Desenvolupament d'un dashboard de mètriques en temps real

Memòria de Projecte Final de Màster
Màster Universitari en Enginyeria Informàtica
Desenvolupament d'aplicacions web

Autor: Eduardo José Heredia Lubino

Consultor: Ignasi Llorente Puchades
Professor: César Córcoles Briongos

17 de Gener de 2016

Crèdits/Copyright

Cita

« Let the future tell the truth, and evaluate each one according to his work and accomplishments. The present is theirs; the future, for which I have really worked, is mine.» *Nikola Tesla*

Abstract

Quina utilitat té la visualització de KPI? Quin guany en tenim?... Hi ha moltes preguntes que ens podem fer i abans de respondre-les hauríem de definir el concepte. Una bona aproximació és la següent:

«Un KPI (key performance indicator) és una mètrica de negoci utilitzada per avaluar factors que són crucials per l'èxit d'una organització.» *Wikipedia*

Per tant una mètrica de negoci és una part important del negoci que ens permet aproximar comportaments futurs i ens alerta d'esdeveniments recents. Permeten prendre decisions basades en dades.

Un dashboard (o panell) de KPI és una eina que permet visualitzar mètriques obtingudes de diferents fonts amb l'objectiu tenir una visió general d'un conjunt de dades i veure la seva evolució en el temps.

El dashboard que s'ha desenvolupat en aquest projecte, intenta aportar una visió moderna i un punt de vista diferent dels actuals sistemes, però sense reinventar la roda. Permet obtenir informació rellevant que pot ser molt interessant per diferents perfils d'una organització com pot ser negoci, departaments tècnics, finances …

What utility has to show the KPI? What we gain? ... There are many questions that we can answer them but before we should define the concept. A good approach is the following:

"A KPI (key performance indicator) is a business metric used to evaluate factors that are crucial to the success of an organization." Wikipedia

So a measure of business is an important part of the business that allows us to approach future behavior and warns of recent events. Allow to make decisions based on data.
A dashboard KPI is a tool that can display metrics obtained from different sources in order to get an overview of a data set and see its evolution in time.

The dashboard has been developed in this project tries to provide a modern vision and a different point of view of the current systems, without reinventing the wheel. Allow obatin relevant information that can be very interesting for different profiles of an organization such as business and technical departments, finance ...

Índex

Figures i taules

Índex de figures

1. Introducció

La programació web és una branca relativament jove del desenvolupament d'aplicacions. Va néixer fa uns 25 anys però no va ser fins finals dels 90, principis de segle quan va començar a popularitzar entre la població.

L'evolució ha permès desenvolupar negocis *online* que generen quantitats molt importants d'ingressos. L'any 2015 les vendes dels *ecommerce* va arribar a 1.7 trilions de dollars[1], una xifra que no farà més que augmentar en els pròxims anys tenint en compte l'ús habitual de dispositius mòbils per la gran majoria de la població.

Les anomenades empreses *Data Driven* (orientades a dades), són aquelles empreses on l'anàlisi de dades formen el nucli del seu negoci o, com a mínim, els aporta informació rellevant per dur a desenvolupar la seva operativa diària i a mig o llarg termini.

Disposar d'eines d'anàlisis i visualització de dades esdevé fonamental per entendre les dinàmiques de comportament, tant dels usuaris com dels sistemes que interactuen en xarxa.

La nostra proposta ha desenvolupat, la nostra versió de com hauria de ser un *dashboard* de KPI (Key Performance Indicator) modern. Pot visualitzar les dades en temps real, i és altament configurable, extensible i amb característiques pròpies que diferenciar d'altres alternatives disponibles.

L'interès que pot tenir aquesta eina, des d'un punt de vista de projecte acadèmic, és que implica interactuar entre moltes àrees de coneixement: programació (backend, frontend), configuració de sistemes, anàlisis de dades, usabilitat, disseny per dispositius mòbils, planificació… I fins i tot es podria ampliar a altres àrees com el d'intel·ligència artificial o alt rendiment.

Per últim remarcar que s'oferirà el codi font com a codi lliure i serà una aportació a la comunitat.

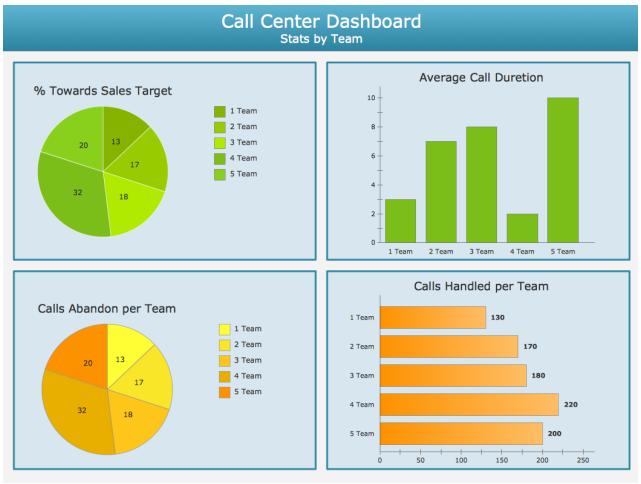


Figura 1: Exemple de dashboard

[1] http://www.slideshare.net/divanteltd/ecommerce-trends-from-2014-to-2015

2. Hipòtesi

Partim de la base de què un error en un entorn de producció costa molts diners, per tant és prioritari obtenir una retroalimentació al més aviat possible. La majoria de dashboards, en el millor dels casos, obtenen la informació a intervals de temps determinats (per exemple cada 15 segons des d'una base de dades) o mitjançant push de dades (a partir d'un cron per exemple). Existeixen aplicacions webs de grans empreses on un retràs tan gran no es pot permetre. Els motius són diversos com poden ser pèrdues d'ingressos, mala imatge per l'usuari ...

Nosaltres analitzarem la possibilitat de buscar una solució més potent que permetrà obtenir les dades d'una font de dades en temps real minimitzant el temps de resposta i d'actuació en cas d'error.

La hipòtesi que s'ha seguit en aquest projecte era la següent:

> És possible desenvolupar un dashboard (panell) on els widgets (elements) que el composen puguin obtenir la informació en temps real i actualitzar-se de forma automàtica sense la necessitat d'un refresc per interval o un push de dades per part d'una aplicació externa o cron.

Utilitzant un *framework* del frontend del tipus *Model View Presenter*, mes una font de dades en temps real i amb un sistema de widgets que es comportin de forma independent, s'obté un dashboard potent.

3. Objectius

L'objectiu principal d'aquest projecte és el següent :

Desenvolupar un dashboard de mètriques modern, altament configurable, responsiu i proactiu (que reaccioni quan una mètrica no està en el rang) amb la característica que s'actualitzi en temps real

3.1 Objectiu Principals

- Desenvolupar un dashboard de mètriques en temps real
- Codi d'alta qualitat
- De fàcil configuració per l'usuari
- Proactiu
- Fàcilment ampliable
- Intuïtiu

3.2 Objectius Secundaris

- Investigar tecnologies emergents que estiguin disponibles
- Integrar un sistema d'alertes
- Diferenciar d'altres propostes existents en el mercat
- Alt rendiment
- Responsive

4. Escenari

Com s'ha comentat anteriorment, l'evolució d'Internet ha fet que es desenvolupin molt tipus de negocis en xarxa. Hi ha negocis de tots els nivells des de petites empreses fins grans multinacionals. El que tenen en comú és com que generen dades, que analitzades correctament poden aportar tendències interessants per negoci. Lògicament com més gran sigui el volum i la criticitat de les mateixes, l'impacte en la seva detecció o anàlisis serà més important.

L'eina desenvolupada en aquest projecte **està orientada a un entorn professional** on les dades es puguin processar en temps real o semblant**.** No estaria pensat per un entorn on les dades no estiguessin normalitzades o el processament de les mateixes trigués més de 2 minuts (com podria ser per exemple una consulta a un clúster de nodes de **Hadoop** de les dades d'un tracking d'usuari d'una web amb alta activitat).

Aquest projecte també es pot utilitzar per projectes personals, en tenir una llicència oberta. El benefici obtingut seria menor però igualment útil.

També és especialment indicat per entorns tècnics on es volen obtenir dades no crítiques, per exemple el nombre de tests unitaris que hi ha, les línies de codi existent en el repositori, etc. Serien dades útils per un entorn tècnic com pot ser un departament de IT. En estar actualitzades en temps real aporten feedback immediat.

Veure les dades no és suficient, moltes vegades les dades són bones o dolentes quan es comparen amb unes altres. Per tant la integració d'un sistema d'alertes per avisar de quan s'ha produït una desviació en una mètrica és un punt fonamental què es té en compte en aquest projecte. Detectar una desviació d'una mètrica pot aportar, com a mínim, una pèrdua econòmica menor.

5. Contingut

S'ha desenvolupat una aplicació web on els **usuaris** tenen la possibilitat de crear i gestionar **dashboards** (panells) on s'incrustaran **instàncies de widgets** (petites aplicacions independents) d'un tipus determinat. Tant la creació d'instàncies com la configuració de les mateixes es realitza de forma intuïtiva mitjançant la UX d'usuari.

El nombre de dashboards creats per l'usuari no està limitat, però per poder realitzar qualsevol operació sobre ell l'usuari s'ha d'identificar. Per tant els dashboards i les instàncies de widgets no són compartits entre usuaris.

La creació dels widgets que serveixen de base per la creació d'instàncies de widget, actualment, només es poden crear per codi. El motiu és que cadascú té una font de dades i una forma de gestionar de diferents. Cercar una solució universal queda fora de l'abast del projecte.

Tant el panell com les instàncies de widgets, es van refrescant per obtenir dades actualitzades. Si una mètrica obtinguda està fora dels valors normals, s'indicarà de forma visual.

En començar el projecte, es va realitzar fer un exercici per unificar el llenguatge tècnic i el llenguatge de negoci. Es van obtenir el següent llenguatge comú (Ubiquitous Language) :

Tècnic	Negoci	Llenguatge comú	Definició
Dashboard	Pantalla	Dashboard	Pantalles amb mètriques de negoci
Widget	Diferents aplicacions	Widget	Base per crear instàncies de widgets
Instància de widget	Petita aplicació amb dades	Instància de widget	Petita aplicació independent que mostra KPI
Dades	KPI	Widget Data	Dades de negoci
Usuari	Client	User	Persona que interactua amb el dashboard

6. Metodologia

En el desenvolupament del projecte es fa servir un marc de treball molt similar al que es fan servir a grans empreses del desenvolupament web. Es basa en 2 pilars fonamentals:

6.1 Visió àgil per aportar valor

Des d'un punt de vista de gestió, s'han creat èpiques (o histories) que tenen tasques. Aquestes s'han prioritzat tenint en compte alguns factors com són les interrelacions de les tasques i sobretot el valor que aporta al client. Cada setmana s'ha ajustat la planificació i veient si hi havia desviacions. L'objectiu era anar obtenint entregables, tan aviat com sigui possible, que aportin valor i que ens permetés veure l'evolució del projecte. Durant el desenvolupament del projecte no s'han produït desviacions i s'ha anat actualitzant la memòria per adaptar-la als avenços.

NOTA: A efectes pràctics s'ha seguit el diagrama de Gannt que es mostra a l'apartat 9, les històries són els punts principals i les tasques són els punts detallats. La planificació s'ha anat ajustant a mesura que s'anaven completant històries amb l'avantatge que no han aparegut grans inconvenients.

6.2 Qualitat del codi i dels processos

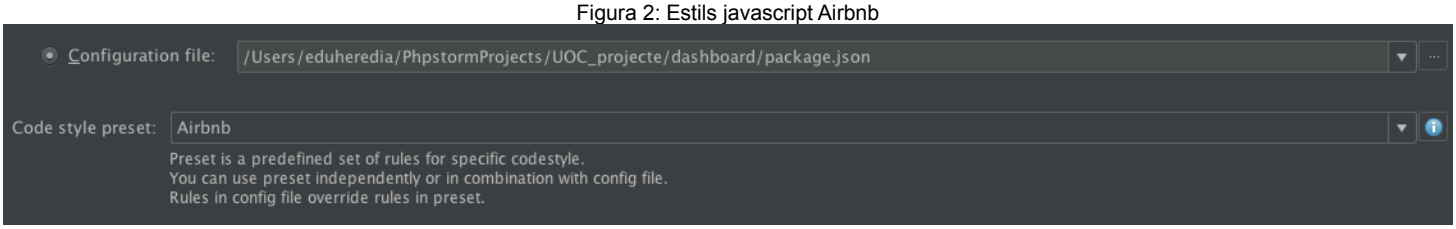
Encara que es faci servir una aproximació àgil, s'ha intentat que no se'n ressenti la qualitat. No s'ha creat gaire deute tècnic. S'ha seguit un conjunt ampli bones pràctiques des de l'inici.

Pel que fa al desenvolupament s'han fet servir patrons com **command** o **factory.** El primer s'explicarà més en detall quan s'expliquin els casos d'ús. Els patrons aporten solucions a problemes coneguts, per tant la visió que hem de tenir de no reinventar la roda s'aplica a aquest nivell també.

Hem tingut molt present les recomanacions SOLID i s'ha obtingut una alta cobertura de tests que ens permet tenir una alta confiança en el codi (ens ajuda no tenir por a refactoritzacions per exemple) i a més serveix de documentació de funcionament dels casos normals i els casos extrems.

Un bon punt de partida són les recomanacions recopilades en les webs de PHP the Rightway, PHP Framework Interop Group o Guia d'estil javascript Airbnb entre d'altres. Aquesta última s'aplica en el projecte.

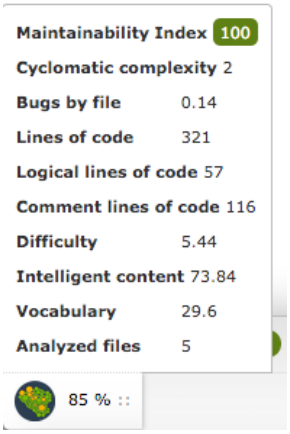
Figura 2: Estils javascript Airbnb



Desde la part de backend s'han fet servir aquestes eines de qualitat (entre d'altres):

- **PHP Git Hooks** → Realitza tasques de verificacions prèvies al commit dels fitxers. Verifica si el fitxer *composer.json* és correcte, si el codi PHP és sintàcticament correcte, si els tests passen …
- **Estàndard de codificació** → La codificació segueix l'estàndard PSR1/2
- **Revisió del codi IDE** → Des de el IDE PhpStorm es poden fer verificacions de qualitat de codi. Hi ha detecció de codi duplicat, anàlisis de codi entre d'altres.
- **Toolbar de debug** → El framework Symfony implementa de sèrie una barra on es poden veure informació molt interessant com accessos a la base de dades, temps de resposta, nº de peticions AJAX. Informació valuosa per millorar el codi.
- **Suites de testeig** → Framework per testejar el codi com phpunit o mocha.
- **PhpMetrics** → És un analitzador de codi estàtic, ens aporta informació de com de mantenible és el projecte, probabilitat de bugs, complexitat ciclomàtica …

Figura 3: Php metrics de la pàgina de login



L'objectiu d'obtenir una qualitat professional ha estat assolit en el projecte. Permet una alta mantenibilitat i un baix cost dels canvis. El projecte no s'ha desviat, al contrari, per tenir com a base la qualitat. Ens hem pogut centrar a desenvolupar noves funcionalitats en lloc de corregir errors.

6.3 Work flow

El projecte només disposa d'un perfil que té diferents rols. Això no és excusa per no disposar d'un sistema de control de versions. Ens ajuda a tenir el codi en un lloc centralitzat i ens permet mantenir un històric.

Pel que fa a la forma de treballar en el projecte, s'ha creat un repositori privat a Github (S'alliberà quan el projecte estigui més avançat si compleix la qualitat necessària). Aquest repositori disposa una branca **origin/master** que considerarem la branca de producció. Això vol dir que compleix els següents requisits:

- Es podria crear un tag de versió (major, minor o patch) i pujar a producció
- Passa tots els tests
- No accepta commits, només a través de pull requests.
- Ha de tenir una qualitat mínima, encara per determinar.

Es fa servir un sistema semblant a les **features branches**, però amb un àmbit una mica més gran. En un entorn real, l'abast de cada branch seria més petit. Un cop els canvis estan ajuntats, s'esborra la branca de desenvolupament que és una bona pràctica, sobretot en ambients amb molts desenvolupadors.

Figura 4: Repositori privat del projecte en github

6.4 Deploy

En el servidor de producció s'ha creat una carpeta on es clona el projecte, per evitar problemes de seguretat, els fitxers .git s'han configurat a fora de la mateixa. Un cop fet el *clone* del projecte tenim un fitxer **Makefile** que copia els fitxers en una carpeta temporal, executa els processos necessaris com instal·lar dependències, minimitzar assets, donar permisos a carpetas de caché o logs i eliminar fitxers innecessaris o potencialment perillosos en un entorn de producció. Un cop tenim una *build,* fem un canvi amb un link simbòlic i reiniciem els servidors per evitar problemes amb la caché.

En aquest cas no s'ha fet servir cap eina externa, el motiu és que hi havia altres tasques més prioritàries i el fet d'haver d'aprendre la nomenclatura de **ansible**, **puppet** o **chef** ens hagués endarrerit. Al final el resultat ha estat correcte, fer un deploy dura menys d'un minut, però encara té el defecte que és una mica manual.

L'ideal hauria estat tenir un servidor **jenkins** amb el deploy automatitzat com es fan en algunes empreses.

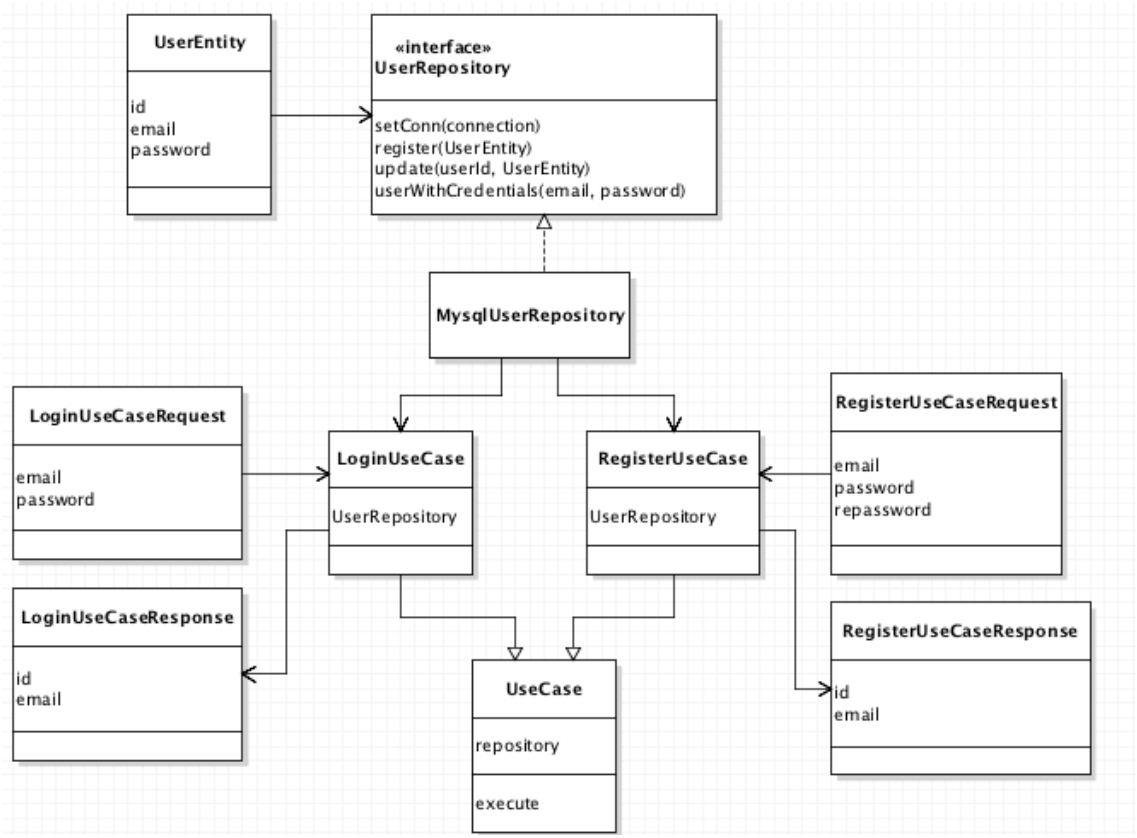
Figura 5: Exemple de deploy manual amb Makefile

```
build-deploy:
	git pull origin master
	mkdir -p /home/chefchef/www/chefuri.net/dashboard_deploy/
	cp * -R /home/chefchef/www/chefuri.net/dashboard_deploy/
	chown chefchef:chefchef /home/chefchef/www/chefuri.net/dashboard_deploy/ -R
	sudo mkdir -p /home/chefchef/www/chefuri.net/dashboard_deploy/app/cache
	sudo mkdir -p /home/chefchef/www/chefuri.net/dashboard_deploy/app/logs
	sudo chmod 777 /home/chefchef/www/chefuri.net/dashboard_deploy/app/cache -R
	sudo chmod 777 /home/chefchef/www/chefuri.net/dashboard_deploy/app/logs -R
	cd /home/chefchef/www/chefuri.net/dashboard_deploy/
	curl -sS https://getcomposer.org/installer | php
	chmod +x composer.phar
	cp app/config/parameters_prod.yml.dist app/config/parameters.yml.dist
	rm -rf /home/chefchef/www/chefuri.net/dashboard/
	mv /home/chefchef/www/chefuri.net/dashboard_deploy/ /home/chefchef/www/chefuri.net/dashboard/

build-prod:
	npm install
	export SYMFONY_ENV=prod
	bower install
	php composer.phar install --optimize-autoloader --no-dev
	php app/console assetic:dump --env=prod --no-debug

build-cache:
	sudo chmod 777 app/cache -R
	sudo chmod 777 app/logs -R
```

Com es pot observar no és la millor manera (tot i que funciona de forma correcta), hauria de ser més parametritzable i el deploy hauria de ser en un sol pas.

7. Arquitectura de l’aplicació

Figura 6: Diagrama de classes Usuaris



En aquesta primera part, s'observa un diagrama de classes de la part de registre i login d'usuari. S'ha utilitzat l' arquitectura de **DDD** (domain driven design) per deixar una arquitectura neta. Els casos d'ús accepten unes peticions i retornen una resposta uniforme. A més permet injectar en el constructor, la dependència del repositori que ha de ser de tipus UserRepository, per tant es podria injectar un repositori diferent del **MysqlUserRepository** per exemple un **RedisUserRepository**. Només cal que compleixi el contracte.

El fet d'injectar les dependències pel constructor, permet que aquesta pugui ser fàcilment reemplaçada, per exemple per un objecte mock (doble), molt útil per testejar classes específiques com poden ser de la infraestructura.

Els altres diagrames seran molt semblants, a continuació mostrem el del dashboard:

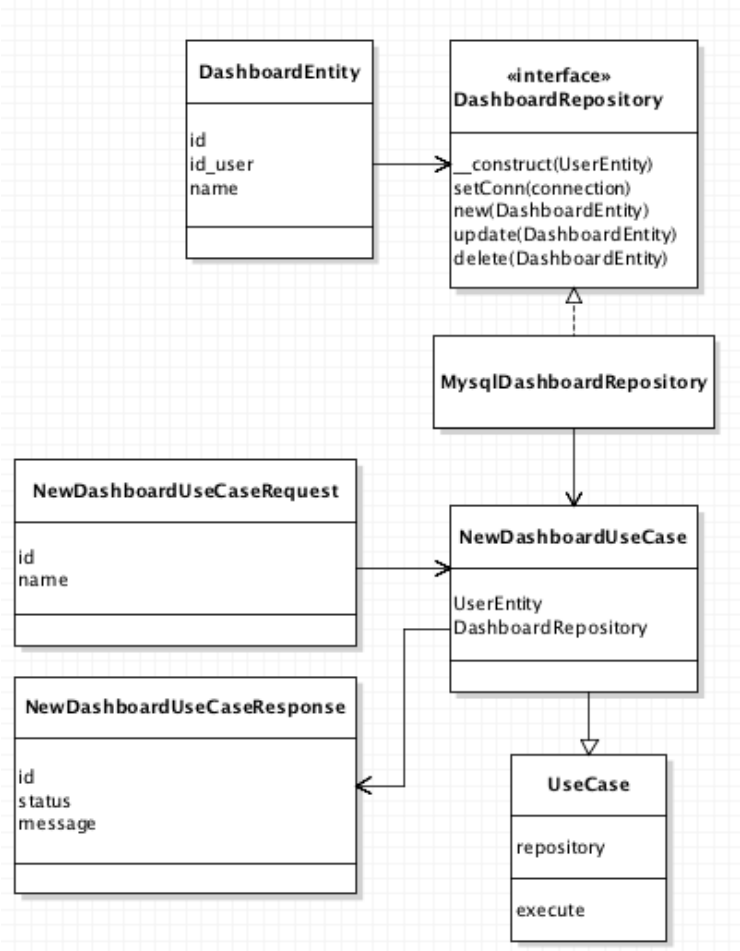


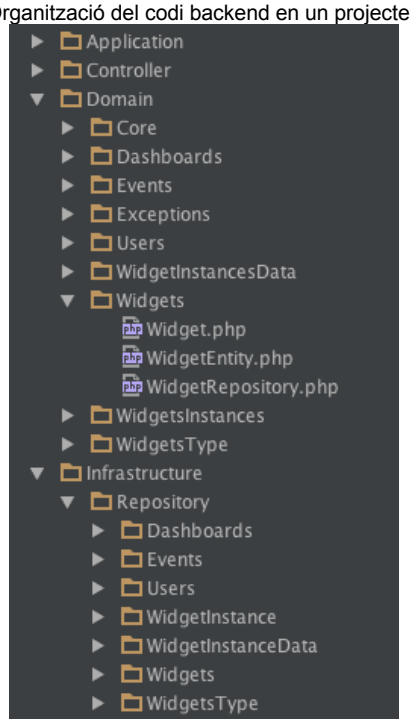
Figura 7: Diagrama de classes Dashboard (casos d'us, repositori, requests i responses)

El diagrama de classes del dashboard, té una dependència que és l'entitat Usuari. És a dir per crear un dashboard cal un usuari registrat. Per motius d'espai només s'ha posat un cas d'ús.

El diagrama de widgets que van dintre del dashboard no difereix gaire, però és una mica més complex, ja que requereix 2 repositoris (widgets i instàncies de widgets) i una entitat que serà el dashboard.

A continuació mostrem com s'organitza el codi :

Figura 8: Organització del codi backend en un projecte amb DDD



L'aplicació es separa en les següents capes:

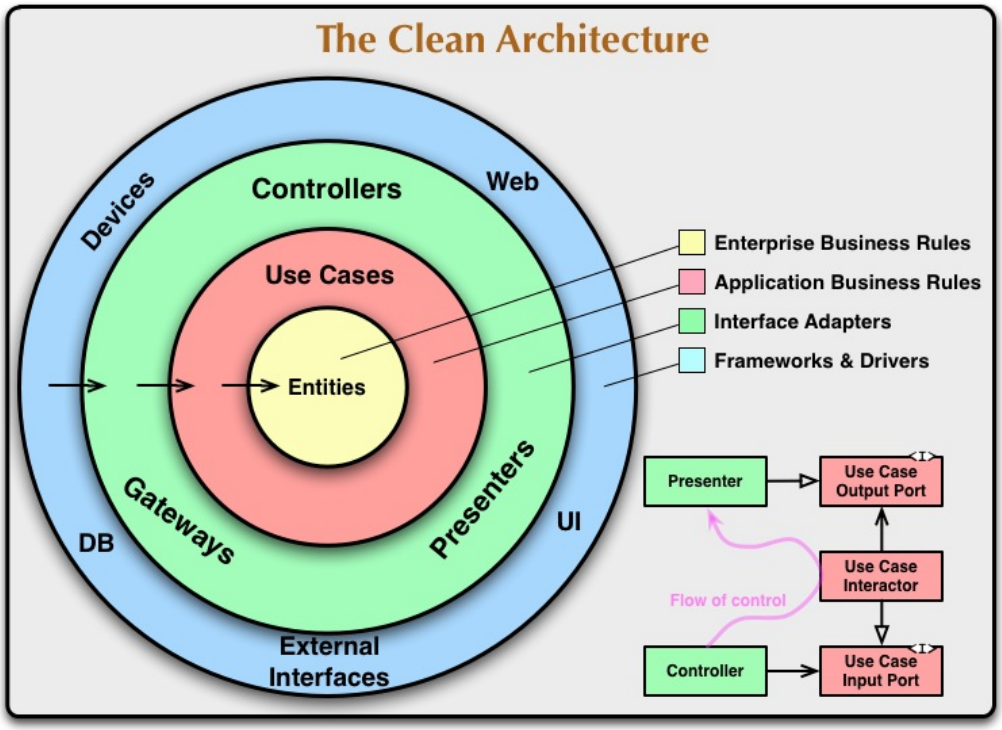
- **Controladors** → Reben i retornen peticions HTTP. Fan servir els casos d'ús, típicament encapsulat en serveis. Hauria de tendir a ser el més petit possible

- **Model**
  - **Entititats** → Son objectes que té una identitat i uns mètodes per interactuar amb ells. Per exemple una entitat usuari.
  - **Casos d'ús** → Tenen una única responsabilitat, donada una entrada ens donen una sortida. Totes les dependències que necessitin se'ls injecta pel constructor, típicament repositoris o d'altres serveis. En el punt 12 de la memòria es poden veure els diagrames de casos d'ús. Un exemple seria un servei de Login
  - **Repositoris** → Interfaç que serveix de contracte per un determinat repositori situat a la infraestructura.
  - **Events de domini** → Son sol·licituds per realitzar accions immediates o futures, ajuda a desacoblar parts del codi, per tant evitar duplicacions entre altres avantatges.
- **Infraestructura** → Implementació concreta d'un servei, per exemple l'accés a una base de dades. Es pot testejar però no és recomanable per ser massa específic.

La comunicació ha de ser entre capes, no es poden saltar:

UX → Controlador → Model (Casos d'ús) → Infraestructura
| Entitat
UX ← Controlador ← Model (Casos d'ús) ←Infraestructura

Figura 9: Clean architecture (https://blog.8thlight.com)



8. Plataforma de desenvolupament

Plataforma de producció

El projecte es una aplicació web, que en l'entorn de producció, està format per:

Servidor

L'aplicació estarà allotjada en el proveïdor de serveis DigitalOcean. En un «droplet» (imatge) amb 2 GB de RAM, 30 GB de disc SSD, i amb una base Ubuntu 14.04 de 64 bits.

Servidor web

Apache

Es pot utilitzar qualsevol servidor web que suporti reescriptura d' URL i els mòduls necessaris com poden ser el de PHP, reescriptura o mysql natiu.

Per servir web, farem servir Apache amb PHP amb aquesta configuració :

```
'./configure' '--with-apxs2=/usr/local/apache2/bin/apxs' '--enable-mysqlnd' '--enable-inline-optimization' '--disable-debug' '--with-pdo-mysql=shared' '--enable-mysqlnd-compression-support' '--with-zlib' '--with-gd=shared' '--enable-mbstring' '--enable-opcache' '--with-mysqli=mysqlnd' '--with-curl=shared' '--with-openssl'
```

amb els següents mòduls:

```
core mod_so http_core event mod_authn_file mod_authn_core mod_authz_host mod_authz_groupfile
mod_authz_user mod_authz_core mod_access_compat mod_auth_basic mod_reqtimeout mod_filter mod_mime
mod_log_config mod_env mod_headers mod_setenvif mod_version mod_unixd mod_status mod_autoindex
mod_dir mod_alias mod_rewrite mod_php7 mod_deflate
```

NOTA : L'extensió openSSL es necessaria per executar Composer a l'hora d'instal·lar les dependències. D'altres modificacions han sigut han sigut activar el opcache, configurar la zona horaria entre d'atres ajustos.

L'aplicació es desenvolupa en un entorn vagrant amb PHP 5.5 (veure punt 8 de la memòria), però en l'entorn de producció s'utilitza l'última versió estable de **PHP, concretament es fa servir PHP 7.0.2** Aquesta versió aporta una millora gran de velocitat sense pràcticament canvis que trenquin la compatibilitat.

Nginx
A part del servidor web, es requereix un servidor d'aplicacions per NodeJS, aleshores utilitzem nginx que farà de servidor proxy. Així és més fàcil de balancejar si l'aplicació de NodeJS requereix de més recursos, també ens pot ajudar a cachejar resultats si fes falta.

NodeJS / Express

S'ha creat una aplicació amb nodeJS que amb ajuda del framework Express, permet crear un petit servidor web que accepta rutes i permet gestionar els websockets.

	Versió	Servei	Port
Apache	2.4.18	Web	80
Nginx	1.4.6	Proxy	3000
NodeJS / Express	4.2.4 / 4.13.3	WebSockets	3001

Per mantenir l'aplicació de NodeJS sempre activa, es converteix en servei amb ajuda de Forever

Servidor base de dades

MySQL
Com a repositori permanent, utilitzem una base de dades relacional com MySQL per emmagatzemar la informació dels dashboards, widgets i la informació dels usuaris. Cal remarcar que l'aplicació de backend està programada de tal forma que té repositoris (de codi) que compleix un contracte i per tant és relativament fàcil canviar la infraestructura. En l'apartat 7 s'ha comentat amb més detall.

Redis

S'utilitza aquest servidor d'estructura de dades en memòria per guardar informació no permanent. Normalment s'utilitza per guardar dades prèviament processades, ja que es recuperen de forma molt ràpida. En la nostra aplicació s'utilitza com a sistema per enviar d'events en temps real (mitjançant les instruccions de PUB/SUB).

Plataforma de desenvolupament

Disposar d'un entorn de desenvolupament, el més aproximat possible a l'entorn real de producció sol ser una bona decisió en entorns corporatius. El motiu és fàcil d'entendre, l'aplicació s'hauria de comportar d'igual o molt similar manera quan s'executa en un entorn de desenvolupament que en un de producció per evitar errors en fases avançades. A la pràctica no sol ser gaire factible per diversos motius com solen ser de costos (tenir una infraestructura duplicada té costos econòmics i de manteniment).

Una altra alternativa és disposar d'un entorn de desenvolupament local i un entorn de pre-producció on poder fer QA(quality assurance) abans de passar l'aplicació a un entorn de producció. Aquesta aproximació té defectes com que es perd temps en configurar l'entorn local i a vegades no funciona de forma semblant a producció.

Una solució intermèdia és virtualitzar entorns, permetent reproduir l'entorn de forma més ràpid i automatitzat. Serveix tant per poder crear entorns iguals de desenvolupament com per crear entorns de producció que es puguin escalar horitzontalment. El cost sol ser una pèrdua de velocitat al virtualitzar recursos però ens aporta flexibilitat.

Nosaltres hem optat per aquesta última aproximació que consisteix a desenvolupar en un entorn virtualitzat i que sigui fàcilment reproduïble. Hem utilitzat **Vagrant** juntament amb **virtualbox** que ens permet disposar d'un entorn de forma ràpida (amb un màxim de 2 instruccions i un parell de minuts tenim l'entorn configurat).

Comú

Framework backend

L'elecció del framework **Symfony 2**, s'ha basat en criteris tècnics i personals. Els motius tècnics són que és un framework madur, amb una comunitat molt gran de col·laboradors, hi ha molta documentació i actualment és un dels estàndards de facto del mercat.

Com a motius personals, volia desenvolupar un projecte a partir de 0 amb aquest framework, ja que actualment ja no hi puc treballar professionalment. A més en tenir coneixement del funcionament es pot reduir la corba de desenvolupament de l'aplicació.

Framework frontend i llibreries

Els motius de l'elecció són similars a l'apartat anterior. A més amb les **JQuery UI**, té funcionalitats interessants per crear la zona de creació de dashboards com poden ser les funcionalitats d'arrossegar o redimensionar elements.

Hi ha altres elements que es detallaran més endavant en el seu context. De moment mostrem un resum del software utilitzat.

Resum

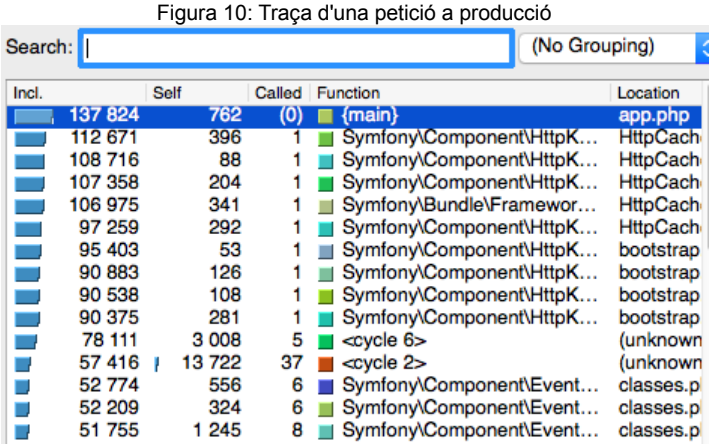
	Versió utilitzada	Alternatives
Sistema operatiu	Ubuntu	CentOS, RedHat ...
Servidor Web	Apache 2.4.27	Nginx, lighttpd …
Servidor Aplicacions	Express (NodeJS)	Hapi Koa.js, diet.js ...
Servidor Base de dades	MySQL 5.5	PostgreSQL, SQLLite ...
Base de dades Memoria	Redis	Mongodb ...
Llenguatge backend	PHP 5.5 / 7.0	Java, Python…
Framework backend	Symfony 2.7	Laravel, Silex….
Javascript Framework	Backbone JS 1.2.3	React PHP, Angular JS
Llibreries frontend	JQuery 2.1.4, JQuery UI ...	YUI
Gestor de paquets	Npm, bower, composer	Grunt (similar)
Carregador de mòduls	RequireJS	Puli Repository Component
Framework frontend	Twitter Bootstrap	Foundation
Templating framework	Underscore	Mustache
Testing suite JS	Mocha	Jasmine, Karma
Testing PHP	Phpunit, Prophesize	PhpSpec, Mockery
Minificat Assets	Uglify	

Un cop en producció l'aplicació, s'ha provat amb bons resultats. La navegació és fluida, els widgets s'actualitzen de forma ràpida, la interacció amb els elements es bona. Els termes de resposta, té un temps de generació d'entre 70 i 130 ms sense aplicar una caché externa com Varnish.

Això no és crític per la navegació, però si per parts crítiques del codi com la petició de sincronització. Fan que l'aplicació pugui caure quan es produeixen relativament poques peticions simultànies (mig miler és el màxim de peticions per segon).

Al fer una traça amb **xdebug**, s'observa que és massa lenta la petició (el nostre codi és menys d'un 1 %). Una possible millora seria utilitzar un microframework per gestionar aquestes peticions a l'API i reduir el overhead. També s'hauria de configurar les capçaleres HTTP en les respostes per no fer més peticions del compte.

Figura 10: Traça d'una petició a producció



9. Planificació

S'ha realitzat un diagrama de Gannt que té 2 parts ben diferenciades. La primera és la memòria, que s'ha subdividit en les diferents entregues. És una tasca continua per tant ha estat activa durant tot el projecte. La segona part se subdivideix en **Investigació** (fer algunes proves per si hi ha alternatives tecnològiques interessants a les «per defecte»), **creació d'entorns** (configuració poder treballar de forma còmoda en el projecte i per poder presentar el projecte), **Kick off de programació** (configuració dels frameworks i eines addicionals que ens garantiran una bona base d'inici). El següent pas era crític perquè consistia a **dissenyar el model**, gran part de l'èxit o fracàs depèn de les decisions que es prenguin en aquesta fase. La fase de **programació** plasma el model en el codi i segueix estàndards de qualitat rigorosos per evitar deute tècnic. Seguidament hem realitzat la **part frontend** (maquetació i connexió del navegador amb les dades del model del backend). Com a última fase s'ha creat una zona d'administració per crear els panells.

En la següent figura es pot veure el diagrama de Gannt de la planificació que s'ha dut a terme durant el projecte, sense contratemps.

Figura 11: Diagrama gantt (part I)

Nombre	Fecha de inicio	Fecha de fin
Memoria	25/09/15	7/01/16
Memoria Part 1	25/09/15	29/09/15
Memoria Part 2	30/09/15	29/10/15
Memoria Part 3	30/10/15	14/12/15
Memoria Final	15/12/15	7/01/16
Investigació tecnologies	1/10/15	6/10/15
Cerca de frameworks back i frontend	1/10/15	5/10/15
Cerca de layouts	6/10/15	6/10/15
Creació d'entorns	30/09/15	7/10/15
Creació d'entorn de desenvolupament (vagrant)	30/09/15	2/10/15
Creació d'entorn de producció (digital ocean)	5/10/15	7/10/15
Kick Off Programació	6/10/15	7/10/15
Instal·lació i configuració framework backend	6/10/15	6/10/15
Configuració frameword frontend	7/10/15	7/10/15
Configuració paquets devel (phpunit, hooks...)	7/10/15	7/10/15
Fase Disseny	2/10/15	6/10/15
Disseny del model	2/10/15	6/10/15
Fase Programació II	22/10/15	24/11/15
Creació del model backend	22/10/15	24/11/15
Crear repositoris / infrastructura	22/10/15	30/10/15
Programar domini	2/11/15	18/11/15
Testejar domini	19/11/15	24/11/15
Creació model frontend	25/11/15	8/12/15
Connectar frontend amb backend	25/11/15	8/12/15
Maquetació	9/12/15	25/12/15
Aplicar disseny	9/12/15	25/12/15
Fase Programació III	28/12/15	8/01/16
Backend gestionar boards	28/12/15	8/01/16
Fase d'entrega	11/01/16	12/01/16
Creación d'un board de proves	11/01/16	11/01/16
Integració amb dades reals	12/01/16	12/01/16

El projecte s'ha dut a terme dintre de les previsions, no hi ha hagut desviacions.

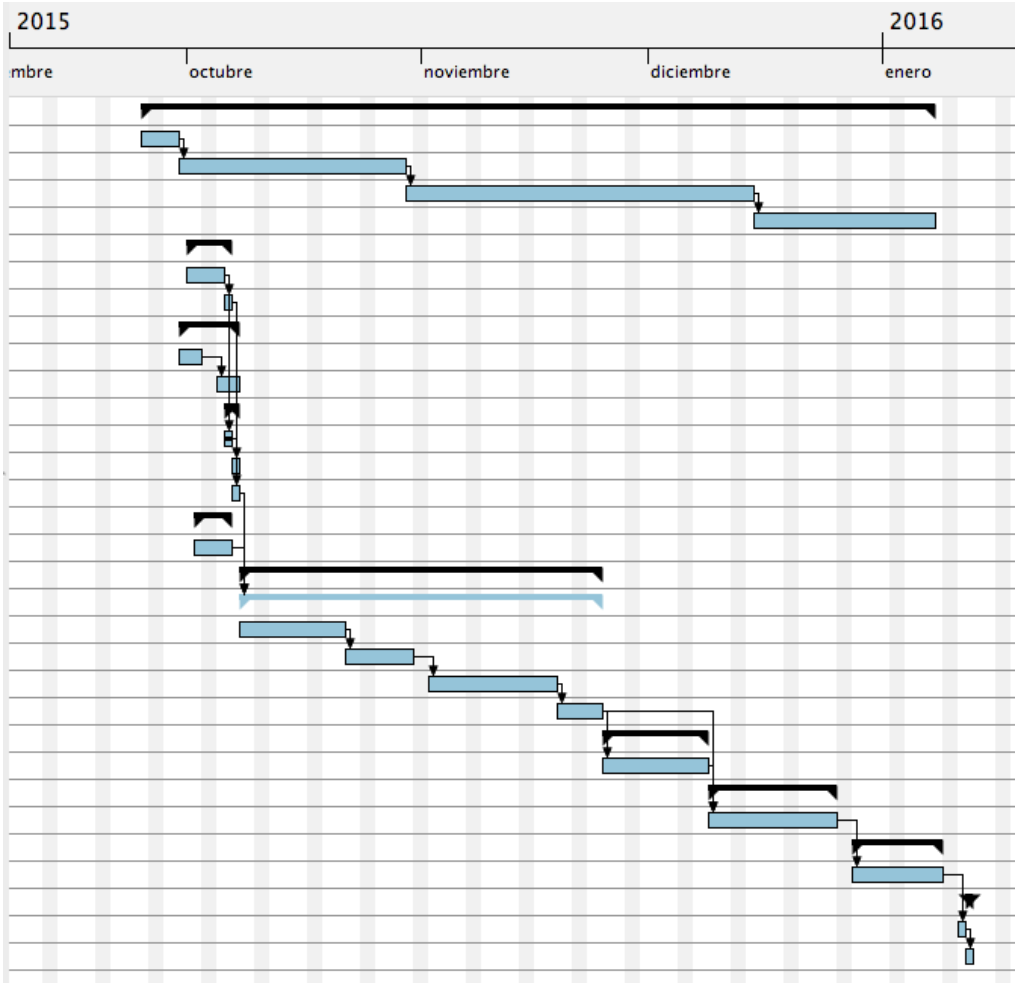


Figura 12: Diagrama gantt part II

En el següent punt de la memòria, es detalla el desenvolupament del projecte amb més nivell de detall.

10. Procés de treball

El projecte es va desenvolupant seguint les previsions inicials. Està subdividit en 4 parts ben diferenciades però interrelacionades (en l' **Annex 2** d'aquesta memòria es centra a nivell de codi) que són les següents:

Backend intern

És la part que s'encarrega de la lògica de domini, casos d'ús, classes auxiliars, repositoris i els controladors Web. El disseny per capes ens permet de tenir una clara separació i és fàcilment extensible. Aquesta ha sigut la primera part que s'ha desenvolupat seguint els diagrames de casos d'ús, creant el domini i testejant-lo.

Backend API

Hem implementat una **API REST** per desacoblar els recursos dels controladors Web. Fa servir els mateixos casos d'ús del backend intern. El fet d'exposar la API permet que els recursos puguin ser consumits des de qualsevol tipus de tecnologia que accepti peticions HTTP.

Frontend Javascript

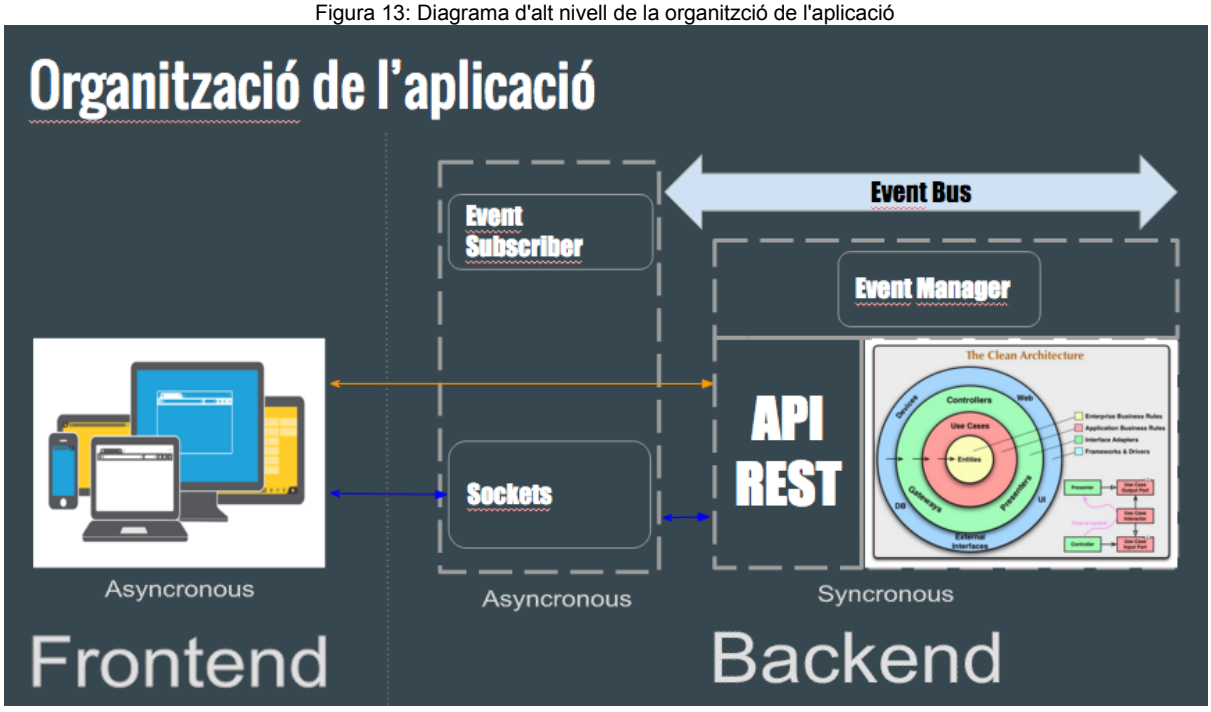
Ens permet dotar l'aplicació d'interacció sense haver de recarregar el navegador. El codi javascript al ser basat en esdeveniments, s'ha de pensar d'una altra manera i els elements han d'estar totalment sincronitzats.

En aquesta fase s'han anat afegint nous recursos a la API que eren necessaris per la interacció amb la UX com pot ser posicionar els widgets.

Frontend Disseny

Permet maquetar el frontend i donar una interfície més agradable i usable per l'usuari d'un navegador. La web és responsive i permet una correcta visualització en diferents tipus de dispositius.

Figura 13: Diagrama d'alt nivell de la organitzció de l'aplicació



El procés de treball desenvolupat durant el projecte ha seguit la planificació.

Primer de tot ens vàrem centrar a disposar una base backend molt avançada i amb alta cobertura de tests per validar-la. La premissa era que la lògica de negoci és independent de la visualització.

Un cop el backend estava avançat, es van començar a crear rutes dintre dels controladors Web. Ens va permetre tenir una visió de la web, això si sense cap tipus de disseny ni interacció frontend. Es justifica perquè volíem disposar d'entregables encara que fossin versions poc avançades. La web era funcional.

Quan ja teníem l'estructura de les rutes i l'esquelet de l'aplicació, es va començar a dotar-lo de dinamisme al frontend. En aquest punt, ja estava previst, calia disposar d'una API que desacoblés les dades dels controladors Web. Possiblement hauria estat millor fer abans l' API que els controladors Web, tot i així no ha sigut problemàtic la migració.

Amb l'API REST disponible, la part frontend podia fer accions sobre els recursos del backend. S'ha dedicat bastants esforços a dotar d'interacció a la web, encara calia polir la gestió de dashboards i creació de widgets en aquesta fase.

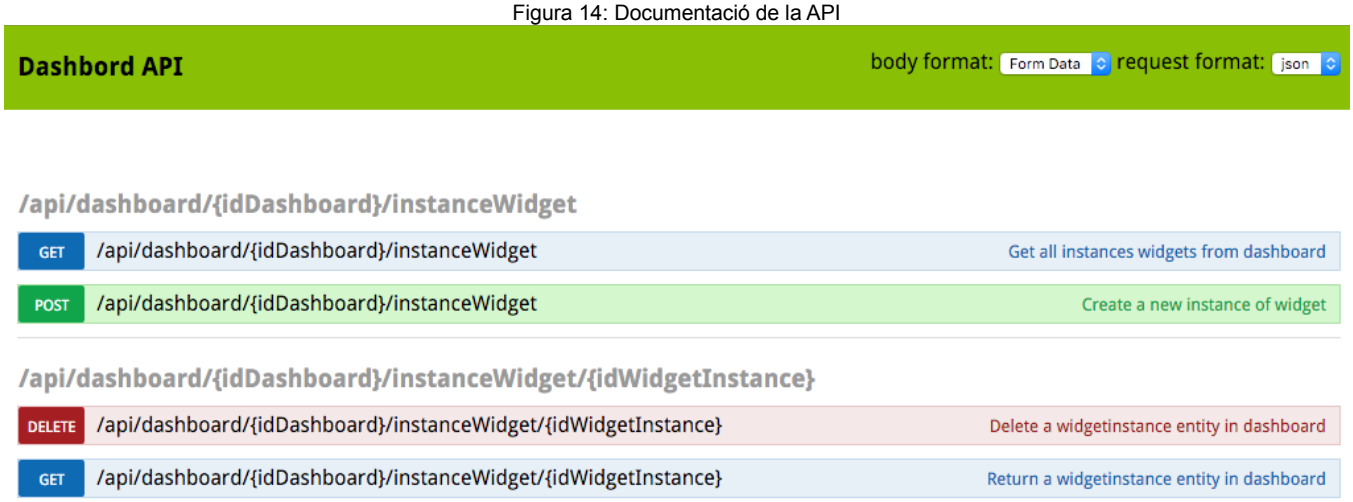
En la següent fase es va començar a maquetar la web i a dotar d'una interfície d'usuaris més simple. Es va seguir els dissenys mocks de l'apartat 13.

En paral·lel es va anar millorant el codi javascript per tenir-lo millor organitzat. En les últimes setmanes del projecte es va afegir la funcionalitat d'actualitzacions en temps real, que permet sincronitzar les dades de les instàncies de widgets (inclús entre diferents finestres web) per la qual cosa va ser necessari un element de sincronització. Aquest element és un backend asíncron que s'encarrega de comunicar-se amb la part front end, mitjançant sockets, i amb l'API en resposta a certs esdeniments.

11. APIs utilitzades

L'aplicació utilitza una **API REST** interna i està documentada de forma bastant automatitzada. Disposa d'un sandbox on poder realitzar peticions de forma independent de l'aplicació.

Figura 14: Documentació de la API



No s'han utilitzat API externes, tot i que el sistema està preparat per acceptar-les en un futur. Actualment, els widgets del dashboard poden agafar diferents fonts de dades, siguin dades internes d'un repositori com externes a través de peticions **cURL's** o similars.

12. Diagrames

Casos d'ús

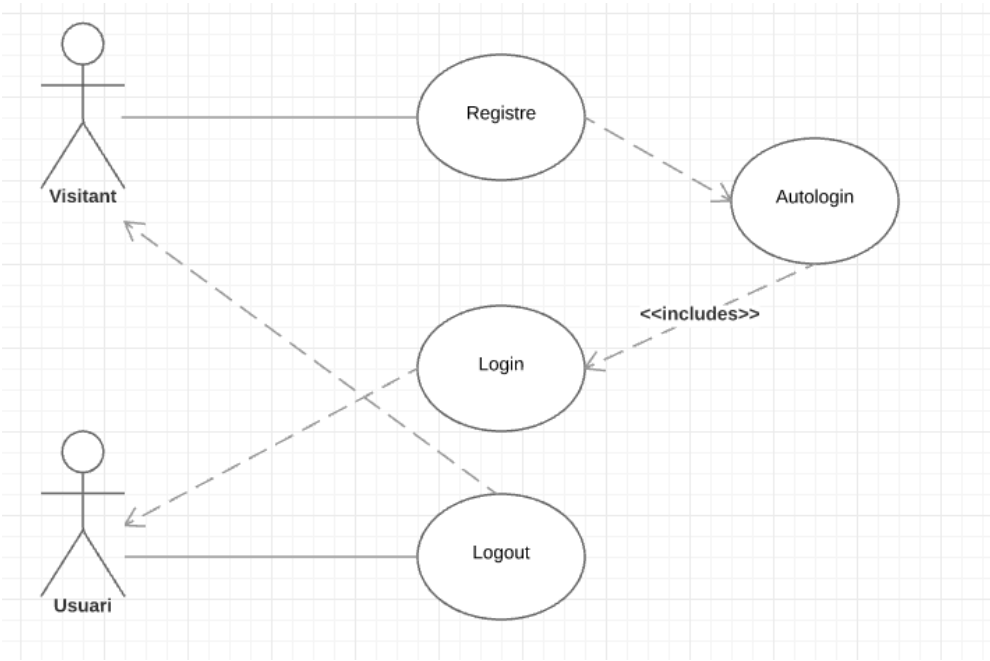
Són la base de l'aplicació backend, s'han de dissenyar en una primera fase del projecte. Cada cas d' ús ha de realitzar només una acció.

Hem dividit els casos d'ús en 2 diagrames. En el primer apareixen les interaccions abans d'entrar al panell d'administració dashboards. Típicament durant el procés de registre i login. Com es pot veure hi ha 2 perfils d'usuari. El **visitant** es pot registrar i fer login. Un cop fet això es converteix en **usuari** que pot fer logout, recuperar o canviar la contrasenya.

Figura 15: Diagrama de casos d'ús en UML d'accions en el admin

Cas d'ús	Actors	Flux següent	Precondicions	Estat final
Registre	Visitant	Autologin	Usuari no estigui registrat	Enviar correu electrònic de registre KO: Missatge d'error i tornar al formulari de registre
Autologin	Visitant (inicial). Usuari (final)	----	Usuari registrat però no logat, desde la confirmació	El visitant el loga com a Usuari KO: Missatge d'error i tornar a la home page
Login	Visitant (inicial). Usuari o Admin (final)	----	Usuari registrat però no logat, des del menú de navegació	El visitant el loga com a Usuari KO: Missatge d'error i tornar al formulari de login
Logout	Usuari o Admin (inicial) / Visitant(final)	----	Usuari logat	L'usuari tanca la sessió es converteix en visitant KO: Missatge d'error i tornar a la home page

Figura 16: Diagrama de casos d'ús en UML de rols



Es podria estendre amb més funcionalitat i rols, de moment hem deixat les bàsiques com estava planificat.

En el segon diagrama de casos d'ús es presenten els mateixos actors i ens mostra les interaccions d'aquests quan entren a la zona d'administració i visualització de dashboars. El **visitant** només pot veure dashboards públics, **l'usuari** pot crear, modificar/configurar i eliminar tant els seus dashboards com les instàncies dels widgets que el componen.

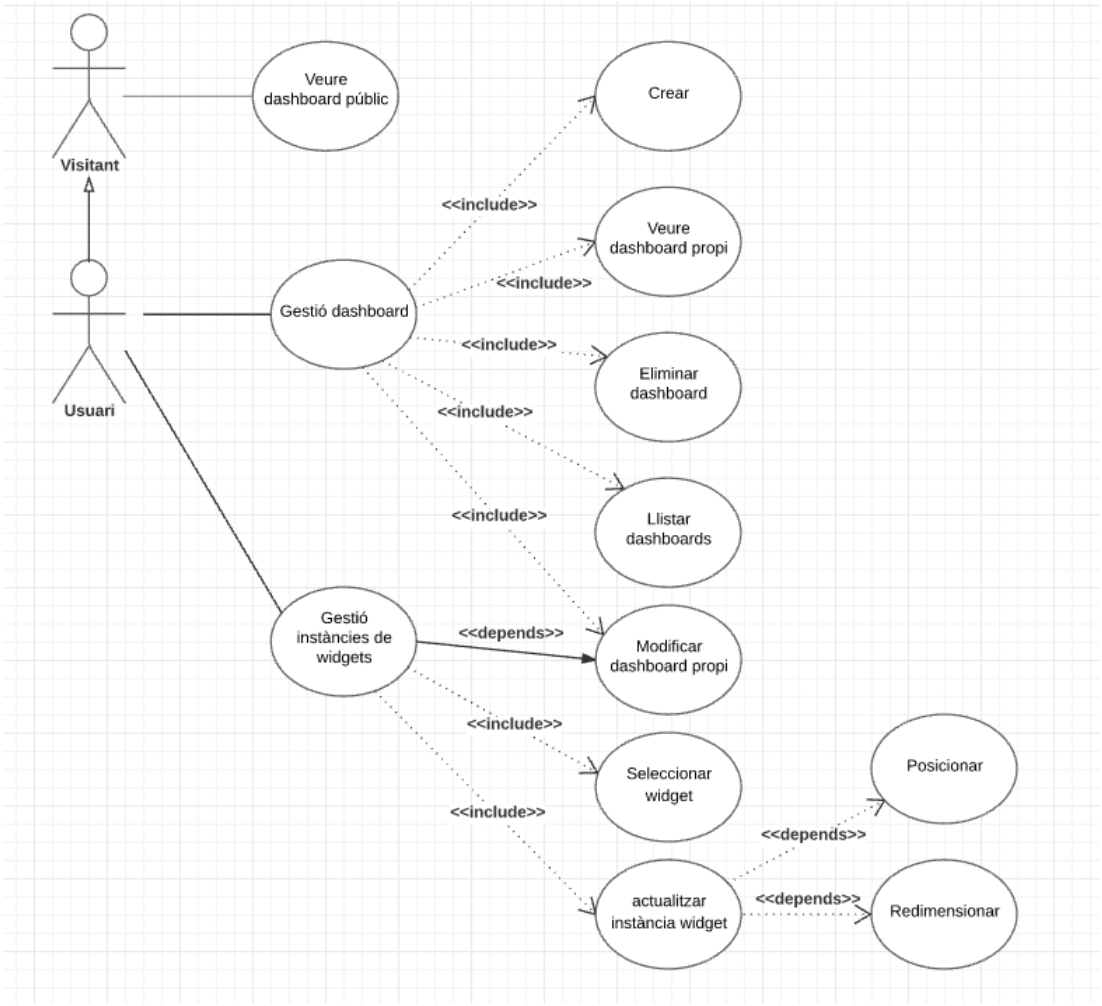


Figura 18: Diagrama de casos d'ús en UML d'accions en el admin

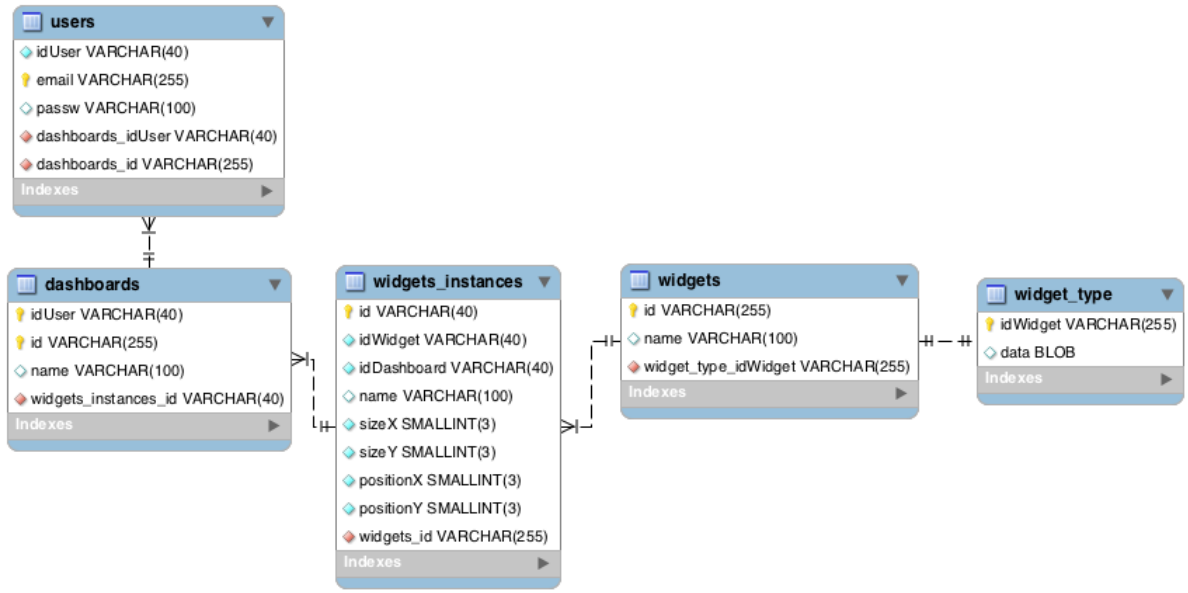
Cas d'ús	Actors	Flux	Precondicions	Estat final
Veure dashboard públic	Visitant / Usuari	–	El dashboard ha d'estar marcat com a públic	No hi ha modificació de l'estat
Crear dashboard	Usuari	–	Cap	Es crea el dashboard, el nom pot estar repetit. KO → Mostrar missatge d'error a l'usuari
Veure dashboard propi	Usuari	–	El dashboard ha de ser del propi usuari	No hi ha modificació de l'estat
Eliminar dashboard	Usuari	–	El dashboard ha de ser del propi usuari.	Cap dada associada al dashboard, a excepció de l'usuari es manté. KO → Mostrar missatge d'error a l'usuari
Llistar dashboard	Usuari	–	El dashboard ha de ser del propi usuari	No hi ha modificació de l'estat
Modificació dashboard	Usuari	Gestió instàncies de widgets	El dashboard ha de ser del propi usuari i estar en mode edició	Es modifica l'estat del dashboard i dels elements relacionats KO → Mostrar missatge d'error a l'usuari
Seleccionar widget	Usuari	Actualitzar instància de widget o ---	El widget ha d'estar actiu	No hi ha modificació de l'estat (fins que vagi al cas d'ús següent)
Actualitzar instància de widget	Usuari	Posicionar o Redimensionar	El dashboard ha de ser del propi usuari i el widget ha d'estar actiu	Es modifica la instància de widget KO → Mostrar missatge d'error a l'usuari
Posicionar instància de widget	Usuari	–	El dashboard ha de ser del propi usuari, estar en mode edició i que la instància de widget existeixi	Es modifica la posició de la instància de widget KO → Missatge d'error a l'usuari
Redimensionar instància de widget	Usuari	–	El dashboard ha de ser del propi usuari, estar en mode edició i que la instància de widget existeixi	Es modifica la mida de la instància de widget KO → Missatge d'error a l'usuari

Base de dades

El diagrama de base de dades consta de 5 taules. Que detallem a continuació:

- **User** → Conté la informació de l'usuari registrat
- **Dashboard** → Un dashboard està relacionat amb un usuari i disposa d'instàncies de widgets. Pot indicar si un dashboard és públic o privat
- **Widget** → És una taula mestre de widgets. És l'element que composa el dashboard per mostrar els KPI, ha de ser d'un tipus determinat
- **WidgetType** → Els widgets són de diferent tipus i per tant tenen fonts de dades diferents i formes de visualitzar les dades diferents. En el camp **data** es serialitzen totes les dades específiques del tipus (per exemple la URL d'on extreure les dades). El motiu és que la informació pot ser heterogènia depenent de la naturalesa de la font de dades, a més així seria més fàcil d'adaptar a un repositori NoSql o similar.
- **Instàncies de Widget** → En un dashboard i poden haver 1 o més widgets d'un determinat tipus. La instància representa aquest widget en una posició i en un panell determinat.

Figura 19: Diagrama de base de dades Mysql de l'aplicació



Aquest diagrama de base de dades, s'ha extret directament de la base de dades amb enginyeria inversa. Per tant representa l'última versió de la mateixa en el moment d'escriure la memòria.

13. Prototips

El prototipus s'han desenvolupat utilitzant l'eina online **moqups**. Es van realitzar en una primera etapa de desenvolupament del projecte i poden diferir en certs elements del resultat final.

Figura 20: Registre d'usuari pas 1

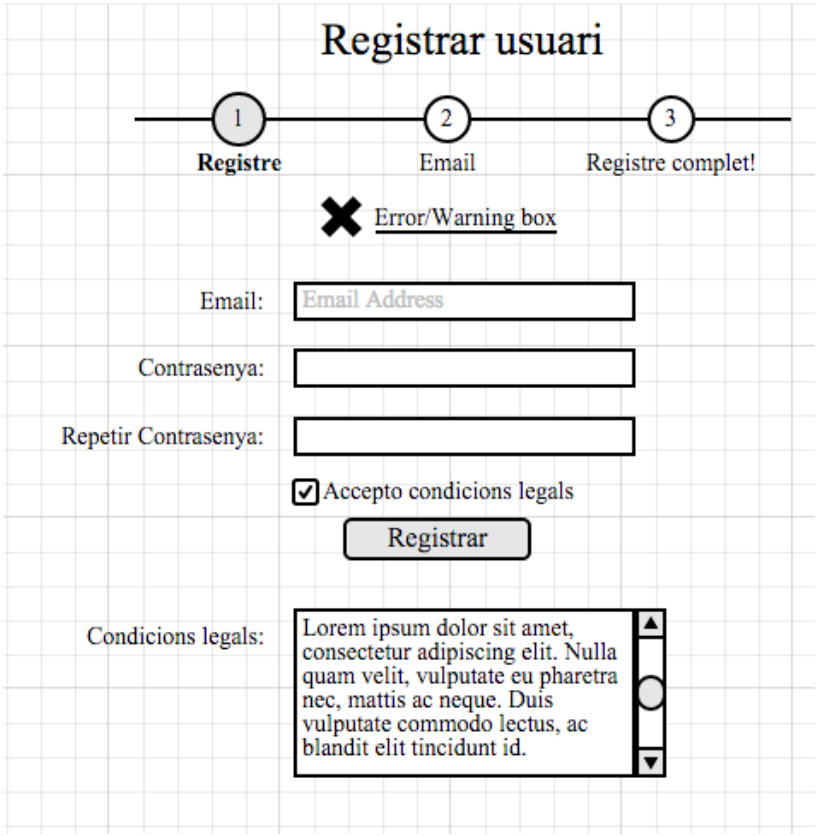


Figura 21: Registre d'usuari pas 2

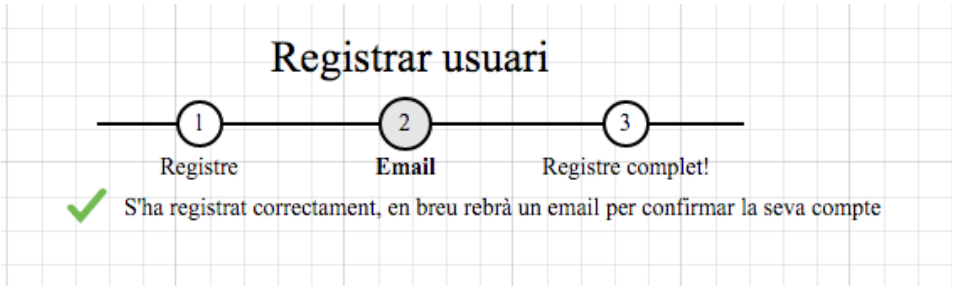


Figura 22: Registre d'usuari pas 3

En els **mockups** de registre de la pàgina anterior, s'ha dissenyat la UX pensant en accessibilitat. Hi ha 2 elements a destacar:

a) Element superior informatiu del pas actual.

Aquest element mostra a l'usuari en quin pas del procés es troba. Ajuda a l'usuari a comprendre que no és suficient anar al pas 2, sinó que cal un pas extra de confirmació.

1) **Registre** → Formulari on introduir les dades del registre

2) **E-mail** → Pàgina on t'explica que has rebut un correu electrònic per confirmar la veracitat del compte de correu electrònic.

3) **Registre final** → Indica a l'usuari que ja disposa d'un usuari vàlid i que pot fer un autologin, per entrar en el seu panell de creació de dashboards.

b) Missatges d'error o avís.

Quan l'usuari introdueix un camp de forma incorrecta, se li avisarà de què les dades no són correctes. També succeeix quan hi ha algun error temporal en el servidor. Se li afegeix una icona de creu quan va malament i de *tick* verd quan tot va bé.

En el següent mockup, es mostra el procés de login, té l' opció de recordar la contrasenya:

Figura 23: Mockup de login

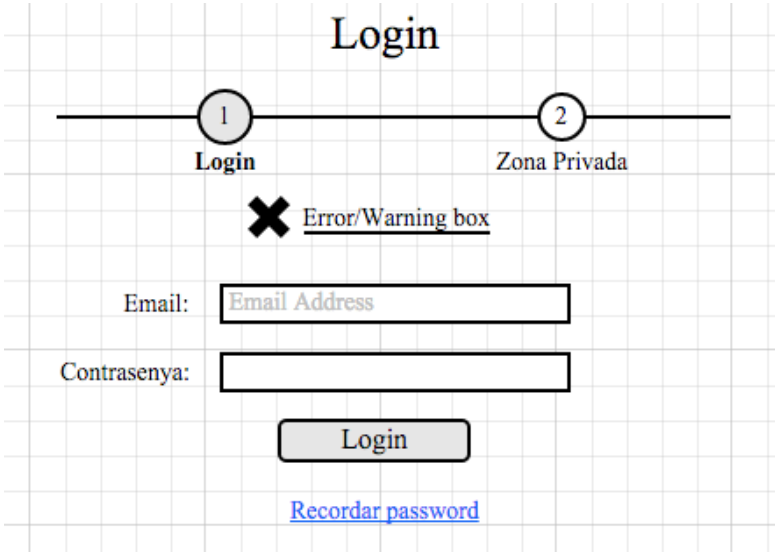
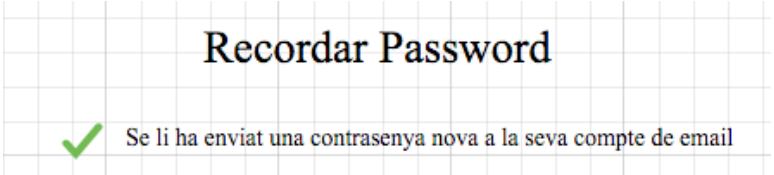


Figura 24: Mockup missatge de Recordar password



En la següent imatge mostrarem el mockup de la zona d'administració del dashboard.

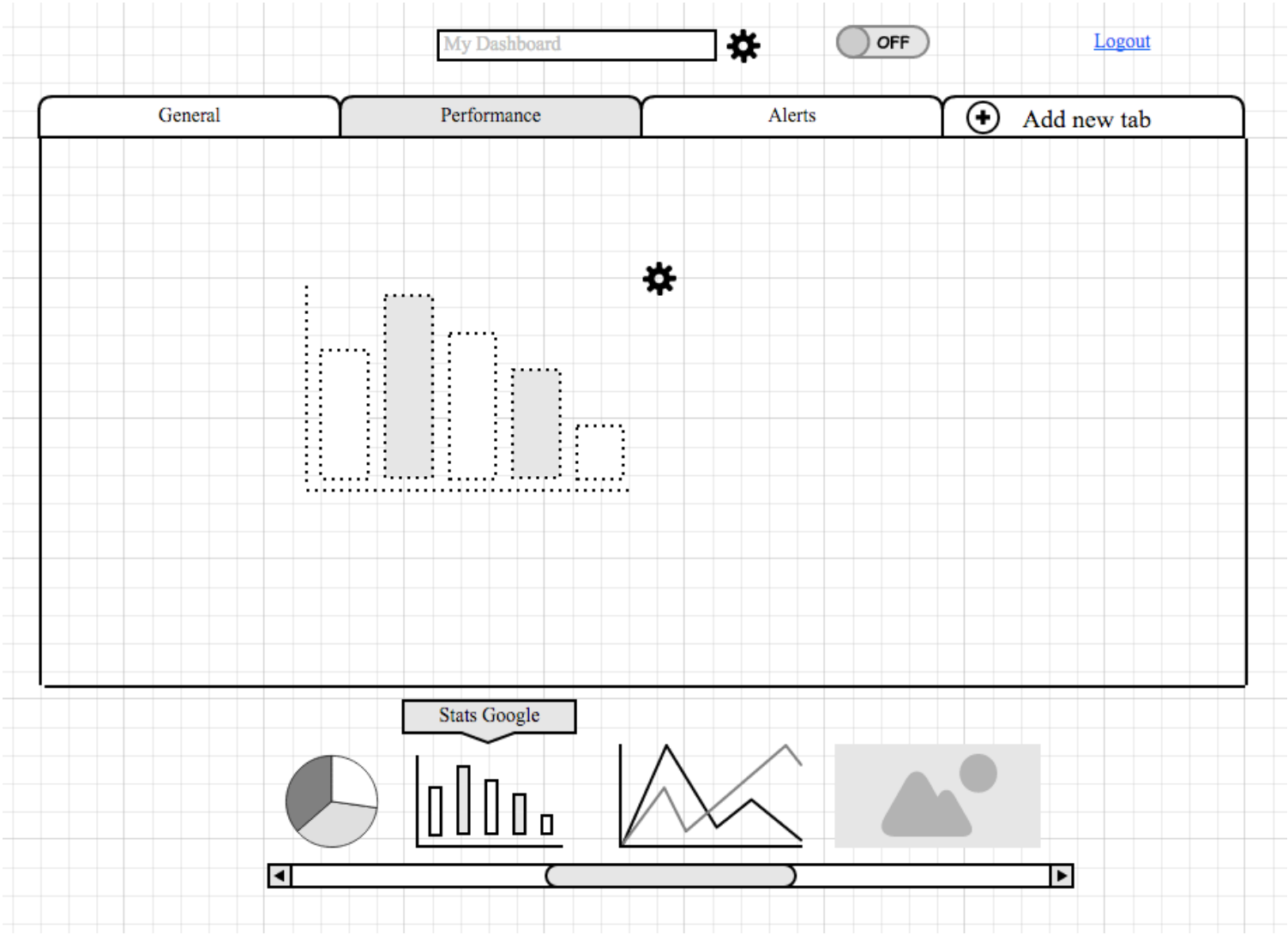


Figura 25: Mockup disseny gestió de dashboard i catàleg de widgets

Es pot observar els següents elements:

1) A la part superior hi ha un textbox amb el nom del dashboard que es pot editar. Té una icona per poder configurar la visibilitat del mateix.

2) Al costat hi ha un botó ON/OFF que representaria quan el panell és editable (no apareixeria si el dashboard és públic).

3) En les pestanyes, hi ha l'opció d'afegir nova pestanya. N'hi haurà un màxim de 5 per defecte

4) A la part inferior hi ha el catàleg de widgets, que té un petit diàleg que indica el nom del mateix.

5) Els widgets es poden instanciar en el dashboard. És molt fàcil només cal arrossegar a la part central, en qualsevol lloc. En el moment de crear aquesta instància de widget, s'haurà de posicionar i seguidament configurar.

Els widgets amb algun problema de mètriques, creixeran en mida com es pot veure en les següents imatges :

Figura 26: Mockup disseny dashboard sense problemes

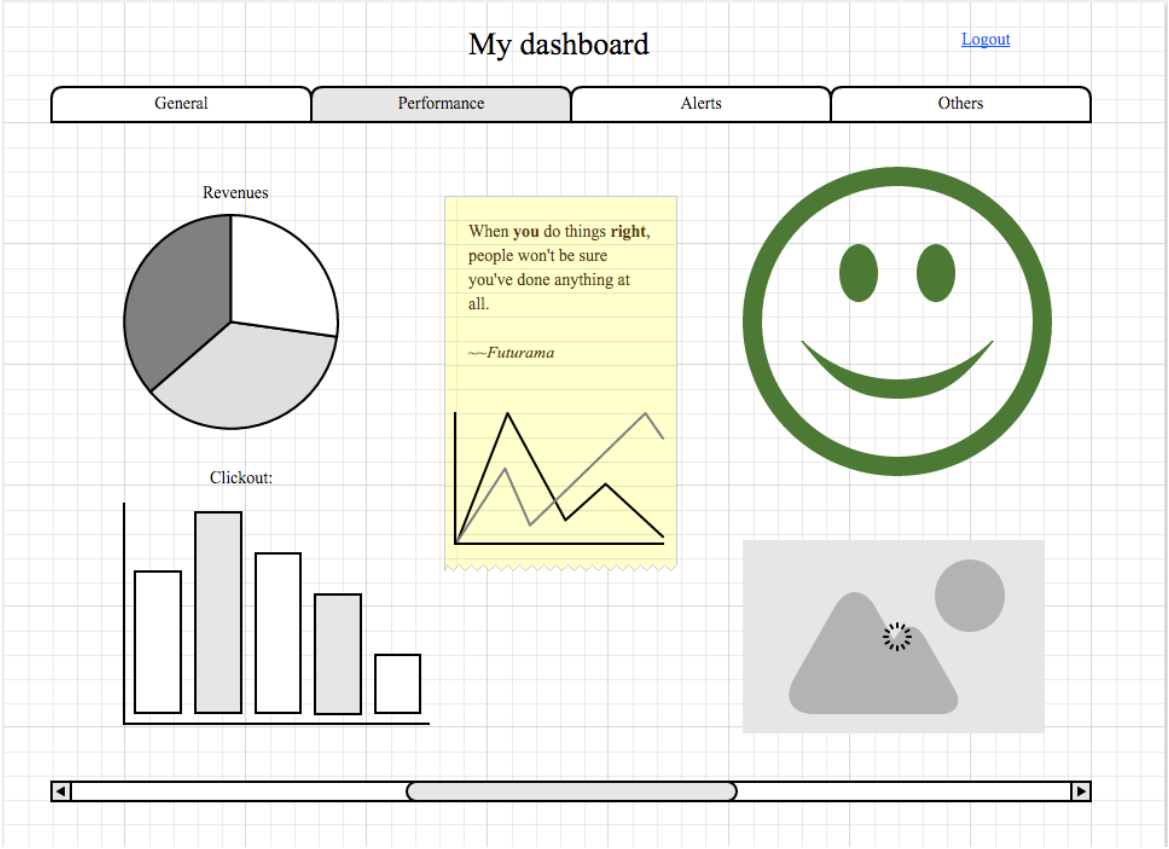
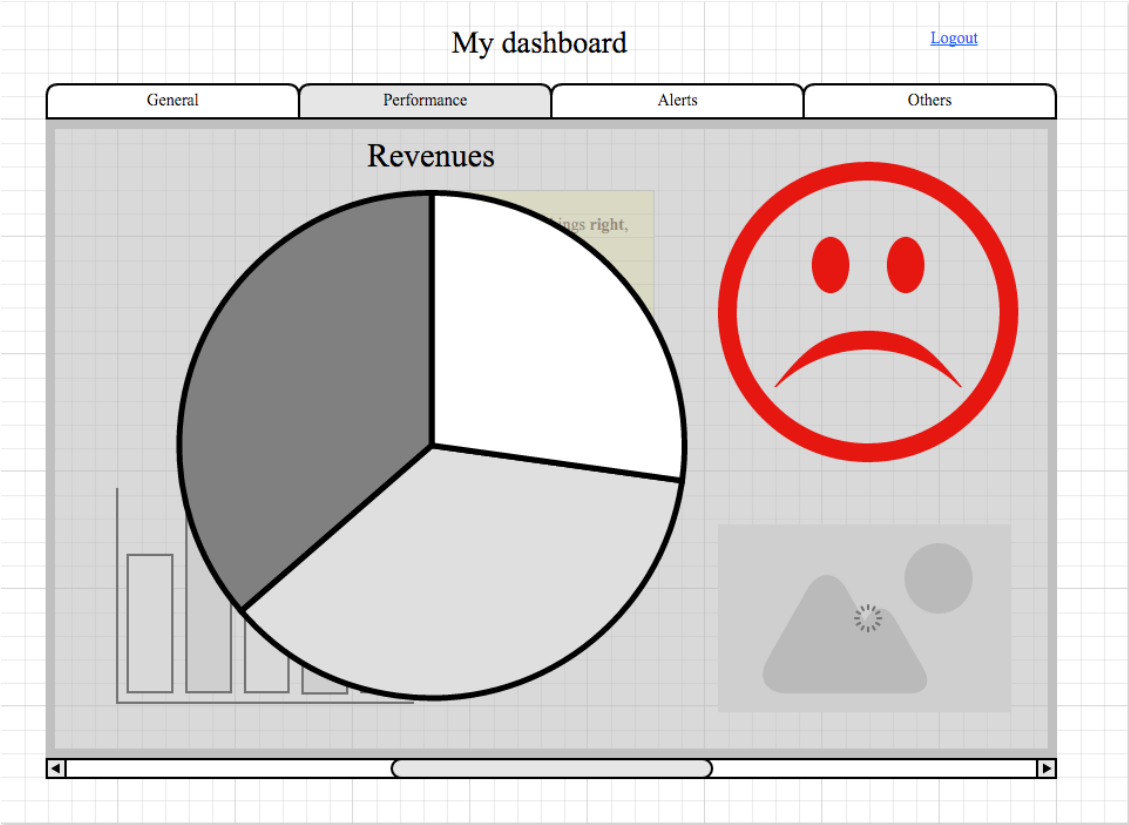


Figura 27: Mockup disseny dashboard amb problema a la métrica de revenues



NOTA: En la versió final hi ha diferències amb respecte els mockups inicials (es poden veure en l'apartat 16). La més significativa és que els widgets no creixen quan hi ha problemes, es marca amb una etiqueta. A vegades també és perquè la funcionalitat està prevista per una següent fase.

14. Perfils d’usuari

L'aplicació està pensada per aquests tipus d'usuari:

- **Project manager / Sales / Business ...** → Son perfils que coneixen molt bé el seu àmbit d'acció (per exemple el producte que venen, el mercat, dominen conceptes com retorns d'inversió, clickout, CPC, CPA …) Però normalment tenen un coneixement tècnic limitat, potser arribant a fer consultes SQL o fer operacions amb excel. Aquest perfil requereix una interfície d'usuari fàcil d'utilitzar, ràpida i intuitiva. Obtenir dades en temps real pot ser fonamental per una bona comprensió de l'estat actual del negoci i per tant és important transmetre la importància de fer servir la nostra eina.

- **Data analist/science** → Persona amb coneixements profunds d'anàlisis de dades, sol tenir coneixements d'eines tècniques complexes d'àrees com el bigdata o analítica web. En ser un perfil mig entre negoci i tècnic té unes necessitats especials. Potser no li fa falta una resposta immediata de les dades, li interessa més les tendències per veure les evolucions. També està interessat a tenir molts graus de les dades (per exemple timestamp, població, acció de l'usuari…) És a dir li podria interessar tenir un panell amb moltes dades relacionades.

- **Developper** → Un perfil amb un nivell tècnic alt però normalment amb coneixement limitat del negoci, en les empreses tecnològiques aquest rol és imprescindible i a vegades pot estar interessat en temes de negoci. Però normalment la nostra eina la farà servir per obtenir gràfiques i informació actualitzada sobre dades tècniques. Valora especialment una interfície ràpida i altament configurable. Potser no valora tant un aspecte atractiu o intuïtiu perquè està acostumat a utilitzar interfícies poc elaborades, que no poc potents, com poden ser els terminals.

- **Usuari nivell intermig** → És un usuari que té coneixements tècnics mitjans, suficients per disposar d'un blog o pàgina web senzilla. Necessita l'aplicació per veure dades bàsiques com n.º de visites. Per tant necessita poder accedir de forma fàcil al panell de control i a la creació de dashboards. Només utilitzarà els widgets, no en crearà de nous.

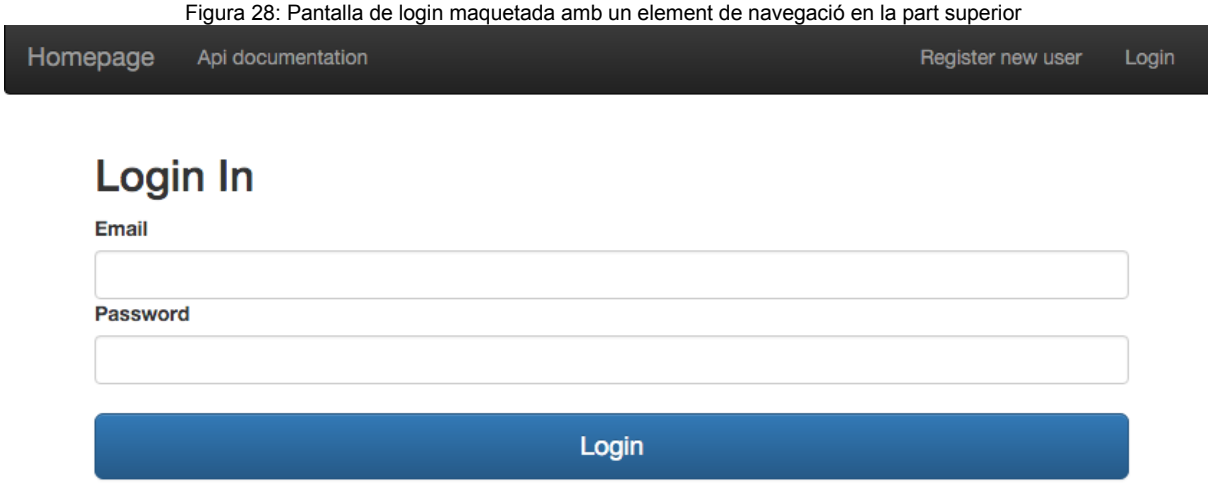
	Nivell Tècnic	N.Negoci	Dashboard	Widget
Project manager / Sales / Business	Mitg	Alt	Multiples Configuració bàsica	Utilitzar widgets Real time
Data analist/science	Alt	Mitg/Alt	Multiples Configuració avançada	Creació de nous tipus Normal / Real time
Developper	Alt/Molt Alt	Baix/Mitg	1 Configuració avançada	Creació de nous tipus Normal
Usuari nivell intermitg	Mitg	---	1 Configuració bàsica	Utilitzar widgets Normal

15. Usabilitat/UX

En la versió final, la web té un nivell d'usabilitat bastant alt. Es disposa d'una barra superior que ajuda a l'usuari en la navegació. En la zona d'administració hi ha llistats amb icones per veure el dashboard en mode gestió i en mode visualització. També hi ha feedback en els formularis, quan una dada no és correcte es mostra un avís.

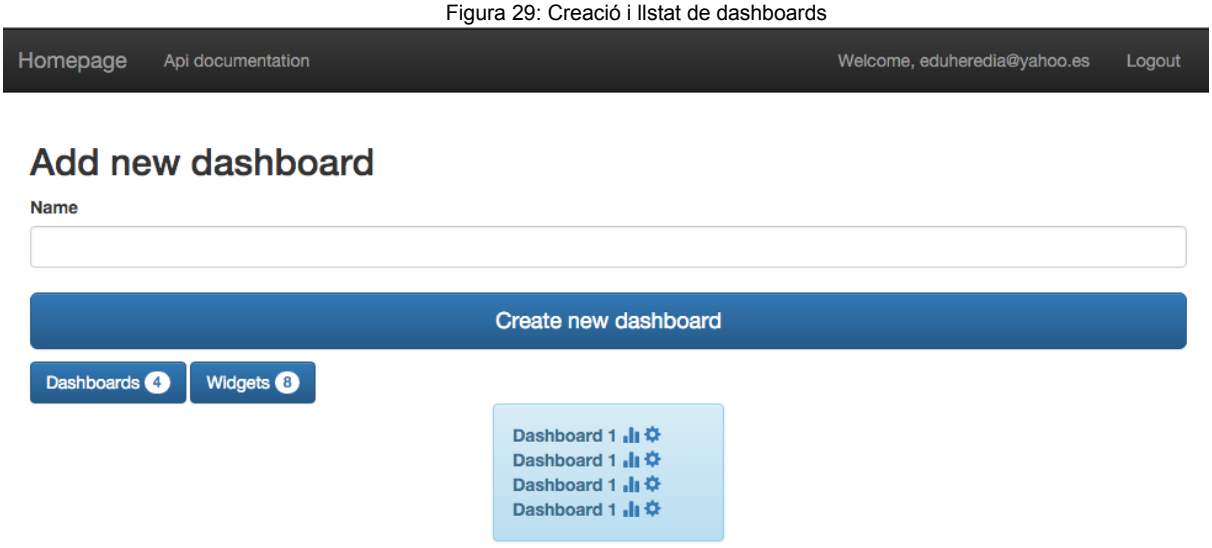
El fet de crear una instància de widget és tan intuitiu com arrossegar-lo al dashboard.

Figura 28: Pantalla de login maquetada amb un element de navegació en la part superior



En la versió final, els dissenys **no seran exactament** igual que els prototipus anteriorment descrits. A mesura que avança el projecte es van iterant i s'observa que organitzant els elements d'un altre forma la web queda més usable. Un exemple és el següent:

Figura 29: Creació i llstat de dashboards



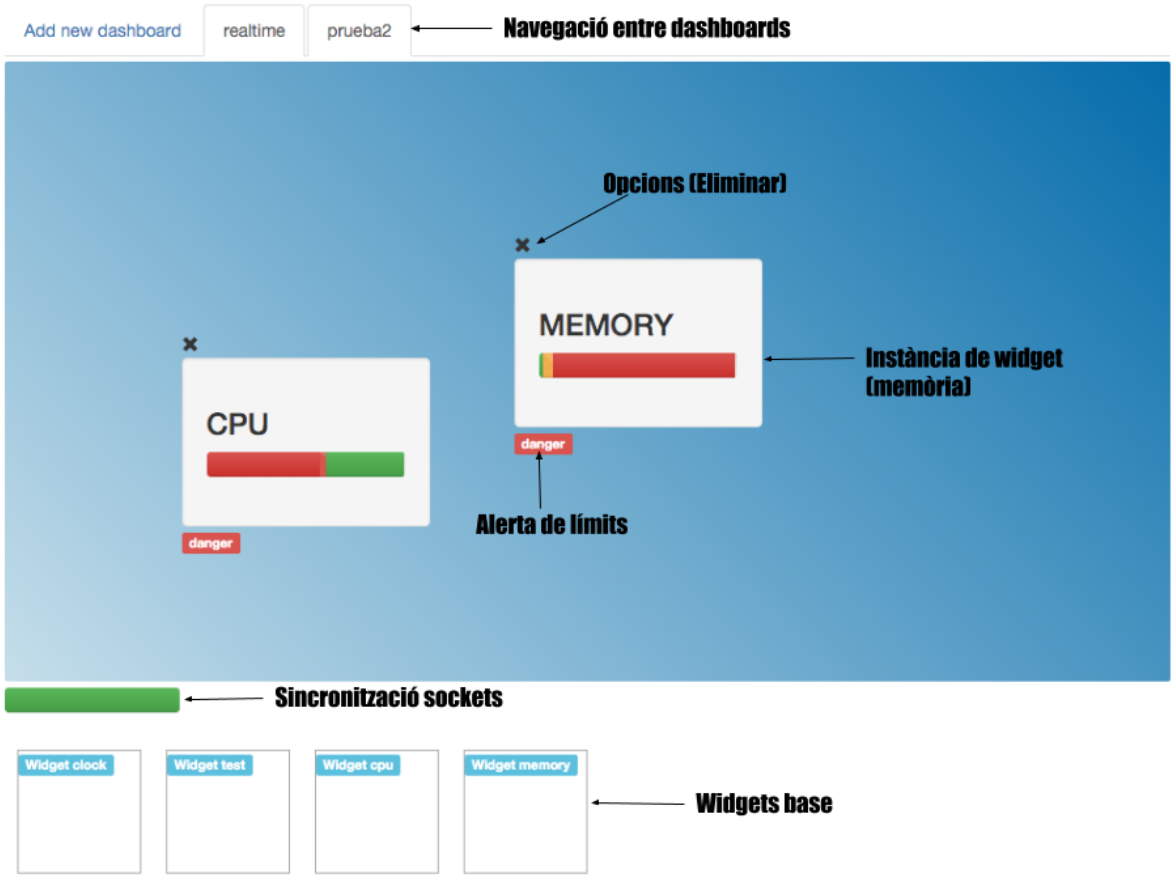
Com es pot observar s'ha posat en una pàgina separada. Conceptualment és més correcte que la creació de dashboards sigui una tasca independent a la configuració del propi dashboard. Queda més clar per l'usuari si cada pantalla té 1 o 2 accions relacionades. En aquest cas de la figura anterior, es pot crear un nou dashboard, i es poden llistar informacions relacionades.

Des de la pàgina de creació de dashboards es pot accedir a la pàgina de modificació, que seria un nivell més de profunditat.

Totes les pantalles estan en una versió final. Només es modificaran si tenen problemes d'usabilitat o conceptualment hi ha elements que es podrien millorar. Per posar un exemple clar, el botó de llistat de widgets base no té gaire utilitat ara mateix en la pantalla de creació de dashboards, però si té tot el sentit del món en la pantalla de configuració de dashboards. És candidat a moure's a la pantalla corresponent, la idea és deixar la mínima funcionalitat imprescindible en cada pantalla.

És un procés iteratiu en el qual t'has de posar en el lloc de l'usuari i facilitar-li la vida. L'èxit o el fracàs d'un producte té a veure amb l'experiència d'usuari. Per últim mostrem la pantalla d'edició de dashboards, que és la més interessant.

Figura 31: Edició de dashboards i creació d'instancies de widgets



En el dashboard es veuen 2 instàncies de widgets que són un rellotge (que s'actualitza a intervals del servidor) i un text que de moment és estàtic. A la part inferior es veu el llistat de widgets «base» que s'arrosseguen al dashboard superior.

Les instàncies de widgets, s'actualitzen en temps real. Vol dir que obtenen dades noves quan és notificat pel servidor. Es poden eliminar, posicionar i canviar la mida. També s'actualitza el dashboard quan s'han produït modificacions com eliminar una instància de widget.

16. Seguretat

Aquest projecte s'ha encarat com un projecte real que tindrà recorregut més enllà de la presentació del mateix. Per tant ha de ser segur perquè podrà ser usat en entorns empresarials. En l'àmbit de l'aplicació no hi hagi errors típics com poden ser la injecció de SQL, HTML/javascript Injection, atacs de força bruta …

A nivell d'aplicació

SQL injection → La base de dades utilitzada es MySQL, farem servir una capa de doctrine anomenada **DBAL (**que està per sobre la capa d'abstracció **PDO)**. Aquesta capa permet preparar les dades abans d'executar la consulta i evitar la injecció SQL.

Altres tipus d'injecció → El framework Symfony 2 **filtrarà les entrades i escaparem les sortides** en les plantilles **Twig**. Així no es podrà injectar javascript en el DOM o introduir caràcters estranys.

Contrasenyes dèbils → No es guardaran les contrasenyes originals, es generarà un hash difícils de trencar.

Atacs de força bruta → És registraran events de login incorrecte per adreça IP per limitar atacs de força bruta.

A nivell de servidor

S'ha configurat el servidor per a poder allotjar l'aplicació. S'ha tingut compte de securitzar el servidor per evitar atacs externs, bàsicament s'ha configurat el següent:

IDS → S'ha activat un sistema de detecció d'intrusos, que bloqueja els intents reiterats de login a través del SSH.

Firewall → S'ha activat un firewall software anomenat **ufw** que només permet connexions HTTP i SSH**.**

Logs i rotació → S'ha activat la rotació de logs pels serveis més importants. En cas de problemes tindrem un registre.

Actualització automàtica → Hi ha un cron que actualitza els paquets a l'última versió quan són fix de seguretat. Així ens assegurem que no hi han obertures conegudes de seguretat.

Informe de seguretat → S'han instal·lat eines com Logwatch, que fan un informe diari amb totes les incidències del servidor.

```
    125.65.245.146                                  1:1
    187.130.15.137 (187-130-15-137.uninet-ide.com.mx)      1:1
    187.177.137.233 (187-177-137-233.dynamic.axtel.net)    1:1
    209.236.124.251 (209.236.124.251.tailormadeservers.com)  1:1
    218.236.113.44                                  1:1

---------- fail2ban-messages End -------------------------


---------- pam_unix Begin -----------------------

cron:
  Sessions Opened:
    root: 25 Time(s)

sshd:
  Authentication Failures:
    root (43.229.53.19): 208 Time(s)
    unknown (178.62.173.36): 186 Time(s)
    root (43.229.53.21): 80 Time(s)
    root (43.229.53.20): 32 Time(s)
    unknown (201.236.236.158): 10 Time(s)
    unknown (177.22.195.36): 6 Time(s)
    unknown (187.141.82.87): 6 Time(s)
    unknown (190.7.218.26): 6 Time(s)
    unknown (190.85.103.250): 6 Time(s)
    unknown (190.95.191.67): 6 Time(s)
    unknown (201.151.95.126): 6 Time(s)
    unknown (201.249.200.109): 6 Time(s)
    unknown (212.176.197.29): 6 Time(s)
    unknown (office.vinta.org): 6 Time(s)
    unknown (187.130.15.137): 4 Time(s)
    unknown (58.215.79.246): 4 Time(s)
```

Figura 32: Report de seguretat en el servidor, Logwatch

Per realitzar totes aquestes tasques, s'han seguit els tutorials de **digitalocean** : https://www.digitalocean.com/community/tutorials

17. Tests

En la nostra aplicació web té 3 nivells de tests:

Testeig unitari → S'ha fet tests de les classes que s'han desenvolupat, intentant obtenir la màxima cobertura possible. No es farà servir TDD (desenvolupament dirigit per test) però sí que s'ha intentat no deixar els tests pel final, s'han fet en paral·lel amb el procés de codificació. L'avantatge és que a mesura que anem desenvolupant l'aplicació qualsevol canvi que afecti el funcionament de la mateixa és detectat en temps de programació. L'aplicació resultant és més robusta i està millor documentada.

Els tests es fan amb **Phpunit** i s'utilitzarà **prophecy** com a framework per mockejar objectes.

```
eduheredia@MacBook-Air-de-Edu:~/PhpstormProjects/UOC_projecte/dashboard$ make run-php-tests
./vendor/phpunit/phpunit/phpunit --bootstrap=vendor/autoload.php -c phpunit.xml.dist
PHPUnit 4.8.20 by Sebastian Bergmann and contributors.

.....................................

Time: 946 ms, Memory: 9.25Mb

OK (39 tests, 77 assertions)
```

Testeig funcional → Actualment es testeja el correcte funcionament de l' API i d'alguns serveis bàsics amb **Mocha**. Alguna de les verificacions que es realitzen és que l' API estigui activa i que les respostes de rutes actives i invàlides sigui les esperades. Es pot veure un exemple en la següent imatge:

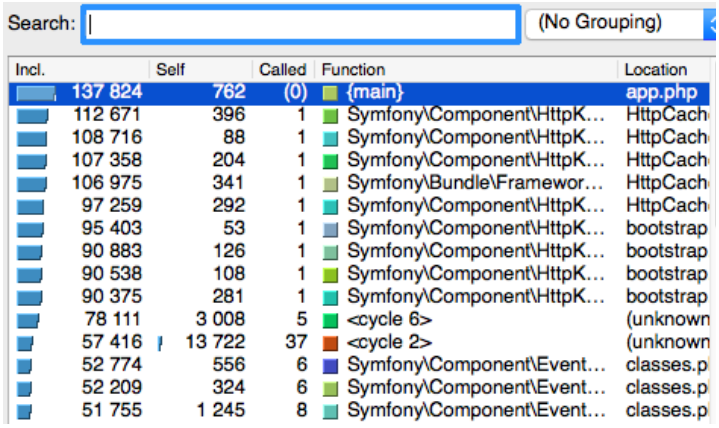
```
eduheredia@MacBook-Air-de-Edu:~/PhpstormProjects/UOC_projecte/dashboard$ make run-all-tests
mocha src/js/Tests/


  api
    Verify if api is running
      ✓ Api is up (1459ms)
    Invalid route
      ✓ /api/aaaaaaa is a invalid route (1187ms)

  api routes
    GET /api/widgets
      ✓ return all widgets (966ms)


  3 passing (4s)
```

Test amb usuaris → Es tenia la intenció de realitzar un test d'usuaris bastant ampli. Al final no ha sigut possible. S'ha intentat que fos el més usable possible però no ha pogut ser validat per un usuari objectiu.

Test de rendiment Backend→ S'ha realitzat un profiling en el servidor de producció (després s'ha deshabilitat l'extensió perquè te un overhead important). S'ha fet servir **cachegrind**, per veure les traces geneades per xdebug. S'ha observat com la majoria del temps de petició el consumeix el Framework Symfony 2. Per la web en general no és problemàtic, ja que te una resposta relativament bona (entre 70 i 120 ms) i que podria ser millor si s'afegís un sistema de caché com Varnish.

Però per peticions de sincronització de dades de les instàncies dels widgets el temps de resposta es crític, així com el nombre de les mateixes que suporta el servidor. Per tant s'arriba a la conclusió que aquesta part de sincronitzar les dades, s'hauria de fer amb un framework més lleuger (es relativament fàcil perquè el codi està desacoplat de framework) o fins i tot es podria replantejar en un altre llenguatge més ràpid com un altre *bounded context*.

Figura 35: traça de xdebug visualitzada en cachegrind



Test de rendiment frontend → S'ha fet servir la web http://www.webpagetest.org per optimitzar el frontend (per exemple minificant els CSS). Això ho farem perquè així l'experiència d'usuari sigui més satisfactòria. També s'han fet servir d'altres eines integrades amb el navegador com Yslow i les eines de desenvolupadors. S'ha intentat minimitzar el javascript però donava algun problema el compressor r.js.

18. Versions de l'aplicació

S'ha etiquetat com a versió v1.0.0 seguint la nomenclatura de versió semàntica.

Figura 36: Release v.1.0.0

Es pot accedir en el repositori privat en github sol·licitant permís d'accés. L'aplicació està deployada en la següent adreça

http://www.dashboarduoc.net/

i es requereix canviar els DNS locals afegint la següent entrada en el fitxer /etc/hosts

```
188.166.45.150 www.dashboarduoc.net
```

19. Instal·lació

Requisits d'instal·lació

Per facilitar el muntatge de l'entorn en local, s'ha utilitzat una màquina **vagrant** sobre **virtualbox**. Està pràcticament tot configurat en la màquina, llevat de les eines de frontend com npm o bower que només es requereixen si cal actualitzar dependències. En un principi, qualsevol sistema operatiu que soporti vagranti tingui un mínim de 512 MB de RAM i espai suficient, no hauria de tenir problemes.

Instruccions d'instal·lació

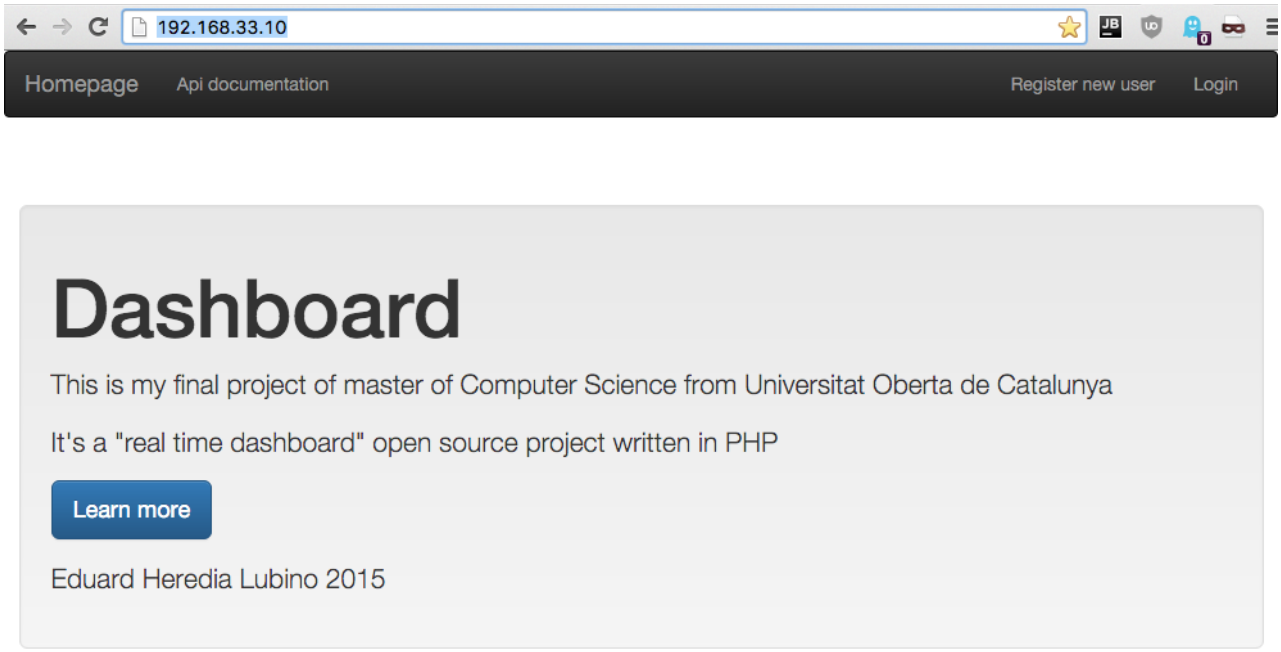
Les instruccions d'instal·lació i configuració actualitzades del projecte estan el fitxer de README.me.

Les instruccions bàsiques d'instal·lació a alt nivell d'un entorn de seríen :

- Clonar / descarregar el projecte
- Crear la màquina virtual amb vagrant (vagrant up)
- Configurar la màquina (vagrant provision)
- Crear les taules a la base de dades. Actualment es fa de forma manual entrant a la màquina. El volcat de dades està al directori info/sql.
- Accedir a la URL http://192.168.33.10/

S'hauria de veure la homepage similar a la següent figura.

Figura 37: Pàgina principal en un entorn vagrant



20. Bugs / Millores

Primer de tot dir que considerem bug a un error que es produeix en qualsevol part de l'aplicació desenvolupada i no ha sigut corregit en el moment del desenvolupament. En altres paraules, un error que estigui en la branca estable de **master** és considerat un bug.

Tots els bugs detectats, es van afegint a la secció Issues del repositori. Es van resolent veient a mesura que ho permet la planificació, depenent de la feina que implica resoldre'l i el benefici aportat.

La metodologia que es fa servir per identificar i eliminar bugs es la següent:

1. Quan es localitza un bug, es crea un nou issue a github.
2. Determinar la importància del bug.
3. Quan es vol resoldre un, es llegeix el issue i es crea una nova branca de desenvolupament.
4. Un cop solucionat, en el commit s'indica la referència del issue. Si cal, ajustar els tests si el comportament ha canviat.
5. Aquesta branca es reintegra a master i s'elimina la feature branch.
6. Es marca com resolt el bug.

Els bugs de l'aplicació localitzats son els següents:

	Prioritat	Estat actual	Descripció	Solució
No posiciona a la primera	Alta	Solucionat	En moure una instància de widget no agafa la posició correcta. Era problema de la llibreria JQueryUI.	Actualitzar llibreria jQuery i JQueryUI
Actualitzar dashboard quan es creen noves instàncies	Mitja	Solucionat	Refrescar per sincronitzar els widgets del dashboard	Creació d'event de refresc del dashboard. Modificar javascript per detectar aquest event i actual en conseqüència
Moure i posicionar widgets en mode administració	Alta	Solucionat	En mode visualització no dotar al widget de les eines d'edició (posicionar, eliminar ...)	Crear una variable per diferenciar el cas, ajustar el codi per a què es comporti diferent depenent d'aquesta variable.
Els formularis no mostren els missatges d'error	Alta	Solucionat	Donar més informació a l'usuari de l'error en el procés de login i registre	S'ha creat una feature branch, s'ha analitzat el codi. S'ha unificat el comportament del login i del registre per a què funcioni similar. Les respostes incorporen un missatge que es passa a la template. Es corregeixen els tests.
No situar widgets fora del dashboard	Alta	En procés	Els widgets no verifiquen la mida del dashboard a l'hora de posicionar-se	
Els widgets no són responsive	Mitja	Fase II	Els widgets haurien de ser relatius a la mida del dashboard	Bug que trenca sistema de posició
No funciona bé redimensionar mida	Alta	En procés	No funciona correctament, es queda bloquejat.	Bug de regressió

Abans de solucionar un bug s'ha de pensar primer **que fa actualment l'aplicació, que hauria de fer i que implica realitzar el canvi**. Un exemple clar és el bug de posicionar els widgets de forma responsive, actualment les instàncies de widgets tenen una posició i mida fixes, però en canvi el dashboard és responsive. Solucionar aquest bug, afecta altres funcionalitats i també implica modificar molta part de domini com les entitats o els repositoris. Per tant és més costós que solucionar el bug de les etiquetes en els formularis.

Si solucionar el bug aporta a negoci, s'hi dedica un *bugdet* d'hores per resoldre'l.

Figura 38: Commitejar canvis d'un issue

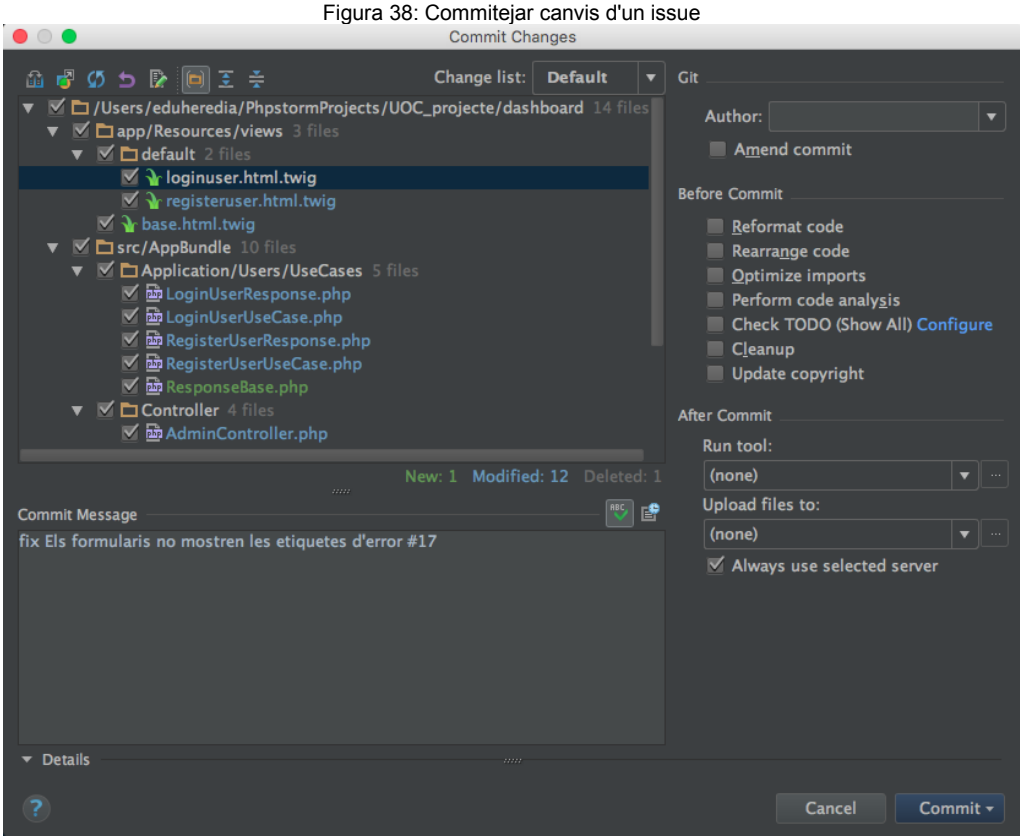


Figura 39: Issue solucionada

Per sol·lucionar els bugs de backend PHP fem servir el depurador pas a pas de xdebug. Amb l'ajuda del IDE Phpstorm ens permet veure tots els valors de les variables, la pila d'execució i seguir el flux del mateix. Un cop l'hem localitzat, ajustem el codi. Seguidament executem els tests i els ajustem amb compte si cal (*). Si el cas és nou, afegim un test per tenir cobert aquesta nova situació.

(*) Nota: Aquí cal determinar si un test falla perquè estava malament o realment ens està avisant de què hem trencat alguna altra funcionalitat. Aquí entra una mica l'experiència del programador i l'ús de tests funcionals ajuden a determinar la naturalesa del problema.

També ens aporta molta informació la barra de depuració de Symfony, en ella podem veure informació com els codis de resposta, ruta executada, peticions AJAX, temps de resposta … Un dels apartats són els logs. En la següent captura es mostra un error detectat en temps d'execució.

Figura 40: Symfony profiler per veure el log d'errors

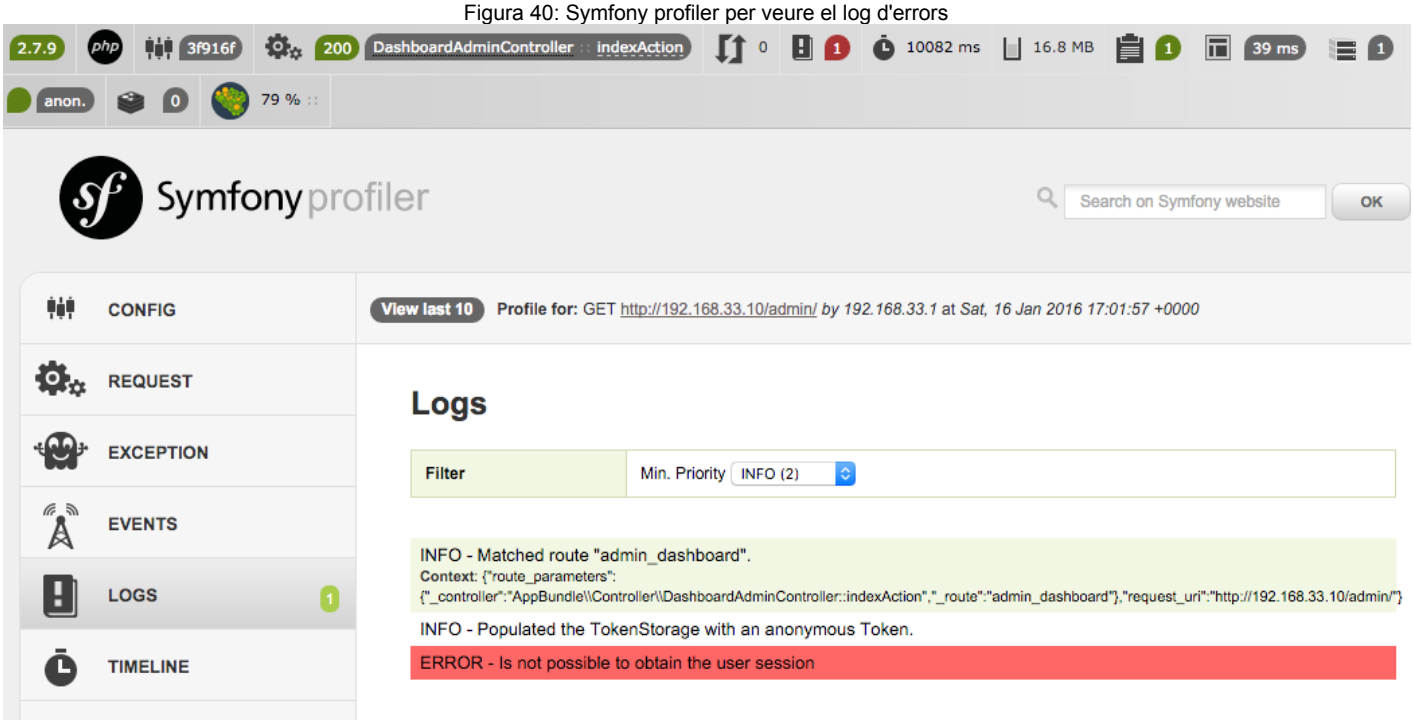
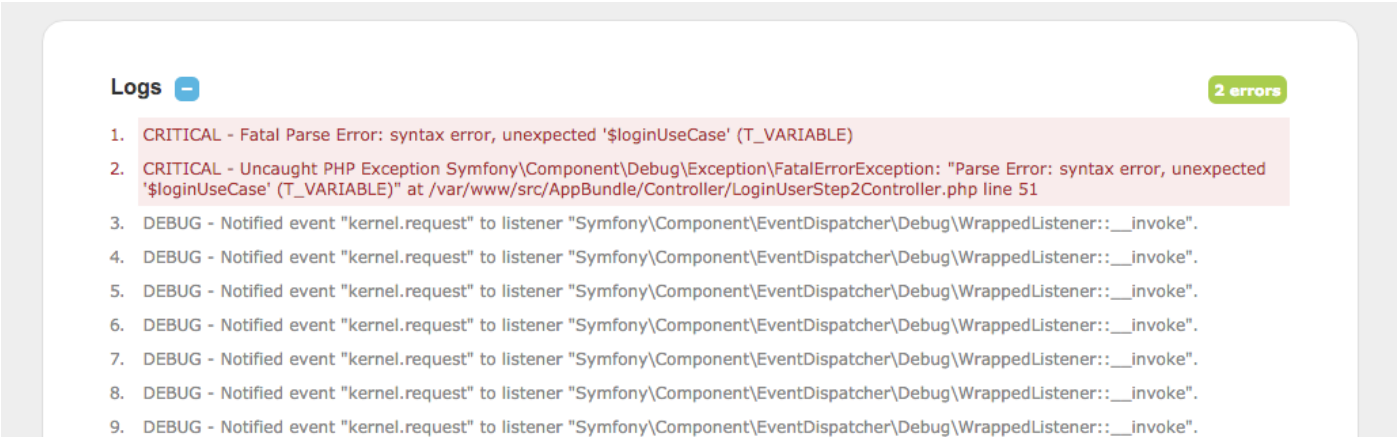


Figura 41: Traça d'un fatal



Per afrontar els bugs de NodeJs, s'ha fet seguint una metodologia més rudimentària. Bàsicament amb **console.log** en el codi. Es podria haver utilitzat **node Inspector** però vaig considerar que no feia falta perquè el codi era petit i no volia introduir més factors que poguessin desviar la planificació de l'entrega final, ja que requeria configuració de la màquina virtual. Es trigava més temps a configurar l'entorn que solucionar els pocs bugs que han sorgit.

Els bugs que hem trobat de la part de l'aplicació frontend, s'ha depurat amb les eines del navegador Chrome. Es pot fer execució pas per pas com en la part de backend. Al ser una execució pseudoasincrona, l'eina de depuració ha estat vital per reproduir errors en temps d'execució i per determinar l'estat de l'àmbit javascript en aquell moment.

Per últim els errors de frontend pur (bàsicament CSS), són solucionats mitjançant prova i error. Moltes vegades s'ha utilitzat l'editor de la consola de Chrome, per evitar de recarregar la pàgina cada cop.

21. Pressupost

En aquesta secció detallem el pressupost del projecte, es pot classificar en aquests grups:

- **Serveis externs** → En projectes reals és normal contractar recursos a terceres parts que s'han especialitzat a oferir aquests recursos. Una de l'avantatge és que et despreocupes de la gestió d'aquests recursos i et pots centrar en el nucli del teu producte.

- **Recursos propis** → Son recursos que són necessaris per realitzar el projecte que hem adquirit. Són inversions que es poden utilitzar per realitzar altres projectes i/o activitats relacionades. Com que el projecte dura 4 mesos, s'ha calculat el pressupost a partir de l'amortització en anys estimat del producte i calculant la part proporcional amb respecte la duració del projecte.

- **Formació** → S'han dedicat recursos per adquirir bibliografia recomanada que ens ajuda a realitzar el projecte i ens permet tenir una millor visió de desenvolupar el projecte amb bones pràctiques.

- **Equip humà** → Els recursos humans són la part imprescindible (i normalment més cara) dels recursos que defineixen un projecte. Per tant s'ha de considerar en el pressupost. En l'actual projecte hi ha un sol recurs que té diversos rols: cap de projectes, arquitecte, analista, developper ...

En la següent taula es detalla el pressupost per desenvolupar tot el projecte durant 4 mesos.

	Tipus	Preu	Pressupost total projecte
Github Micro	Servei extern	7 $ / mes	28 $
Digital Ocean SSD 2 GB instance	Servei extern	10 $/mes	40 $
Plantilles dashboard	Servei extern	20 $	20 $
Mac Air 13" 2014 SSD	Recursos propis	950 $	60 $ (*)
Monitor extern	Recursos propis	150 $	12 $ (*)
Teclat extern	Recursos propis	100 $	5.6 $ (**)
Ratolí Logitech	Recursos propis	30 $	2.5 $ (*)
Llibres	Formació		150 $
Recurs humà	Equip humà	50 $/hora	20.000 $
Total			20.290 $

(*) Amortitzat en 4 anys, part proporcional a 4 mesos de projecte.
(**) Amortitzat en 7 anys, part proporcional a 4 mesos de projecte.
(***) A raó de 25 hores setmanals.

Consideracions

En ser un projecte final de màster, recursos com l'equip humà es comptabilitzen però no apliquen, però hem decidit deixar una aproximació per demostrar com el cost humà representa més del 95% del cost del projecte.

Si el projecte es desenvolupés en una empresa, els costos serien molt més elevats. El motiu és que no s'han considerat recursos com poden ser els costos de l'oficina i derivats com llum, aigua, neteja, gestió, altre personal … i probablement les necessitats de recursos serien més grans. Tampoc s'ha tingut en compte els impostos.

22. Viabilitat

Després d'analitzar el projecte a partir de molts punts de vista. Arribem a la conclusió de què **és tècnicament factible**.

En l'àmbit econòmic hauríem de distingir dos escenaris, el primer seria desenvolupar el projecte com opensource, aleshores moltes de les despeses no serien aplicables (per exemple el repositori o els recursos humans). L'altre escenari seria desenvolupar i comercialitzar el projecte en un entorn professional, aleshores es requeriria un pressupost més elevat i hauríem de tornar a analitzar l'abast del mateix.

No és el mateix fer el desenvolupament i vendre llicencies, que oferir el servei des de la nostra plataforma, així com oferir la programació de nous widgets o oferir assistència tècnica.

23. Conclusions

Aquest projecte ha sigut molt interessant perquè tenia una dificultat mitjana-alta. I tenia bastants reptes que s'ha solucionat de forma satisfactòria. Des d'un inici es va considerar que desenvolupar el projecte en el temps indicat era possible, sempre que es seguís algunes estratègies. La més important va ser el no reinventar la roda, si ja existeix un mòdul disponible per la comunitat, aleshores cal utilitzar-lo (després d'analitzar dues o 3 alternatives). També ha estat important seguir la planificació i no deixar tot per l'últim moment.

El meu nivell tècnic també ha pujat una mica en poder practicar tecnologies diverses. També ha estat interessant el què, per la naturalesa del projecte, s'hagi permès aplicar les meves idees sense limitacions, cosa que és complicat en l'àmbit laboral.

Estic satisfet de com ha quedat el projecte, és gratificant obtenir un producte després de tant esforç.

Projecció a futur

El projecte del sistema de dashboards té molt potencial de creixement. La base és sòlida i està pensat per ser fàcilment extensible. Hi ha components que es podrien canviar si hi ha d'altres que es poden adaptar millor. Aquestes peces estan separades conceptualment en capes i són dependències que es poden reemplaçar.

Sóc conscient de què es pot millorar per obtenir un producte encara més útil, més fàcil d'utilitzar i amb una qualitat global encara més alta. La idea és continuar el projecte després de la presentació, utilitzar-lo en un entorn de producció real i donar-lo a la comunitat. Per arribar a aquesta última fase cal encara feina d'optimització i de creació d'un catàleg més gran de widgets disponibles. També es requereix que augmenti el nivell de flexibilitat, com per exemple la possibilitat de configurar paràmetres/templates addicionals des del dashboard. En l'àmbit visual tota l'aplicació hauria de ser responsive, per poder ser visualitzat desde dispositius mòbils fins a grans pantalles.

Espero millorar-lo aviat amb noves funcionalitats que tinc al cap. Sobretot es revisarà el tema d' optimitzar els temps de resposta per permetre una quantitat de widgets en execució més alta i amb una resposta encara més a temps real. Espero que arribi a ser un producte útil per la comunitat. Considero totalment positiu l'aportació d'aquest projecte a la meva carrera professional.

Annex 1. Lliurables del projecte

S'aporten els següents lliurables:

- Memòria del projecte amb l'estat actual
- Informe d'autoevaluació
- Codi font del projecte versió final
- Volcat de dades del repositori Mysql
- Documentació
- Scripts propis de configuració i deploy

Annex 2. Codi font extractes rellevants

L'aplicació web que s'està desenvolupant en aquest projecte, a alt nivell es pot subdividir en backend, API, backend asíncron, frontend javascript i disseny.

Backend

Per la gestió de les dependències utilitzem **Composer**. En el fitxer **composer.json** es poden anar afegint mòduls de projectes opensource i d'altres configuracions com l'autoload **PSR4**.

```
{
    "name": "eduheredia/dashboard",
    "license": "proprietary",
    "type": "project",
    "autoload": {
        "psr-4": {
            "": "src/"
        },
        "exclude-from-classmap": ["**/Tests/"]
    },
    "require": {
        "php": ">=5.3.9",
        "symfony/symfony": "2.7.*",
        "doctrine/orm": "^2.4.8",
        "doctrine/doctrine-bundle": "~1.4",
        "symfony/assetic-bundle": "~2.3",
        "symfony/swiftmailer-bundle": "~2.3",
        "symfony/monolog-bundle": "~2.4",
        "sensio/distribution-bundle": "~4.0",
        "sensio/framework-extra-bundle": "^3.0.2",
        "incenteev/composer-parameter-handler": "~2.0",
        "doctrine/collections": "~1.3",
        "ramsey/uuid": "^3.0",
        "friendsofsymfony/rest-bundle": "^1.7",
        "jms/serializer-bundle": "^1.1",
        "nesbot/carbon": "^1.21"
    },
    "require-dev": {
        "sensio/generator-bundle": "~2.3",
        "bruli/php-git-hooks": "^2.3",
        "seld/jsonlint": "^1.3",
        "nelmio/api-doc-bundle": "~2.11"
    },
```

Figura 42: Configuració de composer.json amb PSR4

Utilitzem com a base el framework **Symfony versió 2.7** que s'ocupa de tasques com ara la gestió de rutes, controladors, configuracions, sessions, formularis i peticions/respostes. També s'encarrega d'altres funcionalitats opcionals com són Service Container.

Durant el desenvolupament s'ha tingut present que els controladors han de ser el més petit possibles. Tota la lògica de negoci estarà en una altra capa aïllada i només accessible a través de casos d'ús.

Figura 43: Controlador reduït, utilitzant un cas d'us i retornant un JSON

```php
public function getInfoAdmin($idUser)
{
    $userSession = $this->get('app.controller_session.user_session');
    if (!$userSession->isLogged()) {
        $view = $this->view([], 401);

        return $this->handleView($view);
    }

    $user = $this->getUserBySession($userSession);

    if ($user->id() !== $idUser) {
        $view = $this->view([], 401);

        return $this->handleView($view);
    }

    $request = new ListDashboardsRequest();
    $request->setUser($user);
    $request->conn = $this->get('database_connection');
    $listDashboard = $this->get('app.model_dashboards_use_cases.list_dashboard_use_case');
    $response = $listDashboard->execute($request);

    $view = $this->view($response->toArray(), 200)->setFormat('json');

    return $this->handleView($view);
}
```

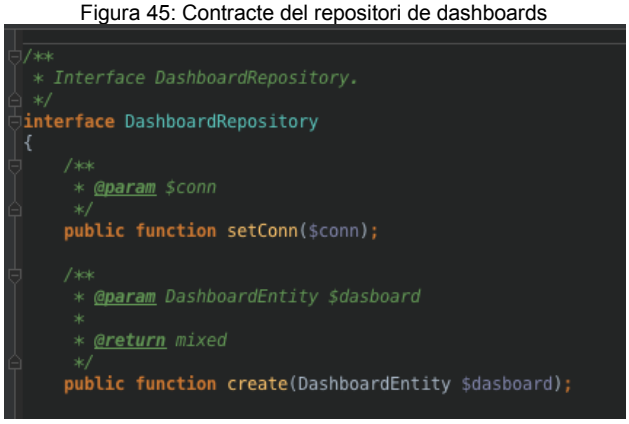
Per a cada acció que feia falta (crear dashboard, registrar usuari, instanciar un widget…) s'ha creat un cas d'ús. El motiu és que compleix el principi de **responsabilitat única** (SRP). A més com utilitza un patró **command**, donada una petició ens retorna una resposta. El nombre de possibilitats de camins d'execució (**complexitat ciclomàtica**) és bastant reduït. Junt amb la injecció de la dependència en el constructor (que facilita injectar objectes *mocks* al ser un contracte), el testeig es redueïx idealment en testejar el **happy path** i l' **error path**). En alguns casos s'havien de fer unes verificacions prèvies, aleshores s'ha utilitzat **clàusules de guàrdia**. El motiu és que la complexitat ciclomàtica només s'incrementa en 1, per tant feia falta un test extra per cada verificació.

Figura 44: Cas d'us, amb injecció de dependència d'un repositori en el constructor.

```php
/**
 * Class ListDashboardsUseCase.
 */
class ListDashboardsUseCase
{
    private $dashboardRepository;

    /**
     * CreateDashboardUseCase constructor.
     *
     * @param DashboardRepository $dashboardRepository
     */
    public function __construct(DashboardRepository $dashboardRepository)
    {
        $this->dashboardRepository = $dashboardRepository;
    }

    /**
     * @param ListDashboardsRequest $request
     *
     * @return ListDashboardsResponse
     */
    public function execute(ListDashboardsRequest $request)
    {
        try {
            $this->dashboardRepository->setConn($request->conn);

            /*
             * @var UserEntity
             */
            $user = $request->getUser();
            $listCollection = $this->dashboardRepository->fetchAll($user);

            $response = new ListDashboardsResponse();
            $response->data = $listCollection;
        } catch (\Exception $e) {
            $response = new ListDashboardsResponse();
            $response->message = 'Cannot retrieve the list of dashboard';
        }

        return $response;
    }
}
```

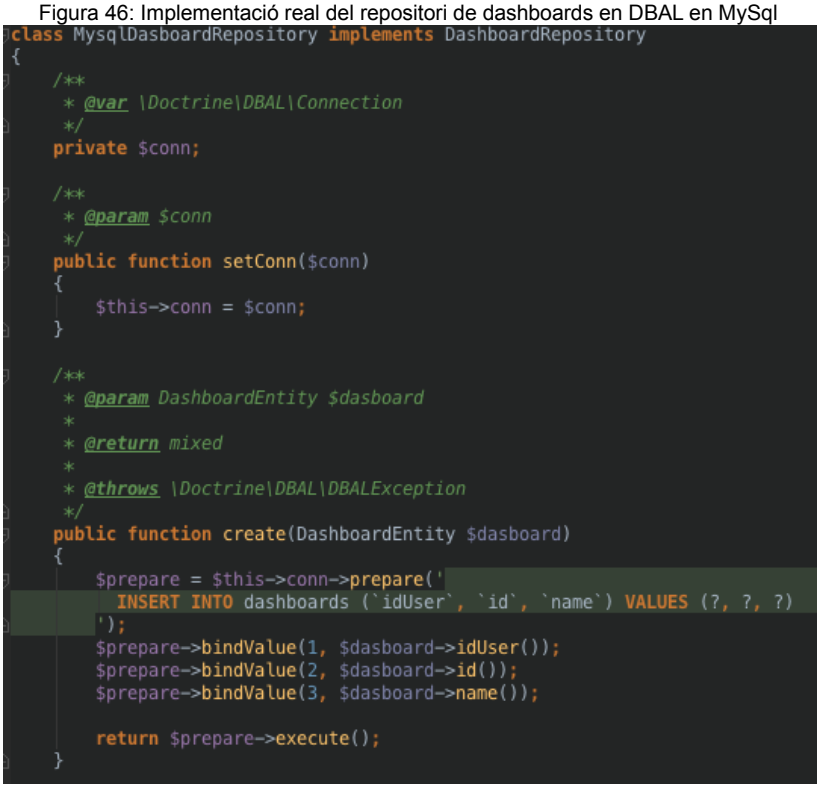
A partir del cas d'ús, ja es poden utilitzar els repositoris. La implementació real es troba a la carpeta d'insfraestructura, i el contracte es troba la carpeta del model. D'aquesta forma podem implementar el repositori en diferents infraestructures com poden ser Mysql, Redis …

Figura 45: Contracte del repositori de dashboards



```php
/**
 * Interface DashboardRepository.
 */
interface DashboardRepository
{
    /**
     * @param $conn
     */
    public function setConn($conn);

    /**
     * @param DashboardEntity $dasboard
     *
     * @return mixed
     */
    public function create(DashboardEntity $dasboard);
```

Figura 46: Implementació real del repositori de dashboards en DBAL en MySql

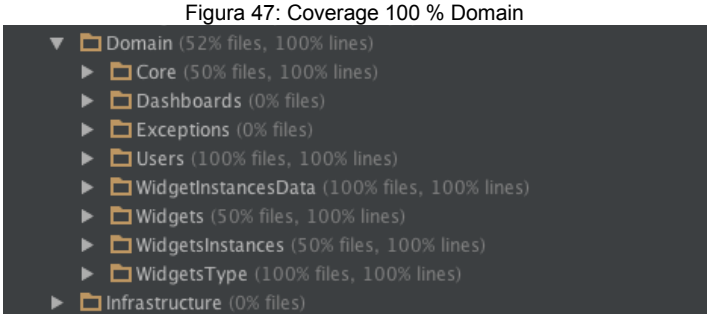


```php
class MysqlDasboardRepository implements DashboardRepository
{
    /**
     * @var \Doctrine\DBAL\Connection
     */
    private $conn;

    /**
     * @param $conn
     */
    public function setConn($conn)
    {
        $this->conn = $conn;
    }

    /**
     * @param DashboardEntity $dasboard
     *
     * @return mixed
     *
     * @throws \Doctrine\DBAL\DBALException
     */
    public function create(DashboardEntity $dasboard)
    {
        $prepare = $this->conn->prepare('
          INSERT INTO dashboards (`idUser`, `id`, `name`) VALUES (?, ?, ?)
        ');
        $prepare->bindValue(1, $dasboard->idUser());
        $prepare->bindValue(2, $dasboard->id());
        $prepare->bindValue(3, $dasboard->name());

        return $prepare->execute();
    }
```

Per últim ens falta comentar les entitats. Aquestes tenen unes propietats que la defineixen uns mètodes per actuar sobre aquestes. Per exemple un usuari té un correu electrònic i una contrasenya. I disposa d'un mètode per verificar si l'usuari és nou o no. Les entitats es poden crear en els casos d'ús i com a resposta d'un repositori en una operació de lectura. En el nostre cas s'ha utilitzat també des de el controlador (trencaria la separació de capes de DDD) per crear els formularis de symfony (en aquest cas es podria discutir si realment és una entitat o no, ja no té propietats inicialitzades i no es pot actuar amb els seus mètodes).

La capa del domini disposa d'una cobertura del 100 %, així estem segurs de no trencar lògica de negoci. Recordar que els noms (de les classes, mètodes …) han de seguir una nomenclatura anomenada *Ubiquitous Language*, que és un llenguatge que entén el programador i el client. Si algun dels termes no és de negoci, s'ha de refactoritzar i extreure d'aquesta capa de domini.

Figura 47: Coverage 100 % Domain



Arribats a aquest punt ens trobem amb un problema (sobretot quan estem fent la part frontend). Un controlador té associada una única acció per cada ruta i a més te una plantilla associada. Això impedeix la reutilització i el codi es comença a duplicar.

La solució és implementar una API.

Backend (API)

Implementar una API REST és relativament senzill en symfony (amb FOSRestBundle). És simplement crear un controlador que retorni recursos en un format adequat com pot ser JSON (es fa normalment amb *data transformers*). Per facilitar encara, es disposa d'un bundle anomenat **nelmio api doc** que documenta de forma impecable una API REST, a més disposa d'un *sandbox* per realitzar proves.

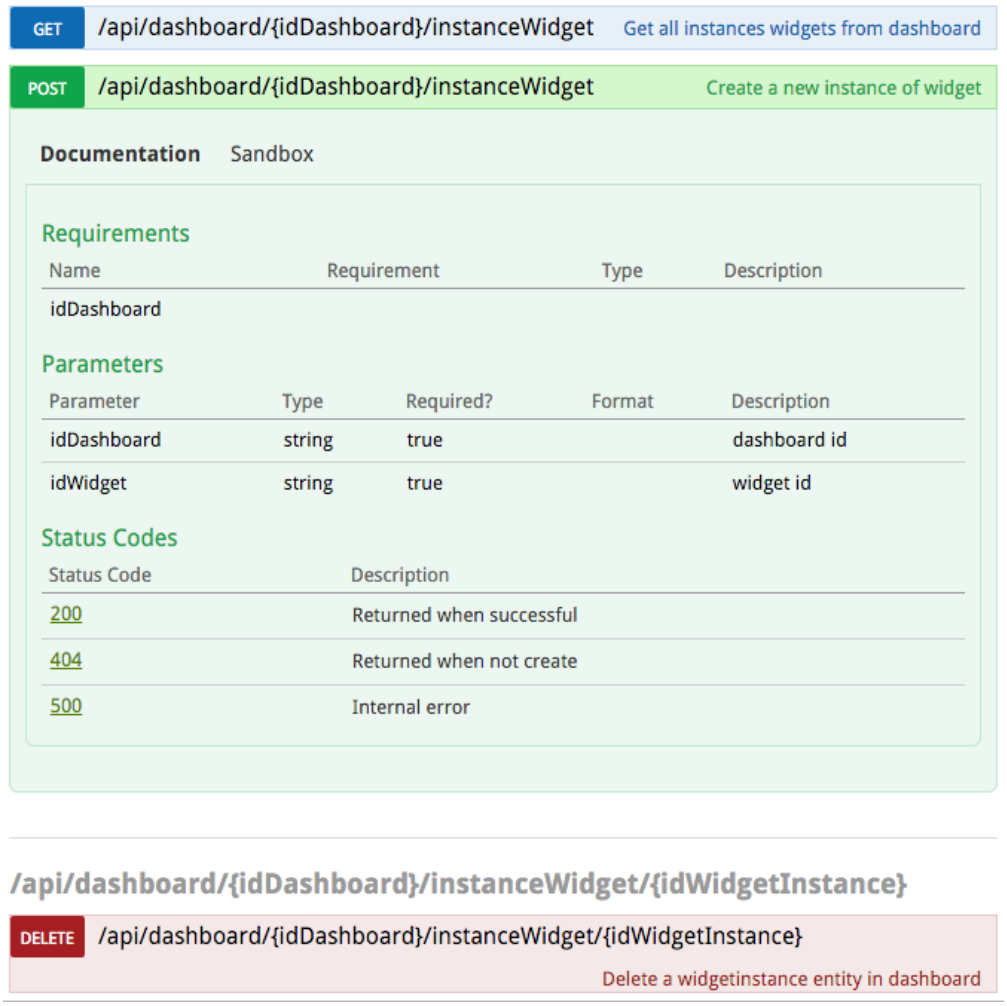
Figura 48: La documentació s'afegeix com a annotation

```
/**
 * @Route("/api/dashboard/{idDashboard}/instanceWidget", name="Post_widget_dashboard")
 * @Method({"POST"})
 * @ApiDoc(
 *  resource=true,
 *  description="Create a new instance of widget",
 *  parameters={
 *      {"name"="idDashboard", "dataType"="string", "required"=true, "description"="dashboard id"},
 *      {"name"="idWidget", "dataType"="string", "required"=true, "description"="widget id"}
 *  },
 *  statusCodes={
 *     200="Returned when successful",
 *     404={
 *          "Returned when not create"
 *     },
 *     500="Internal error"
 *   }
 * )
```

D'aquesta forma desacoplem els controladors Web que s'encarreguen de muntar la vista (entre altres tasques, dels controladors d'API que retornen dades. En un cas ideal, tots els casos d'us haurien d'anar als controladors d'API i els controladors web fer crides a la API (sigui externament per exemple amb Curl o l'abstracció de Guzzle) o internament mitjançant subRequests de controladors.

De la part backend només quedaria afegir recursos que siguin necessaris des de la part frontend. Una feina a aquestes alçades del projecte, relativament ràpid, ja que l'estructura està muntada i tot està testejat.

Figura 49: La documentació generada



Frontend (Javascript)

Un cop teníem la base backed es va procedir amb la part frontend. Per a gestionar les dependències s'ha utilitzat **NPM** i **bower** (té un format similar al del composer). Per carregar les dependències s'ha fet servir **RequireJS**, en un principi les dependències que es van utilitzar eren AMD (Asincronous module definition) però en avançar el projecte es van poder configurar per utilitzar llibreries més actualitzades que no estaven configurades com AMD. En un principi no ha donat cap problema.

Amb aquesta base frontend, s'ha utilitzat **jQuery** per la interacció amb el DOM i creació de diàlegs, **jQuery UI** per interacció en la interfície (per arrossegar widgets, obtenir posicions dels elements …).

Per dotar de funcionalitat d'aplicació, s'ha utilitzat **Backbone.js** i el sistema de plantilles **underscore.**

En **Backbone** existeixen models, vistes i col·leccions. Els models es poden omplir amb les dades de la API de forma relativament fàcil, el motiu és que en la part backend disposem d'entitats i la relació és 1 a 1 amb el model de backbone.

Figura 50: Entitat Dashboard (backend)

```php
/**
 * Class DashBoard.
 */
class DashBoard implements DashboardEntity
{
    /**
     * @var string
     */
    protected $id;

    /**
     * @Assert\NotBlank()
     *
     * @var string
     */
    protected $name;
```

Figura 51: Col·lecció de recursos Dashboards convertida a JSON a través de la API

```
GET /api/user/a160d95b-0de0-4e50-9f61-2cdd494c2623/dashboards?_format=json
```

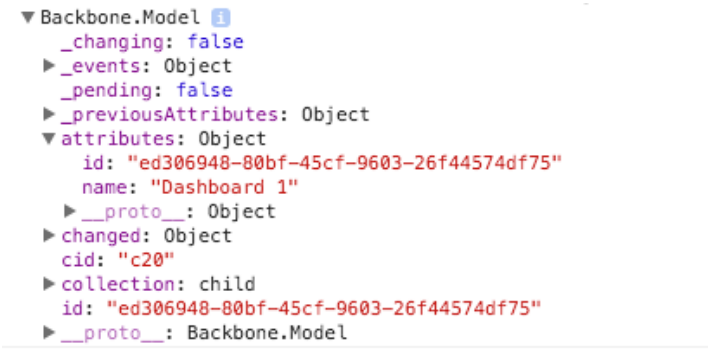
Response Headers [Expand] [Profiler]

```
200 OK
```

Response Body [Raw]


```
▿[
  ▿{
    "id": "11b9a0d2-4d6c-4f3a-b3fd-c5ba4973e30f",
    "name": "Dashboard 1"
  },
  ▿{
    "id": "66fea2ef-f3c7-4e46-8ad5-9bffec2ca00e",
    "name": "Dashboard 1"
  },
  ▿{
    "id": "d5d4d431-f2db-4bd6-8d40-ca755aa00ab0",
    "name": "Dashboard 1"
  },
  ▿{
    "id": "ed306948-80bf-45cf-9603-26f44574df75",
    "name": "Dashboard 1"
  }
]
```


Figura 52: Model backbone d'una col·lecció de Dashboards, omplert amb les dades de la API, els atributs són equivalents a les propietats de l'entitat original



Respecte a les vistes, s'ha optat per 2 estratègies. La primera és tenir un element DIV contenidor juntament amb plantilles **underscore** per col·leccions i models invariables (per exemple llistats) en la pròpia plantilla de cada pàgina. La segona és per vistes més dinàmiques, com els widgets que canvien de disseny depenent del tipus, la *template* s'adjunta a la petició AJAX que realitza el model B**ackbone** en sincronitzar amb el servidor.

Figura 53: Model backbone d'una instancia de widget, amb la template incorporada per la API

En aquesta part es realitza tota la interacció amb l'usuari. A través de l'API es poden obtenir, crear o modificar recursos.

L'aplicació frontend es comunica amb el backend asíncron, mitjançant sockets. Permet realitzar accions d'actualitzacions de dades. Per facilitar-ho, s'ha implementat un patró mediator per facilitar la coordinació de l'aplicació.

Backend (Asíncron)

Per poder obtenir un comportament de l'aplicació en temps real, feia falta un element capaç de coordinar el frontend i el backend de forma asíncrona i basada en esdeveniments. S'ha escollit **NodeJS** amb el framework **Express**, que s'encarrega de comunicar-se amb el frontend per sockets, i amb el backend mitjançant events o la mateixa API.

Aquest servidor, està subscrit a un bus d'events. En el nostre cas està muntat amb Redis amb PUB/SUB.

El funcionament de l'actualització és el següent:
1- El frontend demana les dades del widget a través de l'API.
2- El backend li retorna si n'hi ha. Sinó retorna un 404. La instància de widget s'actualitza.
3- Es genera un event de «**generació de dades**» desde el backend.
4- El backend asíncron, rep aquesta notificació a través de Redis.
5- Fa una petició a l'API per generar dades noves.
6- Quan el backend ha generat noves dades, genera un altre event de «**dades generades**»
7- El backend asíncron, notifica a l'aplicació que hi ha dades per una determinada instància de widget
8- L'aplicació backend refresca el model. Tornant al pas 1.

També s'ha afegit un *fallback*, per a que es faci una petició de generació de dades en cas de què no sigui demanada pel frontend mentre hi hagi un client connectat.

Frontend (Disseny)

La integració del framework **Bootstrap** no ha sigut gaire complex, ja que s'integra de forma coherent amb les plantilles de la pàgina i amb els elements de la mateixa com poden ser formularis. Aquests components s'integren fàcilment. Ha donat com a resultat una web responsive, que és usable i amb un disseny més que acceptable.

Donat el conjunt d'elements que ofereix Bootstrap, s'han anat seleccionant els més adients per dotar d'usabilitat a la web.

S'ha utilitzat components per definir comportaments diferents donat una única font de dades. Així visualment es veuen diferent, però internament són el mateix element. Un exemple és el següent:

Figura 54: Mateix component, diferents visualitzacions

Annex 3. Llibreries/Codi extern utilitzat

Totes les dependències es troben en el fitxers :

- composer.json
- bower.json
- package.json

S'han utilitzat llibreries open source disponibles per no reinventar la roda. En entorns professionals el fet d'utilitzar llibreries externes et pot donar avantatge competitiu perquè els components solen tenir una comunitat darrera que els milloren i solen funcionar correctament. Com a perill és què tens dependències externes que no controles. Normalment aquestes llibreries estan ben mantingudes (amb *semantic versioning*) i disponibles.

També és possible que no s'adapti al 100 % amb el que busques, però sempre pots fer un *fork* del codi i adaptar-lo a les teves necessitats.

No s'han guardat les dependències en el repositori.

Annex 4. Glossari

API REST → Respresentational state transfer. API que es comunica per vers HTTP com GET, POST, PATCH ...
https://en.wikipedia.org/wiki/Representational_state_transfer

Bounded Context → Separació de contextes
http://martinfowler.com/bliki/BoundedContext.html

Complexitat ciclomàtica → Es una mètricaque ens determina quans camins d'execució possibles existeixen en un àmbit concret d'un codi font.

DDD → Domain Driven Design

Feature branch → Branca per desenvolupar petites noves funcionalitats

KPI → Indicador clau de rendiment. És una mètrica de negoci.
https://es.wikipedia.org/wiki/KPI

Ubiquitous Language → Llenguatge comú que interpreta de la mateixa manera una persona de negoci y una técnica.
http://martinfowler.com/bliki/UbiquitousLanguage.html

SOLID → Conjunt de principis de desenvolupament orientat a objectes.

- S → Principi de responsabilitat única
- O → Principi obert / tancat
- L → Substitució de Liskov
- I → Segregació d'interficies
- D → Inversió de dependències

https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)

Annex 4. Bibliografia

- **Patterns, principles and Practices of Domain driven Design.** Scott Millet with Nick Tune
- **Pro Git**. Scott Chacon
- **Javascript Ninja** John Resig. Bear Bibeault
- **Real world solutions for developing high quality PHP Frameworks and applications**. Sebastian Bergmann, Stefan Priebsch
- **Refactoring. Improving the design of existing code** Martin Fowler
- **Clean code.** Rober C Martin
- **Restful Web Services Cookbook**. Subbu Allamaraju

Recursos web:

http://www.wikipedia.com
http://www.martinfowler.com
https://blog.8thlight.com
https://www.digitalocean.com/community/tutorials/
http://www.phptherightway.com/