, per poder desenvolupar la plataforma. Ens centrarem en la base de l'aplicació (Core), que consisteix en crear sales, unir-te a les sales, poder xatejar amb la

gent de la sala, crear presentacions i veure presentacions en temps real. Tenir una base ben estructurada ens permetrà desenvolupar en un futur amb molta més agilitat i rapidesa.

Pel que fa el desenvolupament s'utilizarà Ruby, Ruby on Rails, JavaScript, Grunt, BackoneJS, MarionetteJS, Puppet i Bash. Per la part de servidors s'utilitzarà eines d'automatització com ara Puppet, Capistrano i servidors web com NodeJS, Nginx, Unicorn.

2. Metodologia de treball

Per a la gestió del projecte es necessita un sistema d'organització enfocat a treballar en equip. Es vol utilitzar metodologies àgils, que són aquelles metodologies de desenvolupament que es basen en l'adaptabilitat de qualsevol canvi com a mitjà per augmentar les possibilitats d'èxit d'un projecte. Per a la gestió del flux de treball s'utilitzarà el mètode Kanban que permet dividir el procés productiu en diferents fases perfectament limitades, un tipus de mètode per aplicar la metodologia àgil. Kanban, té dos objectius: d'una banda, aconseguir un producte de qualitat, ja que obliga cada fase del projecte a finalitzar la seva tasca correctament, i d'aquesta manera acabar amb el caos o coll d'ampolla que es pot donar en una fase del projecte en condicions normals en les quals preval la rAPIdesa per sobre de la qualitat del producte. Kanban està pensat per a projectes de software i per augmentar la productivitat de l'equip.

Per poder implementar aquest mètode s'ha escollit la plataforma de gestió de tasques JIRA de l'empresa Atlassian. JIRA és un gestor de tasques que permet als equips planificar, construir i finalitzar projectes. Pel sistema de control de versions s'ha utilitzant el producte Bitbucket de l'empresa Atlassian. Permet la integració amb el sistema de control de versions GIT, un sistema de control de versions distribuït, dissenyat per poder suportar tant projectes petits com grans projectes amb rapidesa i eficàcia, Open Source.

3. Anàlisis de requeriments

En aquest apartat es detalla de forma descriptiva totes les funcionalitats del projecte YourTalks. Ara només ens centrarem en desenvolupar l'estructura bàsica de l'aplicació, anomenada Core. El Core es compon de l'anàlisis de l'arquitectura dels servidors, una API REST amb les funcions més bàsiques, per després poder continuar desenvolupant les altres característiques que seran necessàries per tenir el producte 'complet'. El sistema de processament de les presentacions pujades pels usuaris i el sistema de real time per a la presentació amb el módul de chat.

YourTalks és una aplicació web que ajuda a donar conferències/classes o qualsevol altre tipus de xerrada en temps real a través d'internet.

La plataforma serà modular permetent configurar la sala amb les funcios desitjades, anomenades "widgets". Els "widgets" disponibles són:

- **Videoconferència**: l'expositor fa un broadcast del seu àudio i vídeo als oients.
- **Presentacions**: poder mostrar una presentació en temps real.
- **Xat**: un xat entre tots els participants de la sala.
- **Preguntes**: un llistat de possibles preguntes a fer als participants.
- **Q&A**: preguntes que volen fer els participants al locutor.
- **Informe de comprensió**: els participants voten si entenen o no al locutor.

La plataforma es compon d'un sol tipus d'usuaris amb dos rols diferents segon l'acció a realitzar. Si l'usuari és un participant (viewer), de la sala, podrà:

- Observar en temps real la presentació de la sala.
- Visualitzar i escoltar al locutor (àudio i vídeo).
- Podrà xatejar amb els altres participants.
- Podrà formular preguntes al locutor.
- Podrà respondre les preguntes que es realitzin per part del locutor.
- Podrà donar el seu feedback sobre la presentació i el nivell comprensió.

Si l'usuari és un administrador (owner), el creador de la sala, podrà:

- Convidar nous participants a la sala.
- Compartir la seva càmera i àudio entre tots els participants.
- Pujar i compartir una presentació a la sala.
- Escriure en el chat.
- Formular preguntes directes als participants i veure les seves respostes.
- Visualitzar les preguntes que se l'hi formulin.
- Evaluar el nivell de comprensió dels participants.
- Controlar les presentacions des d'un altre dispositiu (per exemple el mòbil) per no haver d'estar sempre davant de l'ordinador.

Algunes de les opcions futures, després d'analitzar l'ús que fan els usuaris de la plataforma podrien ser:

- Convertir-se en una Xarxa Social de Locutors, on la gent que ha assistit a les seves presentacions podrà donar un feedback de com creu que es aquella persona com a locutora, com creu que ha sigut la presentació i valorar-la amb una puntuació.
- Creació d'una aplicació mòbil, per poder seguir les xerrades des de qualsevol lloc.
- Gestió d'events i quedades.
- Integració amb plataformes de tercers (exemple: Meetup).

Es vol que la plataforma sigui un punt de connexió entre tots els conferenciants i el seu punt de referència per adquirir i mostrar nous coneixements. També es vol que la plataforma es pugui integrar amb plataformes de tercers i arribar a ser un Youtube de les presentacions online.

Per poder desenvolupar la plataforma YourTalks fa falta dividir el projecte en diferents parts pel que fa al seu desenvolupament. Per a la creció d'una aplicació escalable i modular s'ha necesitat distingir cada part com una "subaplicació".

En primer terme per treure la plataforma a producció es requereix un sistema amb alta disponibilitat, tolerant a falles, escalable de forma horitzontal i econòmic. Per poder obtenir aquest tipus d'arquitectura es requereix de dos servidors físics, allotjats en datacenters separats, com a mínim per poder tenir el sistema redundat, sempre es poden

afegir més host per suportar pujades de tràfics o un augment dels usuaris. Un sistema de balanceig de càrrega per distribuir les peticions en els diferents servidors. Un sistema de virtualització (per exemple, Proxmox) que ens permeti muntar-lo en clúster i tenir un control de totes les màquines. Diferents servidors virtualitzats redundats per allotjar les diferents parts de la plataforma.

Per saber si els servidors estan actius ens fa falta un sistema de checking, que detecti el mal funcionament o parada d'un servidor o servei, i el mogui en el servidor de reserva. Un sistema de control o monitoreig que xequegi les constants dels servidors i actuï a partir d'unes regles definides i avisi a la persona corresponen.

Pel que fa a la gestió de les màquines es requereix d'un sistema d'automatització i orquestrat de màquines per poder realitzar la gestió dels servidors des d'un únic punt sense haver d'entrar màquina per màquina a gestionar els seus serveis.

Pel logs es requereix un sistema central on es guardin els logs de totes les màquines durant 30 dies, per a la seva consulta i anàlisi en temps real o no. D'aquesta manera ens permetrà detectar falles en les aplicacions i en el sistema.

Per la part de desenvolupament es necessita crear una aplicació escalable, que no carregui tota la lògica en un sol punt com es fa normalment, ja que requereix menys desenvolupament inicial. S'ha separat el model-vista-controlador en diferents parts, el model-controlador la API REST i la vista a l'aplicació frontend.

L'API REST, s'anomenarà Puigmal, es necessita programar amb Ruby on Rails, que ens ha de proporcionar uns 'endpoints' amb les accions disponibles. Conté tota la lògica de la base de dades. L'API correspon al MC (Model-Controlador), només interactuant aquests dos elements, ja que la lògica de la vista està delegada a una de les altres aplicacions, la de Frontend.

El Frontend, s'anomenarà Aneto, es necessita programar amb JavaScript, aquest és el consumidor de l'API REST i el que conté tota la lògica del client, la seva interfície. Es requereix poder muntar una aplicació de frontend amb JavaScript que sigui escalable utilitzant Backbone.Marionette. Backbone.Marionette és un Framework de JavaScript que agilitza el desenvolupament de grans aplicacions en aquest tipus de tecnologia.

L'aplicació de Real time, s'anomenarà Pedraforca, i haurà de gestionar la comunicació entre clients. Es desenvoluparà en JavaScript utilitzant NodeJS i Socket.io. Aquesta comunicació serà essencial dins la plataforma, ja que donarà a l'usuari una sensació de dinamisme de la plataforma.

L'últim mòdul que forma YourTalks són els "workers", s'anomenarà Carlit, aquest mòdul té una finalitat de processament d'informació, per no provocar esperes innecessàries en els clients. Serà l'encarregat de processar les presentacions i convertir-les en imatges separades, un cop finalitzat ho comunicarà a l'aplicació de real time que al seu moment ho comunicarà al client corresponent.

Aquesta estructura ens permet, una gran modularitat per construir aplicacions entorn d'ella. Per exemple, si es vol implementar l'aplicació per a mòbil, la API ja està feta només cal que l'app mòbil consumeixi l'API REST. Si es vol donar accés a alguna de les funcions concretes de la API, el cost d'implementació serà més reduït i senzill, ja que només afect a l'API. Aquesta separació ens permetrà escalar cada servei segons la demanda.

Es vol utilitzar en la mesura que sigui possible software OpenSource per a la seva gran comunitat i reduït cost, per tant, es requereix un bona recerca de tecnologies.

4. Planificació.

En aquesta fase s'estableixen els objectius i s'escullen els procediments més apropiats per a l'assoliment de cada un d'ells abans d'emprendre una acció. La planificació ens ajudarà a obtenir i repartir els recursos tan materials com humans de què es disposen per aconseguir els objectius. Així també ens ajuda a controlar l'assoliment dels objectius, fixar prioritats i tractar possibles problemes en el transcurs del projecte.

4.1 Kanban board.

La planificació d'sprints s'ha fet pensant en els objectius bàsics i l'abast que se li vol donar al software en un inici. Aquests sprints es refereixen en el que s'havia pensat que es tindria acabat per el dia de la finalització de l'sprint. Cada sprint s'ha definit en un periode màxim de 2 setmanes. La planificació del projecte es va fer entre les dates 24/03/2014 i 02/06/2014, això formen un total de 5 sprints a seguir. A cada sprint se li assignat un valor de risc (% of Error) que vol dir, la desviació possible que pot tenir l'implementació. Cada sprint englova un conjunt de tasques que han estat més detallades al JIRA, en els sprints només s'ha posat un títol i una descripció general del contingut de l'sprint. La taula 1 mostra els sprints planificats per a la realització del projecte.

Number	Dates	Who?	Title	% of Error	Description
#1	24/03/2014 - 07/04/2014	Francesc	User Dashboard	30	Create all the interface for the User Dashboard. CRUD + Sign In
		Adrià	Puppet (Servers)	30	Setup server infrastructure with automate system and investigate about high scalability
#2	24/03/2014 - 07/04/2014	Francesc	Admin Dashboard	20	Create all the interface for the Admin Dashboard, basically statdistics (and some CRUDs)
		Adrià	API REST	20	Develop API and slideshows workers
#3	24/03/2014 - 07/04/2014	Francesc	Room (Chat)	5	Hability to create a Room, invite people and to chat with them
		Adrià	Puppet (Servers)	30	Setup server infrastructure
#4	24/03/2014 - 07/04/2014	Francesc	Room (Slideshows)	5	Hability to add a slideshow to the room and to pass the slides of it
		Adrià	API REST	10	Develop`API
#5	24/03/2014 - 07/04/2014	Francesc	Room (WebRTC)	50	Well, WebRTC
		Adrià	API REST + PUPPET	10	Fix bugs and others things
Total:				210%	

Taula 1: Definició Sprints

4.2 Planificació.

En la taula 2 podem veure la planificació del projecte amb totes les fases.

Tasca	Subtasca	Data inici	Data fi
Avantprojecte		13/01/2014	31/01/2014
	Estudi del projecte	13/01/2014	25/01/2014
	Entrega documentació	25/01/2014	31/01/2014
Análisis i diseny		01/02/2013	
	Investigació de les tecnologies	01/02/2014	01/03/2014
	Definició de funcionalitats	01/07/2014	07/03/2014
	Disseny de prototipus	07/03/2014	
Implementació		24/03/2014	18/05/2014
	Implementació	24/03/2014	18/05/2014
	Proves	19/05/2014	06/06/2014
Memòria i presentació		05/05/2014	19/05/2014
	Memòria comuna	01/05/2014	19/05/2014
	Memòria individual	24/03/2014	18/05/2014
	Presentació	02/06/2014	13/06/2014
Entrega documentació final		02/06/2014	04/06/2014

Taula 2: Planificació

5. Estudi teòric previ

L'estudi previ el dividirem en les eines utilitzades, aquelles que han necessitat un estudi intensiu per poder-les implementar i utilitzar correctament. Aquestes eines estan totes implementades en JavaScript, i són les següents: BackboneJS, UnderscoreJS, JQuery, MarionetteJS, Mustache, NodeJS, Yeoman, Grunt, Bower, Soket.io i Redis. La majoria d'aquestes eines que hem llistat eren desconegudes per nosaltres abans de començar el projecte, ha calgut un gran esforç per poder-les utilitzar de la manera correcte i més eficient.

5.1. JavaScript

JavaScript és un llenguatge de programació d'scripting, dinàmic (té un tipatge dinàmic, a l'inrevés que JAVA que s'han de declarar els tipus) i asíncron. L'asincronia de JavaScript és el que el fa un llenguatge únic, i que esta present en pocs llenguatges (que jo conegui). Entendre aquesta asincronia, a l'inici, pot ser molt difícil, sobretot venint d'un llenguatge seqüencial (JAVA, Ruby, Perl, Phyton i d'altres). El funcionament bàsic és el d'un procès continuu que sempre esta escoltant, quan rep un event (un 'click' en el DOM) o es crida una funció, l'acció que s'ha de desenvolupar es processa a part, sense bloquejar el primer procés, no espera per la resposta de la funció, si necessita executar alguna cosa quan aquella funció hagi acabat hi assignarem una funció de "callback" per recuperar el resultat. Aquest funcionmanet s'anomena "event-drived", ja que tot funciona amb events.
A la Figura 1 es pot veure un exemple senzill s'entendrà millor, executaré el mateix codi amb dos llenguatges diferents, Ruby (síncron, no event-drived) i JavaScript (asíncron i event-drived). El codi es molt simple, dues funcions/mètodes que fan dormir el procés un total de 3 segons (primer 2 i després 1) un cop finalitzat diuen que ha acabat.

```ruby
def sleep_time time
  sleep time
  puts "Here I'm!, time passed: #{time}"
end

sleep_time 2
sleep_time 1
puts "Finish"
```

```javascript
var sleepFunction = function(time) {
  setTimeout(function() {
    console.log("Here I'm!, time passed:", time);
  },time);
}
sleepFunction(2000);
sleepFunction(1000);
console.log("Finish")
```

Figura 1: Exemple dormir

El codi de l'esquerra és de Ruby i el de la dreta és de JavaScript, anem a mirar quins són els resultats en executar els scripts, Figura 2.

```
~/repos/sandbox/node $> ruby sleep.rb
Here I'm!, time passed: 2
Here I'm!, time passed: 1
Finish
~/repos/sandbox/node $> node sleep.js
Finish
Here I'm!, time passed: 1000
Here I'm!, time passed: 2000
~/repos/sandbox/node $>
```

Figura 2: Exemple log

En el resultat es veu clarament la diferència de la qual parlàvem. Si és volgués aconseguir que a JavaScript l'ordre de les funcions for el correcte, l'esperat en un procés (2,1 i Finish) es tindria que posar una cadena de callbacks, Figura 3, per aconseguir-ho i retorcar la funció:

```javascript
var sleepFunction = function(time, callback) {
  setTimeout(function() {
    console.log("Here I'm!, time passed:", time);
    callback();
  },time);
}

var finish = function() {
  console.log("Finish")
}

sleepFunction(2000, function() {
  sleepFunction(1000, function() {
    finish();
  })
})
```

Figura 3: Callback

I el resultat que obtenim llavors és l'esperat, Figura 4:

```
~/repos/sandbox/node $> node callback.js
Here I'm!, time passed: 2000
Here I'm!, time passed: 1000
Finish
```

Figura 4: Resutat seqüencial

Un cop s'entén com funcionen els callbacks, events i funcions en JavaScript, la resta es semblant a altres llenguatges. JavaScript també disposa d'herència, s'anomenen "prototype chain", és un terme complex que no explicarem, ja que entenent els events ja s'entén la majoria.

JavaScript és conegut en web, ja que la seva implementació permet programar lògica en el cantó del Client (Navegador). Ara mateix és l'únic llenguatge de programació suportat per tots els navegadors, tot i que algunes funcionalitats més noves no estan disponibles en tots ells. Tal com explicarem amb les altres tecnologies, tota l'aplicació de Frontend esta feta amb JavaScript, tant per la part del Client (Backbone Marionette) com per la part del Servidor (NodeJS).

5.2. BackboneJS

BackboneJS és una llibreria de JavaScript molt bàsica i modelable que t'ajuda a estructurar aplicacions web en la part del Client, esta desenvolupada amb JavaScript. BackoneJS esta centrat a fer petites aplicacions de JavaScript, SPA (Single Page Aplication), és una llibreria petita però potent que fa que desenvolupar aquestes petites aplicacions sigui molt fàcil. Funciona com a punt de connexió entre l'API REST i el Client, mostrant de manera fàcil les respostes JSON del servidor a la Vista d'una manera estructurada, fins hi tot es podria fer una SPA que no requerís d'un servidor per guardar les dades, amb l'estructura de Models que presenta BackboneJS i les BBDD dels

navegadors, evidentment una implementació així seria poc útil. Ofereix una estructura de MV*, es pot considerar MVP (Model, Vista, Presentador). Esta pensada i estructurada per funcionar amb un Backend (API) amb una estructura REST.

Hi han alternatives a BackboneJS, la més coneguda i puntera és AngularJS, ja que esta suportada per un equip de Google. AngularJS és un framework de JavaScript que esta molt estructurat i pautat, amb normes, tot el contrari de Backbone, que no te normes ni estructura bàsica ni obligatòria. Un altre framework és EmberJS, creat per un desenvolupador del Core de Rails. Tots els frameworks de JavaScript són diferents però solucionen els mateixos problemes, alguns, mes fàcilment que d'altres, tot depèn de com et sentis més agust. Nosaltres personalment preferim la llibertat i no tant l'estructura obligatòria, BackboneJS.

Els Models en BackboneJS són el vincle directe amb l'API, són la representació dels JSON (JavaScript Object Notation) que es reben de l'API. Òbviament aquesta representació no cal que sigui directe, no cal crear un Model per cada JSON del servidor, només cal guardar la informació que sigui necessària per no sobrecarregar el Client amb informació innecessària, cal pensar que com que l'aplicació l'executa el navegador, tots aquests processos d'informació els fa la màquina del Client, el seu ordinador/mòbil i si són molt costosos, pot ser que el navegador els bloquegi. Internament BackboneJS simplement fa una simple petició AJAX (Asynchronous Javascript And Xml) al servidor, a una URL especificada al model.

A part dels models, BackboneJS ofereix una ajuda bàsica per tractar-los, les Collections. Les Collections no són res més que un objecte que inclou un llistat de Models. Amb moltes opcions addicionals per tractar-los (ordenar, buscar, eliminar, filtrar …)

Les Vistes són petits Scripts de HTML, en un format especial (en aquest cas hem triat un anomenat MustacheJS). Les Vistes poden ser representades de maneres diferents: poden ser directament adherides al DOM mitjançant <script type='text/template' id='id-template'></script>, poden ser descarregades en obrir-se la pagina directament des del servidor, ja compilades en JavaScript o poden demana-se al servidor la primera vegada que es necessiti i guardar-la al navegador (cache) per les següents peticions. A part del terme

general d'una Vista com a element del DOM, una vista pot ser una representació d'un Model, això el que permet és tenir el DOM separat per Models i actualitzar cada part del DOM al actualitzar-se un Model sense canviar tot el DOM sencer. Les Vistes també són les encarregades de capturar totes les accions que es produeixen en elles i, si es el cas, delegar el tractament de l'acció a una capa mes amunt.

Tot i que aquestes són de les ajudes més destacades que ofereix BackboneJS, també n'ofereix d'altres molt importants:

- Rutes: ofereix un tractament de rutes per saber quina acció s'ha d'executar en entrar una ruta (cal tenir en compte que les Rutes de JavaScript són amb '#' davant de tot, ja que sinó el navegador entén que és una ruta a un servidor, ruta absoluta). Un exemple de ruta seria el següent: pre.yourtalks.com/#users/sign_in, en tenir el #, el navegador no fa res i BackboneJS entra en acció i la processa i executa l'acció (sign_in) de l'usuari.
- Historial: per poder-te moure endavant i enrere en l'històric,
- Events: són events interns de l'aplicació, JavaScript al se asíncron, el seu funcionament és bàsicament en events (es defineix un escoltador que s'acciona al rebre el seu event, event-drived).

Per funcionar BackboneJS requereix obligatòriament de UnderscoreJS i JQuery.

5.3. UnderscoreJS

UnderscoreJS és una llibreria de JavaScript que ofereix un total de 80 funcionalitats/mètodes que t'ajuden a ser més productiu (estalviar-te línies de codi repetides). Aquestes funcionalitats estan implementades en molts altres llenguatges directament per defecte, però no es el cas de JavaScript, així que necessita l'ajuda externa d'UnderscoreJS per oferir aquestes funcionalitats. Uns exemples serien el tractament de col·leccions (Array), poder ordenar-les, filtrar-les, fer un map i un reduce sobre l'Array o fins hi tot iterar-les d'una manera senzilla. BackboneJS utilitza aquestes funcions d'UnderscoreJS en les seves Collections (conjunt de Models), les inclueix directament en el Core de BackboneJS.

5.3 JQuery

JQuery és una llibreria de JavaScript que et permet manipular elements del DOM d'una manera transparent i senzilla. No només ofereix una manipulació del DOM sinó que també ofereix un conjunt de funcionalitat per fer efectes d'una manera molt bàsica. És una llibreria molt utilitzada en el món de la Web. Com a tal, molta gent aporta llibreries que funcionen per sobre de JQuery i que permeten la utilització d'elements més complexos de JavaScript (Modals, Notificacions, Calendaris …). BackboneJS és una de les llibreries que necessita que JQuery estigui incluit en l'aplicació, l'utilitza per les Vistes i l'interacció que tenen amb el DOM.

5.4. MarionetteJS

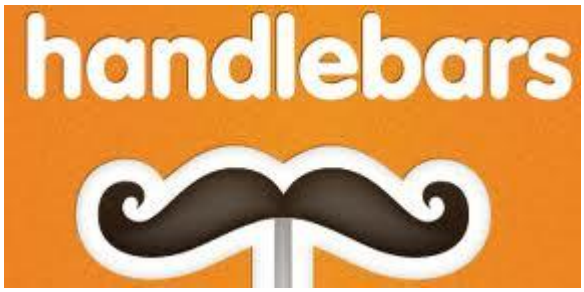
MarinetteJS és una llibreria que converteix BackboneJS en un framework. Una llibreria és un conjunt de mètodes i funcions que pots utilitzar, per altra banda un framework te una estructura de disseny abstracta que s'ha d'entendre per utilitzar-lo, "una llibreria l'utilitzes, un framework t'utilitza".

El seu objectiu es simplificar la construcció de grans aplicacions en Backbone, oferint escalabilitat i un conjunt de patrons de disseny. MarionetteJS afegeix un enorme quantitat d'ajudes que si s'haguessin d'implementar en BackboneJS es tardaria moltíssim.

Converteix el MV* de BackboneJS en un MVC, ja que MarionetteJS sí que incorpora Controladors. Els Controllers que te Marionette engloben algunes funcions bàsiques que ajuden a despreocupar-se d'una part de la lògica de les vistes, una lògica que en BackboneJS es ha de tenir present sempre. Aquesta lògica, entre d'altres, és la renderització d'una vista inicialitzada, directament en una regió, i si ja hi ha alguna altra vista a la regió, la tanca. Un altre element seria tractar els events que es lliguen a les Vistes,

per ell sol el Controller un cop tancat deslliga la Vista dels seus events, en BackboneJS s'ha de recordar de fer-ho manualment.
Permet modulitzar l'aplicació, ja que per ell mateix, Marionette, inclueix Moduls. Faciliza molt el tractament de les Vistes, ja que incorpora diferents tipus de Vistes (ItemView, CollectionView, CompositeView i Layout) amb uns rols i funcionalitats específiques. Un ItemView es el tipus de View més senzilla, no te cap característica especial, una CollectionView és un tipus de vista que no pot tenir cap template definit i s'utilitza per representar col·leccions de ItemViews, normalment s'inicialitza amb una Collection i ell sol itera per la Collection i renderitza l'ItemView per cada Model de la Collection, la CompositeView es igual que una CollectionView però amb la diferència que pot tenir un template definit, el Layout és un tipus de View especial que s'utilitza per dividir el DOM en regions, per poder renderitzar les vistes de manera ordenada, com a característiques bàsiques defineix les Reginos que el seu template conte i que serveixen per poder dividir el DOM i renderitzar altres Vistes dintre

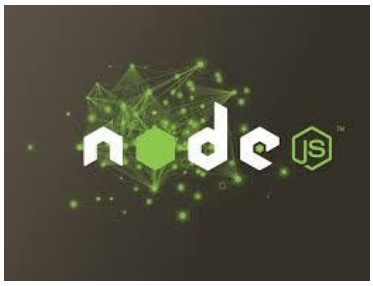
5.5. Handlebars/Mustache



Mustache és el tipus d'enginyeria de templates que utilitzem en el cantó del Client. Simplement són templates que es compilen en JavaScript i que contenen una sintaxi interpretada interna per serialitzar la informació que vols mostra, en el cas de Mustache la sintaxi és {{}}. Per sobre de Mustache he afegit Handlebars que augmenta les funcionalitats que ofereix Mustache. Com totes les altres enginyeries de templates que hi ha disponibles, la seva funcionalitat és la de representar templates definits en JavaScript en el Client. Les enginyeries de templates és diferències segons quina sigui la funcionalitat que se l'hi vol atorgar. Handlebars proporciona una gran facilitat d'implementació i esta preparat per poder executar una lògica complexa en les vistes (Mustache per altra banda no), aquestes lògiques normalment són simplement càlculs i comprovacions, també permet definir "helpers" (mètodes que pots cridar directament des de la vista). Un altre de les opcions, que preferim, és Jade. Jade, com Handlebars és una altre enginy de template que

permet escriure la mateixa lògica que Handlebars, però l'HTML s'escriu d'una manera mes simple, com HAML. Permet que l'escriptura de l'HTML sigui molt més fàcil i ràpida, i un cop t'acostumes a la sintaxi la trobes molt més fàcil de llegir.
Vam elegir Handlebars, ja que la configuració ens va semblar més fàcil que Jade, i no pdiem permetre'ns el luxe de malgastar més temps en una configuració així, la configuració am Handlebars més tard ens va portar algun mal de cap però res de greu.

5.6. NodeJS



NodeJS és una plataforma per construir servidors web en JavaScript. NodeJS és perfecte per aplicacions que requereixen implementacions de temps real, ja que la manera en què esta construïda (event-driven, non-blocking I/O) permet distingir-lo d'altres opcions que serien menys optimes i consumirien més recursos. Es una tecnologia enfocada a un producte concret, serveix per aplicacions que necessiten un I/O molt gran i sense espera, a causa de la seva naturalesa el fa el més útil en aquest sentit, evidentment es pot utilitzar per qualsevol altre tema, però va molt enfocada en aquest tema concret. NodeJS també esta construït en JavaScript i aixo ajuda a donar estructura i coherència de tota l'aplicació, així tan el servidor com el Client tenen un llenguatge comú. NodeJS també inclueix un gestor de paquets que es diu npm (Node Package Manager) que permet instal·lar llibreries i crear un manifest amb les dependències de l'aplicació, anomenat package.js.

5.7. Yeoman

Yeoman és una llibreria que vol resoldre un problema comú en les aplicacions de JavaScript, l'organització i estructuració. Yeaoman permet estructurar les nostres aplicacions de JavaScript, et monte l'esquelet de l'aplicació i instal·la les llibreries de Grunt, Bower i Yo. En crear una nova aplicació utilitzant Yeoamn crea una configuració bàsica:

Genera un Gruntfile.js (amb unes taskes perdefinides), instal·la algunes llibreries amb Bower, crea el servidor de NodeJS i muntar tota l'estructura de carpetes que normalment s'utilitza.

"Yo" no l'hem utilitzat en aquest projecte, la seva funció es de generador de codi, utilitzant unes llibreries, simplement li dius que et generi un Model (BackboneJS) i et crea tota l'estructura que ell creu que necessitaràs per utilitzar el Model, el que sempre crees al fer un Model.

5.8. Grunt

GruntJS és una llibreria que serveix bàsicament per automatitzar taskes. És un fitxer de configuració on determines cada taska quina acció realitza, requereix d'altres llibreries per realitzar les funcions més complexes, per exemple: usemin és una llibreria/taska que el que et permet és compilar, concatenar i minificar els fitxers de JavaScript o CSS que tens definits en el DOM, mitjançant una sintaxi pròpia després reemplaçar tots els fitxers pel resultat de les operacions anteriors, això és útil, ja que així el pes de la descàrrega de JavaScript o CSS del servidor en entrar a la pàgina es redueix considerablement. Les altres taskes que pot desenvolupar són: engegar el servidor de NodeJS, mirar si s'han fet canvis en algun fitxer i renderitzar la web un altre cop, prepara l'aplicació per ser pujada a producció (utilitzant el usemin) i d'altres.

5.9. Bower

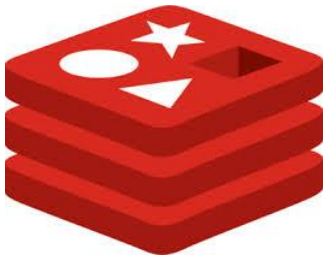
Bower és un gestor de paquets com npm. La diferencia entre npm i Bower és que npm és per llibreries que s'utilitzen en el servidor (socket.io, express …) i Bower és per llibreries que s'utilitzen en el Client (BackboneJS, MarionetteJS, JQuery …).

5.10. Socket.oi

Socket.io és una llibreria que permet la comunicació en temps real entre Clients d'una manera àgil i fàcil d'implementar. Socket.io s'adapta a les necessitats del navegador, si el navegador no suporta WebSocket (que es la primera que intenta implementar) utilitza altres com: Adobe Flash Socket, AJAX long polling o JSONP polling.

El funcionament més bàsic es: des del Client emet informació en el seu socket, el servidor rep que el socket vol emetre, i llavors el servidor (NodeJS) decideix a quin socket vol enviar la informació. També permet crear sales i fer un broadcast a tots els sockets de la sala o a alguns en concret (aquesta seria la implementació bàsica d'un xat).

5.11. Redis



Redis és una Base de Dades no relaciona, es una Base de Dades key/value avançada, ja que les key poden ser de diferents tipus: string, hashes, list, sets and stored sets. És una Base de Dades que té molt potencial, la seva simplicitat fa que les operacions siguin molt ràpides.

6. Explicació Disseny

La part del desenvolupament que em pertoca en el projecte és la del Frontend i la del servidor de RT (Real Time). La unió de les parts que em corresponen serveixen per oferir a l'usuari una interfície i una mobilitat per l'aplicació. El Frontend, dividit entre un servidor de NodeJS, per servir l'aplicació amb Backbone Marionette i l'aplicació anomenada anteriorment. Quan una persona entra a la pàgina web, demana al servidor pels assets, aquests assests són l'aplicació de Backbone Marionette, un cop el Client rep la resposta, tota la resta d'accions només les controla l'aplicació de JavaScript, el servidor de NodeJS no s'utilitza més. Per altra banda el servidor de Real Time serveix per poder comunicar dos o més clients, per ara només s'utilitza per poder utilitzar el xat i per poder passar les presentacions a les sales.

6.1 Treball a realitzar

Ara toca explica com, i que, hem implementat, quines funcionalitats hem creat en el Client utilitzant totes les llibreries i distribució.

6.1.1 Landing

El primer que hem creat ha sigut la Landing, la pàgina principal del projecte, en aquesta no hi ha cap tipus de lògica per al Client, s'ha creat per fer la funció de Landing. Esta composta d'un Carusel i un seguit d'informació més extensa sobre el software, a sota. Aquesta pàgina, mostrà la informació bàsica de qui som, que fem i totes les preguntes que pugui tenir un nouvingut per saber si l'interessa o no el software. La implementació es "senzilla", ja que no te cap tipus de lògica, només el llistat és una crida al servidor per demanar quin és el contingut i quines són les frases, imatges i textos que ha de tenir cada un d'ells. Es pot veure fàcilment que és una Collection de Models, tan el Carousel com el llistat que té associat i la barra de navegació del Carousel.

6.1.2 Footer

El Footer és la vista més simple de totes, ja que nomes és un template per donar la imatge i perspectiva d'un futur Footer. És rederitza un ItemView amb tot el contingut del Footer, només HTML pla.

6.1.3 Header

El Header ja té una mica més de lògica, el Header varia segons si l'usuari ha entra entrat o no a la plataforma, si ho està, mostra l'opció de sortir (Sign Out) i l'opció d'anar al Dashboard del usuari per veure les seves Presentacions, editar el seu perfil i altres opcions. Si l'usuari no ha entrat a la plataforma mostra l'opció de fer Sign In o de Sign Up cada una d'aquestes opcions interactua amb el Servidor. El Dropdown del Sign In envia al servidor una petició amb l'email i el password que l'usuari hi ha introduit, si la combinació és correcta el porta cap al seu Dashboard, si és incorrecta l'hi mostra una notificació dient que la combinació és incorrecte.

6.1.4 Sign Up

Aquesta pantalla té més lògica, ja que permet a l'usuari crear una conta a YourTalks. Aquesta conta té unes validacions que ha de passar prèviament, algunes d'elles es podrien comprovar en el client per estalviar crides en el servidor (Ara mateix no ho tenim validat d'aquesta manera). Aquestes validacions és posarien en el Model en concret que s'utilitza per crear la conta, el de User. Un cop s'envia la petició al servidor, poden passar dues coses:

- Success: Si l'usuari es crea correctament, se'l porta automàticament al seu Dashboard.
- Error: Si la creació de l'usuari és incorrecte, el Servidor respon amb un JSON amb els errors que s'han trobat en la creació de l'usuari, aquests errors es computen i es mostren a l'usuari perquè sàpiga que és el que ha fallat.

6.1.5 Dashboard

Aquí l'Usuari pot consultar la informació de la seva conta, en el menú lateral pot navegar per cada opció. L'opció que esta implementada és la de Presentacions, on pot veure totes les seves presentacions i crear-ne una de nova. El llistat de presentacions és una crida al Servidor per obtenir-les, la creació també és una petició al servidor, com a la creació de l'Usuari, si no es pot crear a causa d'errors, es mostren a l'usuari per informar-lo d'ells.

6.1.6 Room

Les sales es creen en el botó que es troba en el Header, aquest botó mostra un dialog amb un formulari per introduir el nom de la sala i quina presentació es vol mostrar. En finalitzar la creació, es passa a la Vista de la Room. Aquesta vista esta modulitzada utilitzant gridster (gridster.js), cada modul és una de les eines de la sala, ara mateix no es pot elegir quines es volen, directament es mostren la de xat i la de presentacions. Quan es crea la sala, el client fa una crida al servidor de Real Time per dir que s'ha unit a una sala amb el nom especificat, el xat de la sala també utilitza el servidor de Real Time per enviar els missatges, els envia a tots els usuaris que s'han unit a la sala. Les presentacions utilitzen el servidor de Real Time per poder passar la presentació en el mateix moment a tots els clients que estan dintre la Room.

6.1.7 Real Time

El servidor de Real Time és una implementació de NodeJS i socket.io per permetre totes les interaccions que es vulguin entre els Clients directament. Aquest servidor consta d'una petita implementació en socket.io per tractar els events que estan per ara definits a les Vistes, i una implementació de Redis, en Pub/Sub per escoltar els events que l'API vol presentar als Clients.

6.2 Frontend

L'estructura principat esta feta amb YO (l'únic moment en què s'utilitza en tota l'aplicació), aquesta estructura no s'ha de canviar en cap moment. L'estructura divideix l'aplicació en dos parts: la part del **Servidor/Desenvolupament** que distribuirà el index.html amb tot el JavaScript, CSS i taskes de Grunt, i la part de l'aplicació de JavaScript del **Client** (aquesta part YO no la defineix, simplement crea la carpeta scripts/ que es on ha d'anar tot i allà tu mateix montes l'estructura que vulguis). Tot aixo ve degut en què a JavaScript l'hi falta una manera d'estructurar-se, l'estructuració sempre l'ha de fer un mateix i, en aplicacions que són molt grans, aquest és un punt bàsic per no tenir un codi desordenat, no escalable ni mantenible. Aquest problema resideix en el mateix JavaScript, però YO t'ajuda en aquest sentit. YO el que et genera es una estructura de directoris essencial que et fa partir des d'un punt que no és el 0. És molt útil per la gent que no té molt clar com començar una aplicació (com és el nostre cas), amb aquesta estructuració que fa YO per defecte et permet saber on és millor posar el CSS, les imatges i els scripts, ell ja t'els inclueix en el index.html que crea la primera vegada, un cop ja tens l'estructura bàsica, ja es pot començar, excepte que es vulgui canviar aquesta configuració esclar.

6.2.1 Servidor/Desenvolupament

La part que consideraríem de Servidor (NodeJS) i Desenvolupmanet (taskes de Grunt) és molt reduïda, en comparació amb l'aplicació de JavaScript de Backbone Marionette, però alhora molt complexa i potent. El servidor és un fitxer de JavaScript que executa un servidor de NodeJS sense cap complexitat, molt simple. Per altra banda en la part de Desenvolupament tenim el Grunt amb el Gruntfile.js amb totes les seves taskes, Figura 5.

Figura 5: Estrucutura carpetes Servidor/Desenvolupament

En Grunt el que es fa és definir taskes que engloben un conjunt de subtaskes, les taskes que hem definit són les següents: serve (engega el servidor en local) i build (prepara el repositori per pujar-lo a producció), aquestes són les que engloben altres de petites. Tot això, es fa per ajudar al desenvolupament i la preparació de l'aplicació per pujar-la a un servidor.

```
grunt.registerTask('serve', function (target) {
    if (target === 'dist') {
        return grunt.task.run(['build', 'connect:dist:keepalive']);
    }

    grunt.task.run([
        'clean:server',
        'concurrent:server',
        'autoprefixer',
        'handlebars',
        'connect:livereload',
        'watch'
    ]);
});
```

Figura 6: Taks serve

La taska de serve, Figura 6, al executar-se executa 6 subtaskes en ordre.

- ‘clean:serve’: El que permet es buidar la carpeta .tmp/ (utilitzada per guardar fitxers temporals, com els que es tenen que convertit abans de compilar, més endavant s’explica).
- ‘concurrent:server’: En el mateix moment conte 2 tasque més, ‘compass:server’ i ‘copy:styles’ converteix el .scss en .css els nous fitxers .css els emmagatzema al .tmp/.
- ‘autoprefix’: El seu objectiu es mira si el navegador pot o no utilitzar els CSS i els posa el prefixi necessari per funcionar (si és requerit).
- ‘handlebars’: Aquesta es la que s’encarrega d’agafar tots els fitxers dels directoris que l’hi marco, que acabin amb .hbs i compilar-los en un sol fitxer que es dirà: compiled-templates.js, aquest fitxer després el tenim que importat al index.html, aixi estaran les vistes descarregades al Client, anteriorment ho teniem desenvolupat de tal manera que cada vegada si el Client requeria una vista, se la descarregava del servidor, i la guradava en cache, això efecte en el rendiment del Client, baixar-se un sol fitxer amb totes les vistes ja preparades és molt més eficient.
- ‘connect:livereload’: Permet recarregar la pestanya local del navegador quan detecti un canvi, els canvis es detecten amb la sisena taska ‘watch’, que se l’hi marquen uns fitxers a observar, i quan canvien recarrega el navegador. Evidentment aquesta taska nomes serveis en el desenvolupament local.

Aquestes són les 6 taskes que hem configurat per executar en local l’aplicació. Totes aquestes taskes que hem explicat no estan optimitzades. Deixem per més endavant aquesta optimització de les taskes.

L'altre taska de Grunt que tenim configurada és la de build. Aquesta tasca s’encarrega de crear una carpeta anomenada dist/ en la que es troba l’aplicació llesta per pujar a producció. Quan s’executa la taka de build, crida a un seguit de subtaskes (13) que preparen l’aplicació per poder ser pujada a producció. D’aquestes 13 només explicaré les més interessants que són les de **usemin**.

La taska de **usemin** és una de les més potents, ja que prepara els JavaScripts per pujar-los a

producció juntament amb el index.html. Basicament aquesta taska s'encarrega de llegir en el index.html buscant blocs que ell pugui identificar, una sintaxi concreta, Figura 7. Cada un d'aquests blocs significa quelcom per a usemin:

```
<!-- build:js scripts/vendor.js -->
<script src="bower_components/jquery/jquery.min.js"></script>
<script src="bower_components/jquery-cookie/jquery.cookie.js"></script>
<script src="bower_components/gridster/dist/jquery.gridster.min.js"/></script>
<script src="bower_components/noty/js/noty/packaged/jquery.noty.packaged.min.js"></script>
<script src="bower_components/underscore/underscore-min.js"></script>
<script src="bower_components/backbone/backbone-min.js"></script>
<script src="bower_components/marionette/lib/backbone.marionette.min.js"></script>
<script src="bower_components/backbone.syphon/lib/backbone.syphon.min.js"></script>
<script src="bower_components/cocktail/Cocktail-0.5.7.min.js"></script>
<script src="bower_components/Backbone.Chooser/build/backbone-chooser-min.js"></script>
<script src="bower_components/handlebars/handlebars.runtime.js"></script>
<script src="bower_components/handlebars/handlebars.js"></script>
<script src="bower_components/spinjs/spin.js"></script>
<script src="bower_components/spinjs/jquery.spin.js"></script>
<!-- endbuild -->
```

Figura 7: Blocs usemin

Agafa el contingut dels blocs i el trasspassa a .tmp/scripts/vendor.js, en aquest exemple concret. Alla el minifica (posa tot el JavaScript en una sola línia) i com a ultim, en el index.html, en el lloc on teníem el block hi posa la ruta al fitxer nou. Tota aquesta màgia interna ens és molt util, ja que els fitxers quan estan minificats són menys pesats i la descàrrega de tota l'aplicació és més ràpida. Usemin fa el mateix amb els CSS. El el index.html final només queden 4 o 5 lineas, ja que tots els scripts d'importació s'han remplaçat per un o dos de sols, Figura 8.

```
<script src=scripts/7021afed.vendor.js></script>
<script src=http://pre.yourtalks.com/rtc/socket.io.js></script>
<script src=scripts/aaf74903.plugins.js></script>
<script src=scripts/7369ca49.main.js></script>
```

Figura 8: Resultat usemin

La resta del HTML es genera dinàmicament amb l'aplicació de JavaScript (Backbone.Marionette), els scripts els hem agrupat en 4 de diferents, main.css(tots els css), vendor.js (llibreries externes instal·lades, JQuery), plugins.js (llibreries que s'utilitzen com ajuda constant, que ofereixen tan HTML com CSS com JS) i main.js (tota l'aplicació del Client en JavaScript).

6.2.2 Client

L'estructura d'una aplicació de Frontend (per defecte) pot ser molt diferent segons la llibreria de JavaScript que s'utilitzi, i la manera que la persona que programa, cregui que esta més ben organitzat. Per començar una aplicació de Frontend es compon de molts fitxers de JavaScript i un fitxer de HTML, com a estructura més bàsica, que inclou la resta de JavaScript de l'aplicació, aquest fitxer de HTML es el que inicia l'aplicació de JavaScript, un cop inicialitzada, l'aplicació pren el control.
Tot seguit explicarem l'estructura de directoris del client, ja que ajudara a entendre l'aplicació i la distribució. És podrà contemplar que tot segueix una estructura bàsica, un patró, la modulabilitat de Marionette. Cada element de tota l'arquitectura és un Modul, un module que segueix una estructura concreta (s'explicarà més endavant).
Després d'explicar l'estructura de directoris explicarem els Moduls comentats anteriorment i com es comuniquen entre ells.

6.1.2.1 Estructura de Directoris

Explicarem l'estructura de Directoris partint de l'arrel d'on és troba el index.html.

6.1.2.1.1 App

Dintre la carpeta app/ de l'arrel també trobem el favicon.ico, el server.js (servidor de Node), del directori styles/ (per eld CSS, no entraré a explicar res d'ell), el directori de images/ (per les imatges), el directori de bower_components/ (que és on s'instal·len les llibreries utilitzant Bower, ja que com hem comentat formen part del Client) i com a ultim els scripts/, que és la part important, Figura 9.

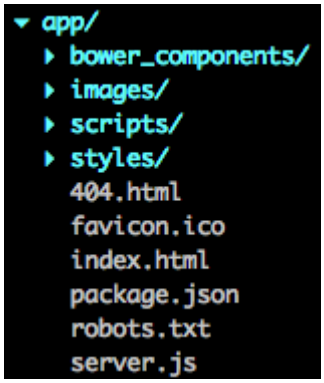


Figura 9: Arrel App

6.1.2.1.2 Scripts

Dintre dels scripts/ és on és troba el núclid de l'aplicació, primer trobem 3 directoris mes:

- backbone: que és on és troben tots els fitxers de JavaScript que forment part directament de l'aplicació.
- config: per configuracions i sobreescriure llibreries que no tenen a veure directament amb l'aplicació.
- vendor: que és per llibreries/scripts de terceres persones (vindrien a ser les llibreries instal·lades amb Bower, però aquestes les deixo separades, utilitzem aquest directori per diferents tipus de llibreries o scripts que no es poden descarregar).

A part dels 3 directoris també trobem els compiled-templates.js, Figura 10, que són totes les vistes que forment part de l'aplicació compilades

Figura 10: Estructura scripts

6.1.2.1.3 Backbone

Tal com hem explicat, aquí és on comença tota la part important de l'aplicació. Esta dividit en 3 carpetes i un fitxer de JavaScript (Figura 11):

- apps: Aquesta carpeta conte cada SubAplicació/Modul/SubModul (s'explicara que és en la part de definició dels Moduls) que conte l'aplicació.
- entities: Aquesta carpeta és on s'emmagatzemen tots i cada un dels Models i Collections que utilitzarà l'aplicació.
- lib: És troben totes les llibreries escrites per nosaltres, son ajudes o "overrides" d'alguns components de l'aplicació.

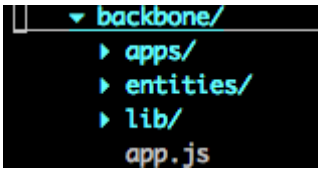


Figura 11: Backbone estructura

El fitxer de JavaScript que conte aquest directori és el que defineix l'aplicació, és el que inicialitza i dona nom al que serà l'aplicació de JavaScript. En el nostre cas es diu YTAneto (cada repositori de YourTalks te un nom clau, aquest en concret, Aneto) i també es solen posar les funcions que es criden en inicialitzar l'aplicació:

- Inicialitzar el Backbone.History per poder mantenir l'estat de l'aplicació (poder anar endavant i enrere).
- Definir algunes variables dintre l'aplicació, que han de ser compartides dintre d'ella (YTAneto.socektURL, YTAneot.rootRoute, YTAneto.hostname … etc)
- Alguns evens que han de ser tractats per la peça més alta de l'aplicació (els events els tractaran més endavant quan parli de la comunicació entre Moduls)
- Definició de les Regions principals.

Les regions és una manera que ha definit Marionet per dividir el DOM en petites porcions. Cada porció d'aquestes pot contenir un Modul amb una Vista, o una Modul amb un tipus

de Vista(Layout) que contè més regions i on poden anar més Moduls (Així és comença a entendre la distribució que ofereix respecte Backbone)

6.1.2.1.4 Entities

Si for per ordre tocaria explicar apps/ hem decidit deixar les apps/ per al final, ja que és el millor exemple per explicar els Moduls de Marionette.
En el directori de entities/, tal com hem dit anteriorment, és on és troben tots els Models i Collections de l'aplicació. Com podeu observar a la imatge (Figura 12), el directori esta compost per una serie de fitxers de JavaScript i un directori. El directori abstract/ conte un conjunt de Models i Collections que no són comunes, aquests Models no vénen donats per cap API REST d'un servidor, aquests Models són per abstreure lògica del codi (per això és diu abstract/), són Models que per al funcionament que te Backbone.Marionette va molt bé que vagin separats (En l'estructuració dels Moduls explicarem el perquè, té a veure amb un pseudo-patro/llei/norma que s'utilitza normalment en Marionette).

Figura 12: Estructura entities

Aquests models abstractes són:

- buttons: Els botons dels Formularis o qualsevol altre tipus de conjunt de botons.
- errors: Els codis d'erro que retorna l'API, així es pot comprovar que l'error 101 vol dir "Can't be blank" i es pot mostrar a l'Usuari.
- nav: La barra de navegació del Dashboard o qualsevol altre tipus de barra de navegació.

Més endavant, quan s'expliqui la comunicació entre Moduls, s'explicara l'estructura interna característica de tots els Models, ja que comparteixen la mateixa.

Els altres Models tots tenen relació amb l'API REST menys el de sessió, aquest es el que s'encarrega de mantenir la sessió del client, saber quin client esta actualment navegant i eliminar la sessió del client quan decideix sortir de la seva conta.

El funcionament que tenen els Models/Collections en Marionette és el mateix que ofereixen el BackboneJS, en aquesta part no té res a afegir. Per fer-vos una idea de quin aspecte te un Model adjuntem (Figura 13) una imatge on surt l'esquelet d'un Model. Aquest en concret pertany al user.js, és el que permet crear un usuari nou a l'API, podem veure que el model fa de punt d'unió entre el Servidor (API) i el Client.

```
YTAneto.module("Entities", function(Entities, YTAneto, Backbone, Marionette, $, _){

  Entities.User = Entities.Model.extend({
    urlRoot: "/v1/user",

    save: function(attrs, options) {
      options || (options = {});

      options.contentType = 'application/json';
      options.data = "{\"user\":" + JSON.stringify(attrs) + "}";

      YTAneto.Entities.Model.prototype.save.call(this, attrs, options);
    },
  })

  Entities.UsersCollection = Entities.Collection.extend({
    model: Entities.User
  })

  var API = {
    getNewUser: function() {
      return new Entities.User
    },

    getUser: function(id) {
    },

    getUsers: function() {
    }
  }

  YTAneto.reqres.setHandler("new:user:entity", function() {
    return API.getNewUser();
  })
})
```

Figura 13: Exemple Model i Collection, entities

6.1.2.1.5 Lib

Figura 14: Estructura Lib

A partir d'aquí ja començant les parts divertides de l'aplicació, tal com es pot veure en l'ultima figura (Figura 14) aquí hem decidit ja mostrar totes les carpetes i subcarpetes, ja que serà més fàcil d'explicar com esta tot dividit.

Primer de tot dintre de lib podem observar que l'hem dividit encara en 7 directoris més, després explicaré el contingut de cada un en concret:

- components: Els components són parts de l'aplicació que estan abstretes, ja que són complexes i molt utilitzades per l'aplicació, són serveis externs/llibreries. D'aquesta manera sempre que es vulgui es pot demanar el servei que ofereix el component.
- concerns: Els concerns són petits moduls que es poden importar/utilitzar en llocs concrets de l'aplicació i ofereixen un seguit de funcionalitats.
- controllers: En aquest directori en concret és on hem posat l' "override" del Controller que implementa Marionette per defecte. Els mètodes que sobreescrivim ho faig per fer més fàcil la implementació del codi, coses que aniriem repetint cada vegada en els Controllers les hem extret i sobreescrit a les que faria per defecte amb Marionette.
- entities: Aquí és on sobreescrivim els Models i Collections que venen per defecte amb Backbone. És el mateix principi explicat amb els Controllers.
- reginos: Aquí és on es posen les resgions customitzades. Tal com hem explicat anteriorment, les regions són un ajud que dona Marionette per distribuir el DOM i les vistes. Aquí pots posar-ne de customitzades perquè quan la Visat és rederitzi a la regió faci alguna cosa especial (Més endavant explicarem a què ens referim).
- utilities: Basicament és un conjunt d'utilitats (com el seu nom indica), són petits "overrides" que faciliten la implementació o que permeten fer alguns canvis fàcilment.
- views: Aquí és, com a les Entities i el Controller, on és sobreescriuen les vistes que ofereix Marionette per oferir una estructura en arbre, on podem establir que un tipus de vistes tindran algun atribut o mètode sempre per defecte.

Després d'aquesta breu introducció, explicarem els punts mes importants de cada apartat per entendre millor que contenen, amb exemples pràctics sempre es veu millor. Explicarem els que creiem que són més importants dels anteriors: Components, Regions i Utilities (alguns).

Components

Com es pot veure a Components hi tenim 4 components, cada un d'aquests desenvolupa un taca única i repetitiva, amb la seva abstracció permet que qualsevol altre Moduel de l'aplicació els requereixi per fer una funció específica.

El Component anomenat **Form** presenta una estructura molt característica que explicaré més endavant (amb els Moduls de Marionetter i la manera en què estructorem els components de l'aplicació i la seva comunicació), la seva funcionalitat és proporcionar ajuda en el moment de crear un formulari. El Form ofereix la vista, els tags de <form>, i només cal cridar-lo passant una vista que es la que conte els camps del formulari. El mateix formulari s'encarrega de capturar el events de 'submit' i 'cancel' i de processar l'acció que pertoqui al Servidor, per exemple crear un element.

El Component de **gridster** és la llibreria (Gridster.js) que ens permet fer que els elements de la sala siguin ordenables, movibles i redimensionables. La seva funcionalitat és cridar la funció de gridster.js que permet fer que els elements del DOM es converteixin els elements de gridster, aixo ho fa quan sap que s'han carregat totes les vistes que necessita la sala (aquesta va ser una solució simple i neta que vaig arrivar per no complicar molt el codi de dintre la sala).

El Component de **loading** és el que ens permet donar l'efecte que la pagina està carregant, s'utilitza en el Dashboard de l'usuari en les presentacions. Ofereix l'efecte visual de veure un spiner girant i ens fa entendre que el que hem demanat està tardant i que esperem. És un dels inconvenients de les aplicacions de JavaScript, si les peticions en AJAX són lentes, no hi ha cap informació cap a l'usuari, normalment si la pagina tarda a carregar el que tenim és una pàgina blanca i al costat del símbol de la tab del navegador un spiner, aquest es l'efecte que aconseguim amb això. Aquest component utilitza una llibreria anomenada spin.js que permet crear aquests spiners.

El Component **notify** el que fa es proporcionar-nos fàcilment una manera de mostrar notificacions a l'aplicació per a informar l'usuari de què ha passat en les accions

importants, per exemple si s'ha pogut crear la presentació, si ha pogut sortir del seu compte correctament, etc … Utilitza una llibreria anomenada noty.

Regions

A les Regions només hem necessitat crear-ne una d'especial, la del dialog_region.js. Aquesta Region, tal com indica el seu propi nom, és per facilitar l'us de dialogs/modals. El dialog és una molt bona eina de JavaScript per fer més dinàmica i entenedora alguna informació de la pàgina, també és molt útil com a petit formulari ràpid d'interacció. Per exemple si es vol crear una sala, apareix un dialog amb un petit formulari demanant el nom que tindrà la sala (més endavant també és demanaran els components que es vol que formin part d'ella). La seva principal funcionalitat és molt senzilla, simplement en renderitzar una vista, el que dius és que vols que sigui a la region de dialog, i la regió s'encarreg de fer que la vista aparegui en un dialog.

Utilities

L'últim dels directoris de lib que explicarem és el de Utilities. Consta de 6 fitxers, cada un amb una funcionalitat que aporta algun detall o utilitat a l'aplicació. La majoria són molt complexos a nivell de JavaScript, ja que serveixen per sobreescriure mètodes de Marionette o Backbone, però explicarem el de **socket.js**. Aquest petit Modul permet extreure la lògica del canal de dades que és socket.io, permet que si en algun moment es decideix canviar el socket.io per una altra llibreria, l'aplicació no s'enteraria de res, és un patró Façana. El que fa és rebre els events que envia socket.io, del servidor, i convertir-los en el canal de comunicació que utilitza Backbone.Marionette, i si es vol enviar un missatge per al socket, es converteix la informació que es vol enviar en la corresponent a la llibreria, en aquest cas socket.io, Figura 15.

Figura 15: Utilities socket.io

6.1.2.1.6 Apps

Figura 16: Estructura Apps

Si en la lib hem comentat que començaven les parts divertides, aquí comença el més emocionant, tot el core de l'aplicació, les apps. Cada un d'aquests directoris (Figura 15) representa una aplicació independent de la resta, cada una s'encarrega del seu rol en concret, el Footer no controlara si un usuari es pot crear o no (excepte que sigui el seu rol, poc provable). Com que el següent tema tractaré exactament com funcionen els Moduls de Marionette i la seva comunicació interna. Aquí explicarem l'estructura de

les apps i quin és disseny que comparteixen totes (tal com es pot comprovar totes tenen una estructura igual), no explicarem que fa cada aplicació en concret, simplement explicaré detalladament el funcionament d'una en concret, amb una sola es pot extrapola el disseny a totes les altres.

L'app que hem elegit per desglossar és una que no és molt complexa i que pot mostrar tota l'estructura bàsica de les altres apps. El users_sign_up. El rol d'aquesta app es mostra a l'usuari la pantalla de Sign Up quan entre la url de pre.yourtalks.com/#users/sign_up (recordem que el # és necessari per totes les url de JavaScript). Un cop la vista esta mostrada ha de permetre al nou usuari entrar les seves dades i crear l'usuari.

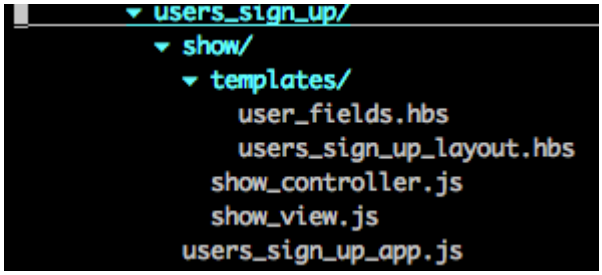


Figura 17: Estrucutra user_sign_up, Modular

Primer de tot observem (Figura 17) que a l'arrel trobem un directori (show) i un fitxer users_sign_up_app.js, el fitxer es el Modul principal de l'app, el que orquestra que es fa i el que inicialitza els Controllers.

```
YTAneto.module("UsersSignUpApp", function(UsersSignUpApp, YTAneto, Backbone, Marionette, $, _){
  var API;

  UsersSignUpApp.Router = Marionette.AppRouter.extend({
    appRoutes: {
      "users/sign_up" : "sign_up"
    }
  })

  API = {
    sign_up: function() {
      return new UsersSignUpApp.Show.Controller()
    }
  }

  YTAneto.addInitializer(function(){
    return new UsersSignUpApp.Router({
      controller: API
    })
  })

})
```

Figura 18: UserSignUp, App

El primer que trobem en fitxer (Figura 18) és la descripció de les rutes, amb aquesta descripció diem que si un usuari entre URL_ROOT + "#/users/sign_up" ha de fer l'acció sign_up. Com sap quina és l'acció que ha de fer i on esta? Al final del fitxer trobem un YTAneto.addInitializer, que el que fa és registrar al app principal (YTAneto creada en el app.js) una nova ruta escoltadora, amb un controlador, el que conte les accions que s'han de realitzar al accedir a la ruta, anomenat API. En el fitxer podem trobar aquesta API en el mig del fitxer, és un simple Objecte de JavaScript amb l'atribut sign_up, que és la funció que és crida en entrar a la ruta descrita. La funció sign_up el que fa és crear un nou Controller, el de show. El fitxer show_controller.js en el directori Show.

```
YTAneto.module("UsersSignUpApp.Show", function(Show, YTAneto, Backbone, Marionette, $, _){
  Show.Controller = YTAneto.Controllers.Application.extend({

    initialize: function() {
      var user        = YTAneto.request("new:user:entity")

      this.layout = this.getUsersSignUpLayout();
      this.listenTo(this.layout, "show", function() {
        this.formRegion(user);
      })
      this.show(this.layout);
    },

    getUsersSignUpLayout: function() {
      return new Show.UsersSignUpLayout;
    },

    getUserFields: function(user) {
      return new Show.UserFields({
        model: user
      })
    },

    formRegion: function(user) {
      var userFields  = this.getUserFields(user);
      var form        = YTAneto.request("form:wrapper", userFields)
      this.listenTo(userFields, "form:cancel",function() {
        YTAneto.vent.trigger("return:home")
      })
      this.listenTo(user, "created", function(user) {
        YTAneto.vent.trigger("session:create", user)
      })
      this.show(form, {region: this.layout.formRegion})
    }

  })
})
```

Figura 19: UserSignUp controller

Entenem perfectament que aquest exemple (**Error! Reference source not found.**) pot sembla molt confús si no s'està acostumat a l'estructura de JavaScript però només ens centrarem a explicar quin procediment segueix. Quan s'entra a la ruta el que fa és crear aquest nou Controller, en crear un Controller es crida la funció initializer. Aquesta funció l'estrucuta que te és la següent, demanar el model que pot tenir la vista, tot seguit demanar la vista principal (normalment Layout) on aniran la resta de vistes adheries, tot seguit s'escola a l'event de "show" de la vista (quan ha sigut pintada al DOM, renderitzada) i un cop renderitzada es pinta el contingut de les seves regions (només en te una, from-regino) després fem un show de la vista. Per pintar el contingut de les seves regions seguim el mateix procediment, primer demanarem els Modles (si en tingues) i després les vistes (userFields) i després la pintem a la vista a la regió que l'hi correspon(formRegino), la part de codi que ens hem saltat s'explicara més endavant, ja que ara portaria molta confusió, per exemple apareix la crida a demanar el Form de Utilities.

```
YTAneto.module("UsersSignUpApp.Show", function(Show, YTAneto, Backbone, Marionette, $, _){
  Show.UsersSignUpLayout = YTAneto.Views.Layout.extend({
    template: "users_sign_up/show/templates/users_sign_up_layout",
    regions: {
      formRegion: "#form-region"
    },
  }),

  Show.UserFields = YTAneto.Views.ItemView.extend({
    template: "users_sign_up/show/templates/user_fields",
    form: {
      buttons: {
        placement: "pull-right"
      }
    }
  })

})
```

Figura 20: UserSignUp, View

Aquí (Figura 20) podem veure com estan definides les vistes en Backbone i Marionette. I en el Controller podem veure com les inicialitza. En el Layout (la primera de totes) podem veure que es defineix on esta el fitxer HTML que ha de rederitzar i les regions que conte (són ID's definides en el DOM) el següent són els Fields i podem veure que es defineix el la ruta a les vistes (el form es per el Form de Utilities, recordeu?,

són unes opcions per determinar on es volen els botons). Els fitxers HTML no tenen cap repte.

Fields (Figura 21):

```
<div class="form-group">
  <div class="input-group">
    <span class="input-group-addon"><span class="glyphicon glyphicon-user"></span></span>
    <input type="text" name="name" class="form-control" placeholder="Name" />
  </div>
</div>
<div class="form-group">
  <div class="input-group">
    <span class="input-group-addon"><span class="glyphicon glyphicon-user"></span></span>
    <input type="text" name="username" class="form-control" placeholder="Username" />
  </div>
</div>
<div class="form-group">
  <div class="input-group">
    <span class="input-group-addon"><span class="glyphicon glyphicon-envelope"></span>
    </span>
    <input type="text" name="email" class="form-control" placeholder="Email Address" />
  </div>
</div>
<div class="form-group">
  <div class="input-group">
    <span class="input-group-addon"><span class="glyphicon glyphicon-lock"></span></span>
    <input type="password" name="password" class="form-control" placeholder="Password" />
  </div>
</div>
<div class="form-group">
  <div class="input-group">
    <span class="input-group-addon"><span class="glyphicon glyphicon-lock"></span></span>
    <input type="password" name="password_confirmation" class="form-control" placeholder="Password Confirmation" />
  </div>
</div>
```

Figura 21: UserSignUp, HTML Fields

Layout (Figura 22):

```
<div id="sign-up-form" class="container">
  <div class="row">
    <div class="col-md-4 col-md-offset-4">
      <div class="panel panel-default">
        <div class="panel-body">
          <h5 class="text-center"> SIGN UP </h5>
          <div id="form-region"></div>
        </div>
      </div>
    </div>
  </div>
</div>
```

Figura 22: UserSignUp, HTML Layout

En el Layout podem veure definida la form-region, que és on anirà adherit a els Fields (englobats per el <form> que posara el Form Utilites).

Ens és molt difícil d'explicar tot el funcionament intern de l'aplicació, ja que tota la lògica de JavaScript és molt complexa, així que per ara molts detalls són màgia (un terme que a nosaltres no ens agrada quan programem, sóm conscients).

Doncs això és tota la lògica del sign_up, la lògica de crear l'usuari al fer Submit, esta delegada al Form de Utilities. Amb aquest exemple hem vist el funcionament de les rutes, l'estructura dels directoris d'una app, com funciona un Controller i l'ordre de les seves accions més bàsiques i com estan descrites les vistes. Aquest exemple és el més simple (i per simple em refereixo que conte menys lògica i menys línies), la complexitat pot ser extrema fins al punt de tenir SubAplicaccions de SubAplicacions (a YourTalks en tenim un cas, la Room amb els moduls interns), tenir vistes amb molta més lògica (com control d'events que actuen a la vista i informar el Controller) i Layouts amb moltes regions que inclouen altres Layouts amb més regions, la complexitat pot ser infinita, mentres mantingui una coherència.

6.1.2.2 Estructura Modular de Marionette

En moltes parts anteriors comentem l'estructura Modular de Marionette i aclarim que s'explicara més endavant, ara és el moment. Si us heu fixat en les figures anteriors haureu pogut contemplar que a l'inici de cada fitxer (menys en el app.js) hi ha una línia similar a la següent (Figura 23):

```
YTAneto.module("UsersSignUpApp.Show", function(Show, YTAneto, Backbone, Marionette, $, _){
```

Figura 23: UserSignUp.Show Module

Aquesta és la sintaxis que utilitza Marionette per definir Moduls, els paràmetres tenen el següent significat:

- Primer el nom del Modul, si és un SubModul es separa per punt, per tant entenem que el modul es diu Show i que el seu pare és UsersSignUpApp (Figura 24).

```
YTAneto.module("UsersSignUpApp", function(UsersSignUpApp, YTAneto, Backbone, Marionette, $, _){
```

Figura 24: UserSingUP Module

- El següent paràmetre és una funció, aquesta funció rep uns paràmetres que sempre són els mínims (es poden afegir més si es vol, però els mínims són aquests), són les dependències del Modul, cal entendre que aquests noms són variables que arriben en inicialitzar el moduel, per tant el nom que posem aquí no és important però és rellevant, ja que sempre arribaran en el mateix ordre i si canviem l'ordre de les variables, seguiran arribant en el mateix ordre, no diferent:
    - Show: És el nom del Modul.
    - YTAneot: És el nom de l'aplicació principal, la pare.
    - Backbone: És la llibreria de BackboneJS per ella mateixa.
    - Marionette: És el Framework de Marionette per ell mateix.
    - $: És refereix a JQuery, es una dependència oblicagoria de Backbone.
    - _: És refereix a UnderscoreJS, és una dependència obligatoria de Backbone.

Ara que entenem això si revisem cada apartat anterior, ens podrem fixar en els noms que s'han donat als Moduls (Figura 25, Figura 26, Figura 27) i com estan organitzats:

```
YTAneto.module("Components.Form", function(Form, YTAneto, Backbone, Marionette, $, _){
```

Figura 25: Components Form Module

```
YTAneto.module('Utilities', function(Utilities, YTAneto, Backbone, Marionette, $, _) {
```

Figura 26: Utilities Module

```
YTAneto.module('Regions', function(Regions, YTAneto, Backbone, Marioneter, $, _){
```

Figura 27: Regions Module

Utilitzan els Moduls de Marionette aconseguim una estructuració de l'aplicació perfectament separada i comprensible. En trets generals cada Module de l'aplicació correspon a un elemnt únic de la vista (Header, Footer, MainRegion …), cada un d'aquests elements no te comunicació directe, no esta lligat, amb cap altre modul o funció, són

independents. Aquesta independència és molt important, ja que es podrien extreure del codi i segui funcionant, es podrien testejar separats del seu entorn i veure que passa i una de les característiques més importants, són 100% reutilitzables per altres aplicacions futures, sobretot aquells que és troben dintre de LIB, tots aquells han estat escrits des de 0, la següent aplicació que fem els podrem reutilitzar i així traurem un gran pes de sobre i un temps d'implementació innecessari, es poden anar millorant el temps.
JavaScript per ell sol no te una eina de Modulitzar, s'han d'utilitzar eines separades, llibreries, les més comunes com CommonJS, RequireJS i AMD. Ara només ens queda entendre com es comuniquen entre ells els Moduls per no esta acoblats entre ells d'una manera directa.

6.1.2.3 Comunicació entre Moduls i l'aplicació

Aquest és l'apartat final per entendre tot el funcionament de l'aplicació, la comunicació interna, com es comuniquen els Moduls entre ells per no esta acoblats. Mitjançant un bus de comunicació que ens ho proporciona Backbone.Wreqr. Marionette l'adapta per proporcionar-nos 3 canals diferents de comunicació, que representen 3 patrons, funcionen bàsicament amb events, com que JavaScript es un llenguatge asíncron per ell sol ja funciona amb events i callbacks per als events:

- Observador (Pub/Sub): Marionette ens ofereix la possibilitat d'integrar comunicació mitjançant Pub/Sub. Aquest patró ens permet executar un event (Pub) i que algú escolti aquest event (Sub). Un exemple en d'aquesta implementació en Marionette seria el següent. El primer exemple (Figura 28) mostra com publicar a un canal ("new:session") s'utilitza el .vent.trigger, i transmeten uns paràmetres, el model. El segon exemple (Figura 29) mostra com subscriure's al canal i rebre el model, s'utilitza el .on, al rebre el la publicació de new:session crida un callback (en aquest cas guardar la sessió). S'utlitiza en tota l'aplicació per comunicar que algunes accions han succeït, si algú vol estar atén a què han succeït s'ha de subscriure.

```
YTAneto.vent.trigger("new:session", model)
```

Figura 28: Creació (PUB) de l'event

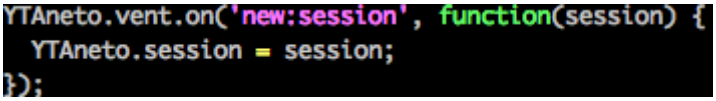


Figura 29: Escoltador (SUB) del event

- Request/Response: Es semblant al Pub/Sub l'únic es que aquest canal suporta objectes. En comptes de publicar i subscriure's i executar un callback al rebre la publicació, el que fa es demanar que algú que estigui subscrit al canal l'hi respongui a la petició que ha fet. S'utlitiza sobretot per demanar els Models en el Controller. Al primer exemple (Figura 30) podem observar com demana per les "inquiry_questions:entities" mitjançant el .request. En el segon exemple (Figura 31) veiem com amb .reqres.setHandler escolta per la petició de les "inquiry_question:entities" i respon amb les inquiryQuestions. Ha d'existir un setHandler per cada .request que hi hagi, sinó d'ona un error en temps d'execució.

Figura 30: Creació Request

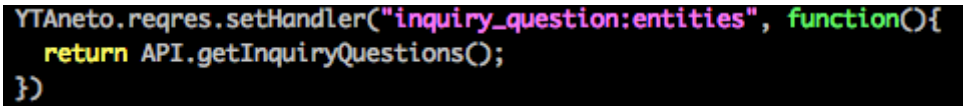


Figura 31: Reponsable de respondre la Request

- Command: Aquest tipus de missatgeria s'utilitza per executar comandes, ordres. No espera una resposta a la comanda, només mana que algú ha de fer l'acció. La manera d'executar una comanda és amb .execute i el nom del canal, la resta d'opcions són per al receptor (Figura 32) (en aquest cas perquè surti una notificació). Per obeir la comanda s'utilitza el .commands.setHandler d'aquesta manera s'executa (Figura 33) la comanda ordenada (en aquest cas la notificació es mostrarà per pantalla).

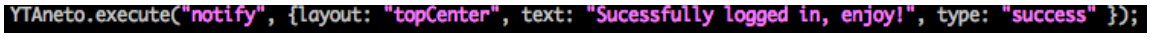


Figura 32: Creació Command

```
YTAneto.commands.setHandler('notify', function(options){
  return new Notify.NotifyController({
    region: YTAneto.notifyRegion,
    config: options
  })
})
```

Figura 33: Responsable Command

Pot semblar un punt molt complexe d'entendre el de la comunicació interna per es molt senzill, cada una de les 3 possibles maneres té un rol diferent, no només es distingeixen en la descripció. Si ho traslladéssim a la vida real, podríem dir que cada un seria una marea de demanar una cosa diferent:

- Pub/Sub: "Hi ha un llibre nou"
- Request/Response: "Donam el llibre"
- Command: "Dona-l'hi el llibre!"

Aquesta és la manera de comunicar-se entre Moduls de l'aplicació sense acoblar-los uns amb els altres i provocar dependències innecessàries. La manera en què entenem aquesta missatgeria es com si fos una API, cada Modul disposa d'una API, tan d'entrada com de sortida, si vols que el Modul faci això, l'hi tindras que dir d'aquesta manera i, el Modul pot ser que et demani això d'aquesta manera.

6.3 Servidor de Real Time

El servidor de Real Time (RT) és el que s'encarrega de fer la comunicació entre clients o entre l'API i els Clients. És un servidor de NodeJS amb socket.io, tal com vam explicar al definir socket.io, la seva implementació no requereix molta complicació, el que si requereix és una estructura ben pensada i robusta, ja que si no es pot acabar creant un embolic d'events amunt i avall. La figura (Figura 34) que hem adjuntat més avall mostra tot el servidor de Real Time, com es pot veure són un total de 32 línies, amb aquestes 32 línies s'aconsegueix crear les següents funcions:

- Fer que un usuari crei la room, crear rooms en socket.io és la manera més fàcil d'agrupar conjunts de Clients que comparteixen alguna cosa en comú, que en algun moment a tots ells se'ls pot notificar quelcom. En el nostre cas d'ús creem una sala

per a cada Conferencia creada, tots els participats estaran en la mateixa room amb el nom que te la sala en el Client. Un cop creada, es comunica al creador que ha creat la sala

- Fer que un usuari s'uneixi a la sala amb els altres, un cop s'ha unit, és comunicat a tots els usuaris menys ell mateix.
- Fer que s'enviïn els "im" Instan Message, el xat que te la sala, en aquest cas, s'envia la comunicació a tots els sockets de la sala.
- Fer que les slides es passin per a tots els clients de la sala.

Aquesta informació l'hi arriba al servidor mitjançant un JSON amb tota la informació que el Client vol comunicar als altres Clients, aquest JSON te una estructura semblant a la següent:

```
{
  "room":"eupmt",
  "type":"im",
  "im":{
        "message":"El missatge del xat",
        "from":"xescugc"
  }
}
```

Aquest JSON mostra un exemple de l'estructura dels JSON enviats quan es fa un event en el canal de "room". A l'arrel de JSON sempre tindrem l'atribut room i type, a continuació un atribut que serà el mateix que el tipus que sigui, i dintre amb la informació referent al type, en aquest cas al ser un IM, enviem el missatge, i qui l'ha enviat. És una manera molt bàsica de poder-ho implementar, en un futur aquesta estructura canviara i serà més complexa.

Redis l'utilitzem per Pub/Sub, esta escoltant a canals i quan es publica quelcom, el socket envia un event al Client o Clients que pertoqui, per exemple quan els workers han acabat de processar una presentació, se l'hi enviara a l'usuari un missatge per pantalla dient que la seva presentació ja ha acabat.

```
var io = require('socket.io').listen(2014);

io.sockets.on('connection', function(socket) {
  socket.on('message', function(data) {
  });

  socket.on('room', function(data) {
    console.log('room');
    switch (data.type) {
      case 'create':
        socket.join(data.room);
        console.log("rooms", io.sockets.manager.rooms);
        socket.emit('created');
        break;
      case 'join':
        console.log("rooms", io.sockets.manager.rooms);
        socket.join(data.room);
        socket.broadcast.to(room).emit('joined');
        break;
      case 'im':
        io.sockets.in(data.room).emit('im', data);
        break;
      case 'slide':
        io.sockets.in(data.room).emit('slide', data);
        break;
    }
  });

  socket.on('disconnect', function(data) {
    console.log('disconnect');
  });
});
```

Figura 34: Servidor de Real Time

7. Estudi Econòmic.

Per a realitzar una bona estimació dels costos, s'ha dividit en dos grups, d'una banda els costos generats pels recursos humans i d'altra banda els costos que provenen dels recursos materials. A tots i cadascun dels costos inclosos en aquest apartat se'ls ha afegit l'impost sobre el valor afegit (IVA).

En primer lloc ens centrarem en els costos procedents dels recursos humans i per això farem ús de JIRA. S'ha realitzat a partir de les tasques introduïdes a JIRA i una aproximació de les hores necessàries per desenvolupar el projecte fins a la seva entrega inicial. Aquí es descarta els costos futurs, en properes versions.

Personal	Hores de treball (aprox)	Preu/hora	Total	Desviació hores aprox	Preu total
Anàlisis de requeriments					
Gestor del projecte	20 h	35 €/h	700 €	20 %	840 €
Desenvolupador 1	450 h	30 €/h	13500 €	20 %	16200 €
Desenvolupador 2	350 h	30 €/h	10500 €	20 %	12600 €
Administrador de sistemes	100 h	35 €/h	3500 €	20 %	4200 €
Total del projecte	920 h		28200 €	20 %	33840 €

Taula 3: Estudi econòmic

Com es pot observar a la taula anterior, el cost total procedent dels recursos humans és de 16200 €. També es pot veure el temps dedicat al projecte de cadascuna de les persones, el salari que cobra per hora i el salari total per a la seva participació en el projecte.

Per estimar els costos procedents dels recursos materials tindrem en compte tant la despesa produïda pel programari com pel maquinari.

Material	Cost
Ordinador portàtil (Macbook Pro)	180 €
Ordinador portàtil (Macbook Air)	180 €
Material fungible	50 €
Impressora	100 €
Total	510 €

Taula 4: Cost material

Com es pot veure en la taula anterior, els costos generats pels recursos materials ascendeixen a 510 €. Per tant, tenint en compte la despesa generada tan pels recursos materials com pels humans, el cost total estimat del projecte és de 34350 €.

7.1 Moneteització

S'ha fet un petit brainstorming de les possibles sortides de negoci de l'aplicació. Algunes de les quals són:

- Mostrar anuncis via Adwords o plataformes d'afiliats.
- Freemium:
    - Oferir un servei gratuït fins a un total de X sales al mes.
    - Oferir només alguns mòduls gratuïts i altres de pagament.

    - Sales amb **límit de gent**.
- Premium
    - Accés a totes les característiques de la plataforma.

8. Conclusions

L'objectiu principal d'aquest projecte ha estat la realització de l'estructura base i l'investigació de les tecnologies per desenvolupar YourTalks que consisteix en crear una plataforma web per a la realització de conferències/classes o qualsevol altre tipus de xerrada en temps real.

La principal conclusió en el desenvolupament del treball ha estat la presa de decisions per escollir les eines a utilitza ja que existeixen moltes tecnologies per realitzar les mateixes accions, però cada una amb les seves particularitats.

La dificultat més gran que ens hem trobat ha estat l'aprenentantage de les noves tecnologies utilitzades, com per exemple MariontteJS o Puppet, amb el qual no havíem treballat mai. Aquest fet va provocar que les estimacions inicials no s'aproximesin a la realitat del desenvolupament.

Des de l'inici s'ha portat un control de l'estat del projecte a través del software JIRA i utilitzant les metodologies àgils, en concret, Kanban com una de les seves aplicacions. S'ha intentat seguir la planificació dels sprints, i segons l'evolució de les tasques s'han anat redefinint el següents sprints degut a les dificultats trobades. Aquest ha estat el punt clau que ens ha permès dur a terme amb èxit el projecte i complir amb els objectius establerts en l'apartat Abast. En l'anàlisi de requeriments s'ha intentat descriure amb el màxim de detall tots els aspectes que havia de cobrir l'aplicació per evitar que durant la implementació apareguessin masses canvis, tot i que, com és habitual en Kanban, els requisits van canviant segons les necessitas de cada moment.

Un dels grans reptes del projecte ha estat les tasques d'integració dels diferents components utilitzats en el projecte d'una forma escalada i modular. D'una banda els servidors, el frontend, l'API, el real time, els workers. Un dels principals objectius de la integració ha estat dissenyar l'arquitectura modular de la plataforma per a realitzar futures ampliacions.

En resum els objectius que ens havíem plantejat inicialment han estat aconseguits. L'objectiu de construir una aplicació que dugués el servei que hem estat desenvolupat durant tota la descripció del text, utilitzant noves tecnologies com Backbone Marionette comunicada amb un Servidor que proveeix l'API un de Real Time per la comunicació entre clients, tot les les altres tecnologies que hem investigat i utilitzat en el projecte com NodeJS, Socket.io, Grunt, Handlebars, Bower, Redis, Yeoman, JavaScript, UnderscoreJS i JQuery. Totes aquestes tecnologies presenten un repte, ja que la majoria són molt recents.

Aquesta nova distribució que hem aplicat, de tenir l'API per un cantó i el client per un altre, totalment separats, és la distribució que s'utilitza en totes les grans empreses, és el futur al qual està evolucionant la web, ja que presenta una major escalabilitat i distribució de les lògiques, cada una fa el que l'hi pertoca, i poder nosaltres aplicar aquesta nova distribució ha sigut una gran experiència.

Hem aconseguit entrar més profundament en tota l'estructuració i implementació d'una aplicació que ha començat des de zero hi ha anat creixent progressivament i escaladament fins a convertir-se en una aplicació amb una lògica complexe que pot augmentar fàcilment i sense un gran grau de dificultats, ja que des de l'inici tot ha estat pensat i estructurat en aquest propòsit.

El fet de treballar amb aquestes tecnologies que són relativament noves és un punt que sempre d'ona molts ànims i ganes de desenvolupar i de descobrir quins secrets amagats tenen les llibreries que utilitzem, anar aprenent dels errors per què a la següent aplicació que desenvolupem no tornar-los a cometre i així desenvolupar més rapidament. Tots els coneixements obtinguts durant el desenvolupament seran de gran ajuda per al nostre futur.

Com ha conclusió final podem dir que s'ha implementat l'estructura base per crear una plataforma que permeti oferir una experiència el més semblant possible a l'assistència física a una conferència.

Per això, podem concloure que s'han assolit amb èxit tots i cadascun dels objectius establerts.

9. Referències

Backbone.JS. (2014). From http://backbonejs.org/

Bailey, D. (23 / 12 / 2011). *http://lostechies.com/derickbailey/2011/12/23/backbone-js-is-not-an-mvc-framework/.* Consultat el 2 / 3 / 2014, a http://lostechies.com: http://lostechies.com/derickbailey/2011/12/23/backbone-js-is-not-an-mvc-framework/

Bailey, D. (19 / 4 / 2012). *http://lostechies.com/derickbailey/2012/04/19/decoupling-backbone-apps-from-websockets/.* Consultat el 8 / 2 / 2014, a http://lostechies.com: http://lostechies.com/derickbailey/2012/04/19/decoupling-backbone-apps-from-websockets/

Bailey, D. (sense data). *https://github.com/marionettejs/backbone.marionette*. Recollit de https://github.com/marionettejs/backbone.marionette: https://github.com/marionettejs/backbone.marionette

Bash. (n.d.). Retrieved 2014 from http://www.gnu.org/software/bash/

Capistrano. (n.d.). Retrieved 2014 from http://capistranorb.com/

Cluster de alta disponibilidad. (n.d.). Retrieved 2014 from http://lobobinario.blogspot.com.es/2011/09/cluster-de-alta-disponibilidad-balanceo.html

Crockford, D. (2008). *JavaScript: The Good Parts.* Minnesota: O'REILLY.

Denoe - Estructura cloud computing. (n.d.). Retrieved 2014 from http://www.denoe.es/test/wp-content/uploads/estructura-cloud-computing.png

Dev2Ops. (n.d.). Retrieved 2014 from http://dev2ops.org/2010/11/devops-is-not-a-technology-problem-devops-is-a-business-problem/

Funio - Arquitectura de alta disponibilidad. (n.d.). Retrieved 2014 from https://es.funio.com/arquitectura-alta-disponibilidad

https://www.ruby-lang.org/es. (n.d.).

Killilea, J. *Marionette Exposé.* Jack Killilea.

Mann, B. (12 / 2 / 2013). *http://www.backbonerails.com/*. Consultat el 5 / 6 / 2013, a http://www.backbonerails.com/: http://www.backbonerails.com/

Marionette. (n.d.). Retrieved 2014 from http://marionettejs.com/

Nginx. (n.d.). Retrieved 2014 from http://nginx.com/

Node.JS. (n.d.). Retrieved 2014 from http://nodejs.org/

Ruby On Rails Website. (n.d.). From http:/www.rubyonrails.org

Sulc, D. *Backbone.Marionette.js: A Gentle Introduction.* Switzerland: David Sulc.

The Javascript Task Runner. (n.d.). Retrieved 2014 from http://gruntjs.com/

Unicorn. (n.d.). Retrieved 2014 from http://unicorn.bogomips.org/

What is Puppet? (n.d.). Retrieved 2014 from http://puppetlabs.com/puppet/what-is-puppet

Wikipedia - Alta disponibilidad. (n.d.). Retrieved 2014 from https://es.wikipedia.org/wiki/Alta_disponibilidad

Wikipedia. (n.d.). *Javascript.* Retrieved 2014 from http://es.wikipedia.org/wiki/JavaScript