



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática



TFG del Grado en Ingeniería Informática
DOCUMENTACIÓN TÉCNICA



Presentado por Adrian Aguado
en Universidad de Burgos — 29 de junio de 2017
Tutor: Luis R.Izquierdo

Índice general

Índice general	I
Índice de figuras	III
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	2
A.3. Estudio de viabilidad	8
Apéndice B Especificación de Requisitos	13
B.1. Introducción	13
B.2. Objetivos generales	14
B.3. Catálogo de requisitos	14
B.4. Especificación de requisitos	15
Apéndice C Especificación de diseño	30
C.1. Introducción	30
C.2. Diseño de datos	30
C.3. Diseño arquitectónico	41
Apéndice D Documentación técnica de programación	43
D.1. Introducción	43
D.2. Requerimientos mínimos necesarios	44
D.3. AngularCLI	44
D.4. Manual del programador	46
Apéndice E Documentación de usuario	55
E.1. Introducción	55
E.2. Usuario Web	55
E.3. Usuario Móvil	56

ÍNDICE GENERAL

II

Bibliografía

57

Índice de figuras

A.1. Detalle sprint 0.	2
A.2. Detalle sprint 1.	3
A.3. Detalle sprint 2.	4
A.4. Detalle sprint 3.	4
A.5. Detalle sprint 4.	5
A.6. Detalle sprint 5.	5
A.7. Detalle sprint 6.	6
A.8. Detalle sprint 7.	6
A.9. Detalle sprint 8.	7
A.10. Detalle sprint 9.	8
A.11. Gráfico licencias Open Source in GitHub. Fuente: https://cartograf.net . . .	11
A.12. Licencia Creative Commons.	12
B.1. Casos de uso. Fuente: Elaboración propia.	16
C.1. Componente Angular . Fuente: http://blog.enriqueoriol.com/	30
C.2. Relación plantilla-componente. Fuente: http://academia-binaria.com/	31
C.3. Diagrama relacional. Fuente: Elaboración propia.	33
C.4. Esquema Angular por dentro. Fuente: www.angular.io	34
C.5. Envío-respuesta http . Fuente: http://expressjs.com/	36
C.6. Funcionamiento <i>API</i> . Fuente: https://juanda.gitbooks.io/	39
C.7. Vista de la conexión entre el servidor y el cliente. Fuente: Elaboración propia.	40
C.8. Esquema Arquitectura. Fuente: http://www.cantabriatic.com/	41
D.1. Comandos <i>AngularCLI</i> . Fuente: https://cli.angular.io/	45
D.2. Consola ejecución. Fuente: http://codigoxules.org/	45
D.3. Resultado después de creado. Fuente: http://codigoxules.org/	46
D.4. Vista carpetas general <i>BarterAPP</i> . Fuente: Elaboración propia.	48
D.5. Vista Robomongo. Fuente: Elaboración propia.	52
D.6. Documentos configuración [10]. Fuente: http://nanobox.io/	54

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En este capítulo se detalla la planificación del proyecto. Como gestor de tareas se comenzó utilizando *Trello+Github* pero más tarde se pasó a utilizar *Zenhub*, extensión de Google Chrome que permite integrar los *boards* dentro del mismo repositorio de código alojado en *github*.

Se han utilizado metodologías ágiles para el desarrollo del proyecto y de este modo, se ha realizado un desarrollo dividido en iteraciones. Terminada una iteración empezaba la siguiente y se agregaban a las tareas planeadas las que no habían sido completado de la iteración precedente. Las iteraciones del proyecto estaban pensadas para durar unos diez días aproximadamente. No obstante, hay alguna excepción en la que la iteración duró más tiempo. También existe alguna demora entre algún sprint debido a que tenía demasiada carga de trabajo de las asignaturas, trabajaba o estaba de viaje.

Este primer apéndice se puede dividir a su vez en:

- Planificación temporal.
- Estudio de viabilidad.

La primera parte me centro en la programación y desarrollo de la aplicación. Es decir elaboro un programa de tiempos con una serie de tareas a seguir para cumplimentar el proyecto.

La segunda parte se centra en el estudio de viabilidad. Es necesario destacar en este punto que desde la segunda semana de marzo vengo realizando un plan de empresa con el *programa Yuzz* por lo que ello me va a facilitar el estudio de viabilidad de mi proyecto. Se desarrollará tanto la viabilidad legal como también la económica.

A.2. Planificación temporal

Desde el comienzo del proyecto se planteó utilizar una metodología ágil como *Scrum* para la gestión del proyecto. Aunque no se ha seguido al 100% la metodología al tratarse de un proyecto para la universidad, sí que se ha aplicado en líneas generales una filosofía ágil y metódica. La diferencia fundamental radica en que esta metodología esta pensada para equipos y no para individuos.

A continuación se describen los diferentes *sprints* que se han realizado. Dentro de *github* cada *milestone* recibe el número del sprint asignado y dentro de cada uno de ellos existen diferentes tareas que describiré a continuación. A cada tarea le acompaña un número a la derecha el cuál es denominado *story point*, de alguna manera sirve para realizar una estimación de lo que te va llevar completar esa determinada tarea. En mi caso concreto el **1** resulta ser el más bajo lo cuál indicaría que no más de dos o tres horas con cada tarea y el **13** el más alto lo cuál significa varios días de trabajo.

Sprint 0: 18/02/2017 - 28/02/2017

Tareas principales:

- Terminar formación.
- Aprender *Sonarqube*.
- Inicio Back-End.
- Inicio Decidir base de datos a emplear.

Completed Issues and Pull Requests	Story points
Terminar formación courses barterAPP #1 III New Issues ↑ Sprint 0	8
Aprender uso https://www.sonarqube.org/ research various barterAPP #2 III New Issues ↑ Sprint 0	1
Decidir base de datos a usar. research various barterAPP #3 III New Issues ↑ Sprint 0	1
Comenzar desarrollo Back-End. web app (Back-End) barterAPP #5 III New Issues ↑ Sprint 0	8
Información sobre bases de datos NoSQL courses research barterAPP #6 III New Issues ↑ Sprint 0	9

Figura A.1: Detalle sprint 0.

La primera vez que hice uso de *ZenHub* ya llevaba algún tiempo formándome, de ahí el primer *Issue* del Sprint 0. Esta primera toma de contacto fue para comenzar a desarrollar el Back-End de la aplicación, además se consultaron varias fuentes para decidir el tipo de base de datos emplear.

Sprint 1: 18/02/2017 - 28/02/2017

Tareas principales:

- Corrección de errores en Back-End.
- Desarrollo Back-end.
- Errores en base de datos.







Completed Issues and Pull Requests	Story points
 Actualizar Mendeley con el nuevo contenido consultado documentation research barterAPP #8 III New Issues ↑ Sprint 1	3
 Corregir errores en base de datos bugs web app (Back-End) barterAPP #9 III New Issues ↑ Sprint 1	5
 Añadir identificación usuarios web app (Back-End) barterAPP #10 III New Issues ↑ Sprint 1	8
 Actualizar package.json web app (Back-End) barterAPP #11 III New Issues ↑ Sprint 1	2
 Corregir errores entre versiones bugs web app (Back-End) barterAPP #12 III New Issues ↑ Sprint 1	5
 Errores en package.json bugs web app (Back-End) barterAPP #16 III New Issues ↑ Sprint 1	8

Figura A.2: Detalle sprint 1.

El Sprint 1 sirvió para continuar con el Back-end de la aplicación, sin dudarlo fue un de las partes más complicadas al pelearme con bases de datos con conceptos nuevos por lo que tuve numerosos bugs a la hora de guardar los usuarios en la base de datos.

Sprint 2: 15/03/2017 - 31/03/2017

Tareas principales:

- Corrección de errores en Back-End.
- Inicio Front-End.
- Bug en base de datos.
- Comienzo a leer sobre la documentación.

Completed Issues and Pull Requests	Story points
<div> Landing-page web app (Front-End) barterAPP #13 III New Issues ↑ Sprint 2 </div>	5
<div> Estructura visual web app (Front-End) barterAPP #15 III New Issues ↑ Sprint 2 </div>	3
<div> MVC enhancement web app (Back-End) web app (Front-End) barterAPP #17 III New Issues ↑ Sprint 2 </div>	13
<div> Bug sobre usuarios bugs web app (Back-End) barterAPP #18 III New Issues ↑ Sprint 2 </div>	5
<div> Crear componentes en angular web app (Front-End) barterAPP #19 III New Issues ↑ Sprint 2 </div>	8
<div> ReadTheDocs documentation enhancement barterAPP #20 III New Issues ↑ Sprint 2 </div>	5

Figura A.3: Detalle sprint 2.

El Sprint 2 fue el momento donde una vez tenía un back-end sólido debía trasladarlo a la parte del usuario por lo que comencé a realizar la parte del front-end.

Sprint 3: 07/04/2017 - 15/04/2017

Tareas principales:

- Subir documentación.
- Elección del calendario.
- Corrección en componentes.

Completed Issues and Pull Requests	Story points
<div> Publicar landing page en github pages bugs enhancement barterAPP #22 III New Issues ↑ Sprint 3 </div>	2
<div> Subir /docs documentation barterAPP #23 III New Issues ↑ Sprint 3 </div>	1
<div> Login de usuarios web app (Back-End) web app (Front-End) barterAPP #24 III New Issues ↑ Sprint 3 </div>	8
<div> Crear nuevos usuarios en la app. web app (Back-End) web app (Front-End) barterAPP #25 III New Issues ↑ Sprint 3 </div>	5
<div> Nuevo Schema para calendarios. web app (Front-End) barterAPP #26 III New Issues ↑ Sprint 3 </div>	3
<div> Estudiar que calendario incluir dentro de la app enhancement research web app (Front-End) barterAPP #27 III New Issues ↑ Sprint 3 </div>	1
<div> Corregir el fallo del servicio de angular2 bugs web app (Back-End) barterAPP #28 III New Issues ↑ Sprint 3 </div>	3

Figura A.4: Detalle sprint 3.

El Sprint 3 se centra en la parte del calendario sobre todo, además de algo de documentación y corregir los errores que he arrastrado del back-end.

Sprint 4: 16/04/2017 - 22/04/2017

Tareas principales:

- Actualizar a Angular CLI.
- Heroku y MLab
- Documentación

Completed Issues and Pull Requests				Story points
Actualizar a angular CLI	enhancement	web app (Back-End)	web app (Front-End)	5
barterAPP #29	III New Issues	↑ Sprint 4		
Crea base de datos online	enhancement	web app (Back-End)		1
barterAPP #30	III New Issues	↑ Sprint 4		
Desplegar web en Heroku.	enhancement			2
barterAPP #31	III New Issues	↑ Sprint 4		
Mejorar documentación.	documentation	enhancement		2
barterAPP #32	III New Issues	↑ Sprint 4		

Figura A.5: Detalle sprint 4.

El Sprint 4 es más corto dado que requiere un menor tiempo en realizar las tareas.

Sprint 5: 30/04/2017 - 07/05/2017

Tareas principales:

- Angular CLI.
- Bugs: base de datos
- Bugs: calendario

Completed Issues and Pull Requests				Story points
Continuar con la documentación	documentation			3
barterAPP #33	III New Issues	↑ Sprint 5		
Angular CLI vs Calendar	bugs	research	web app (Front-End)	6
barterAPP #34	III New Issues	↑ Sprint 5		
Error al guardar los elementos en la BD	bugs	web app (Front-End)		6
barterAPP #35	III New Issues	↑ Sprint 5		
Angular CLI	enhancement	web app (Back-End)	web app (Front-End)	3
barterAPP #36	III New Issues	↑ Sprint 5		

Figura A.6: Detalle sprint 5.

El Sprint 5 fue complicado debido a que cambiar a Angular CLI resulta más sencillo a la hora de desplegar en servidor pero hay que saber cómo funciona realmente los proyectos en Angular CLI.

Sprint 6: 09/05/2017 - 16/05/2017

Tareas principales:

- Error en turnos diarios, se decide implementar turnos 24 horas para comprobar si el algoritmo intercambia los turnos.
- Anexos en la documentación.

Completed Issues and Pull Requests	Story points
Error en turnos largos bug web app (Back-End) web app (Front-End) barterAPP #37 III New Issues ↑ Sprint 6	6
Anexos de la documentación. documentation barterAPP #38 III New Issues ↑ Sprint 6	2
Mejorar aspecto visual. enhancement web app (Front-End) barterAPP #39 III New Issues ↑ Sprint 6	13

Figura A.7: Detalle sprint 6.

El Sprint 6 se centra en que el algoritmo intercambie los turnos de manera eficiente.

Sprint 7: 24/05/2017 - 31/05/2017

Tareas principales:

- Tras consultar a Luis he obviado un importante elemento y es necesario refactorizar el algoritmo.
- Se incluye una parte para tests pero no me da tiempo.

Completed Issues and Pull Requests	Story points
Continuar documentación. documentation barterAPP #40 III New Issues ↑ Sprint 7	5
Cambiar lógica de la app. web app (Back-End) web app (Front-End) barterAPP #41 III New Issues ↑ Sprint 7	13
Aplicar test para ver la calidad del código. tests barterAPP #42 III New Issues ↑ Sprint 7	6
Leer libro "Clean Code" research various barterAPP #43 III New Issues ↑ Sprint 7	Not estimated

Figura A.8: Detalle sprint 7.

El Sprint 7 se centra en en conseguir que el algoritmo funcione para todos los tipos de turnos.

Sprint 8: 01/06/2017 - 10/06/2017

Tareas principales:

- Completar últimos componentes
- Terminar documentación.
- Bugs: Travis.
- Bugs: Heroku y fallo en el algoritmo.
- Pruebas de calidad del código (Como *SonarQube* no da soporte a Typescript decidí prescindir de esta herramienta y utilizar otras.)

Completed Issues and Pull Requests	Story points
Subir los últimos componentes. web app (Front-End) barterAPP #44 III New Issues ↑ Sprint 8.	3
Terminar documentación. documentation barterAPP #45 III New Issues ↑ Sprint 8.	8
Revisar bugs. bugs enhancement web app (Back-End) web app (Front-End) barterAPP #46 III New Issues ↑ Sprint 8.	8
Revisar Travis. bugs enhancement barterAPP #47 III New Issues ↑ Sprint 8.	5
Realizar pruebas de la calidad de código. documentation enhancement barterAPP #48 III New Issues ↑ Sprint 8.	5

Figura A.9: Detalle sprint 8.

El Sprint 8 se centra en corregir errores, uno de los cuales no consigo solventar, y realizar pruebas en la calidad del código.

Sprint 9: 10/06/2017 - 20/06/2017

Tareas principales:

- Refactorización.
- App móvil: scaffolding básico y creación de componentes.
- Envío a Luis de la documentación para posibles comentarios.
- Error en el servidor.

Completed Issues and Pull Requests	Story points
<div> Refactorización app web. web app (Back-End) web app (Front-End) barterAPP #49 III New Issues ↑ Sprint 9 </div>	(13)
<div> App móvil. mobile app barterAPP #50 III In Progress ↑ Sprint 9 </div>	(13)
<div> Terminar documentación. documentation barterAPP #51 III New Issues ↑ Sprint 9 </div>	(8)
<div> Crear ayuda en landing page. enhancement web app (Front-End) barterAPP #52 III In Progress ↑ Sprint 9 </div>	(5)
<div> Bug Heroku bugs barterAPP #53 III In Progress ↑ Sprint 9 </div>	(5)

Figura A.10: Detalle sprint 9.

El Sprint 9 trata de refactorizar la app para que la calidad del código aumente. Se consigue en un tanto por ciento considerable.

Sprint 10: 20/06/2017 - 30/06/2017

Tareas principales:

■

El Sprint 10

Sprint 11: 30/06/2017 - 10/07/2017

Tareas principales:

■

El Sprint 11

A.3. Estudio de viabilidad

En esta sección se lleva a cabo un estudio para comprobar la viabilidad del proyecto realizado. Paralelamente al desarrollo de la aplicación, como ya se ha nombrado anteriormente, el proyecto formó parte del *programa YUZZ* para jóvenes emprendedores en el que durante cinco meses realicé un plan de empresa completo. Se detalla por tanto en un documento extra que adjuntaré un estudio de viabilidad exhaustivo y muy completo en el que se incluyen entre otras cosas: plan de marketing, plan de financiación, estudio de viabilidad o plan de puesta en marcha del negocio a cinco años vista. Así como también encuestas, logotipado o un *landing page* necesaria para la presentación del proyecto ante el jurado.

Por lo tanto en esta sección voy a realizar un resumen del documento descrito en el párrafo anterior en el que como conclusión definitiva tendremos un boceto de lo que supondría transformar un trabajo fin de grado en un producto real. Así mismo voy a intentar adaptarlo a las condiciones que se exigen en el proyecto dado que el plan de empresa adjunto como extra es un estudio de viabilidad íntegro de aquí a cinco años por lo que resulta ser más extenso y detallado. Se intentará por tanto aquí realizar una estimación.

Viabilidad económica

La viabilidad económica es la parte donde lograremos detectar si el proyecto es o no rentable económicamente hablando.

Análisis de costes

Lo subdividiremos en diferentes tipos de costes.

Coste de personal Se considerará que el proyecto ha sido desarrollado en un periodo de cinco meses. Considerando que se ha trabajado unas 6 horas al día cada semana, y que el programador, que en este caso es una sola persona, ha percibido un salario de 13 €/hora, el coste del personal por lo tanto se resume en la siguiente tabla:

	Total
13 €/hora * 6 horas/día	78 €/día
78 €/día * 5 días/semana	390 €/semana
390 €/semana * 4 semanas/mes	1560 €/mes
Coste total salario	7800 €/5 meses

Tabla A.1: Tabla salarios.

Coste de seguridad social al coste de personal hay que sumar el coste de seguridad social.

<i>Seguridad social</i>	
	23,60 % en contingencias comunes
	7,70 % en concepto de desempleo
	0,10 % en concepto de formación personal.
Gastos Seguridad Social	7800 € * 31,40 % = 2449,20 €
Gastos personal	7800 € + 2449,20 € = 10249,2 €

Tabla A.2: Tabla seguridad social.

Coste de software Todo el software utilizado para este proyecto es gratuito y de libre distribución.

Coste de Hardware En este apartado se analizan los costes derivados de la compra de los materiales que son necesarios para llevar a cabo el proyecto. Según la ley de Impuesto de Sociedades, la duración media del inmovilizado hardware oscila entre 8 y 3 años. Yo consideraré 3 años. Más detalles en la siguiente tabla.

<i>Hardware utilizado</i>	
Portátil MacBook Pro 15"	1650 €
Monitor externo Samsung 22"	107'99 €
Gastos Hardware	1650 €+ 107,99 = 1757,99 €
Coste amortización	1757,99 €/ 36 meses = 48,8 €/ mes

Tabla A.3: Tabla costes hardware.

Coste final hardware (5 meses)	48,8 €/ mes * 5 meses = 244 €
---------------------------------------	-------------------------------

Tabla A.4: Coste final hardware.

Coste final desarrollo del proyecto

El coste final del desarrollo del proyecto, con una duración estimada de cinco meses es el siguiente.

Coste total final del desarrollo proyecto	244 €+ 10249,2 € = 10493 €
--	----------------------------

Tabla A.5: Coste proyecto final..

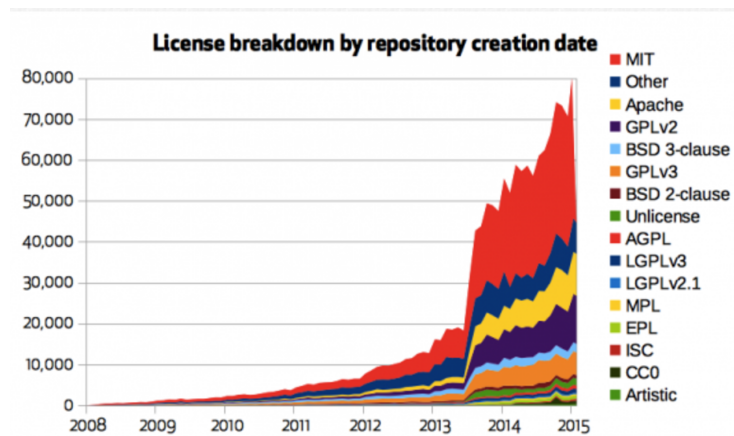
Viabilidad legal

La viabilidad legal se centra principalmente en el estudio de las licencias software utilizadas. Realizaremos una tabla resumen sobre las partes más importantes de la aplicación y la licencia que emplean. Realmente son muchos los módulos que se han empleado para el desarrollo pero la mayoría tienen la misma licencia por lo que no he nombrado todos simplemente los que más importantes parecen.

Dependencias	Licencia
<i>angular-calendar</i>	MIT
<i>@angular/cli</i>	MIT
<i>@types/node</i>	MIT
<i>primeng</i>	MIT
<i>tslint</i>	Apache-2.0
<i>typescript</i>	Apache-2.0
<i>font-awesome</i>	OFL-1.1
<i>mongoose</i>	MIT
<i>express</i>	MIT
<i>bootstrap</i>	MIT

Tabla A.6: Tabla resumen-licencias.

Una licencia software es un contrato entre el autor o titular de los derechos de explotación o distribución y el usuario consumidor, usuario profesional o empresa, para la utilización del software cumpliendo una serie de términos y condiciones establecidas dentro de sus cláusulas. Todo el software que empleo es su amplia mayoría es libre ya que la mayoría de código viene o bien de desarrolladores *amateur* o es libre desde su creación. Emplean por tanto las licencias descritas en la tabla anterior por lo que diferencia entre ellas varían en términos como el nombramiento del autor o la garantía, son licencia comunes en software libre [3], además github provee una página para ayudarte en la elección de tu licencia [9]. La menos permisiva de entre las descritas es *MIT* [13] y la más es la licencia del *Apache-2.0* [2].

Figura A.11: Gráfico licencias Open Source in GitHub. Fuente: <https://cartograf.net>.

Tal y como he nombrado anteriormente, y el tutor así lo deseaba, el software puede ser modificado, ayudando así a que crezca en un futuro por lo que he elegido *Apache-2.0*.

Para la documentación he elegido una licencia Creative commons, en concreto se ha elegido la *Attribution-NonCommercial 4.0 International (CC BY-NC 4.0)* visible en A.3 .

Reconocimiento-NoComercial 4.0 Internacional



Figura A.12: Licencia Creative Commons.

Especificación de Requisitos

B.1. Introducción

Este anexo recoge la especificación de requisitos que define el comportamiento del sistema desarrollado. El objetivo principal de la Especificación de Requisitos del Sistema (*ERS*) es servir como medio de comunicación entre clientes, usuarios, ingenieros de requisitos y desarrolladores.

La ERS es correcta si y sólo si todo requisito que figura en ella refleja alguna necesidad real. La corrección de la ERS implica que el sistema implementado será el sistema deseado. Se han seguido las recomendaciones del estándar IEEE 830 según la última versión del estándar citado. Las características deseables para una especificación de requisitos son:

1. No ambigua.
2. Completa
3. Verificable
4. Consistente
5. Clasificada
6. Modificable
7. Explorable
8. Utilizable durante las tareas de mantenimiento y uso

B.2. Objetivos generales

Los objetivos generales que se perseguían con el proyecto han sido:

- Desarrollar una aplicación web que permita intercambiar turnos.
- Desarrollar una aplicación híbrida que adapte la anterior aplicación web a cualquier dispositivo.

B.3. Catálogo de requisitos

A continuación, se enumeran los requisitos específicos:

En esta sección se especificarán los requisitos del sistema, diferenciando los requisitos funcionales, o sea el comportamiento del sistema, de los requisitos no funcionales, que describen características de funcionamiento.

Requisitos funcionales

RF-1 Gestión de usuarios: Los usuarios son una parte fundamental.

- **RF-1.1:** Registrar usuarios
- **RF-1.2:** Logear usuarios
- **RF-1.3:** Deslogearse

RF-2. Gestión de calendario: Funciona como un calendario normal.

- **RF-2.1:** Añadir turnos.
- **RF-2.2:** Eliminar turnos.
- **RF-2.3:** Editar turnos.
- **RF-2.4:** Visualizar turnos.

RF-3. Gestión de cambios: Parte de intercambio de turnos.

- **RF-3.1:** Enviar petición.
- **RF-3.2:** Aceptar petición.
- **RF-3.3:** Rechazar petición.
- **RF-3.4:** Aceptar intercambio.
- **RF-3.5:** Rechazar intercambio.
- **RF-3.6:** Visualizar detalles.
 - **RF-3.6.1:** Visualizar turnos.
 - **RF-3.6.2:** Visualizar pendientes.
 - **RF-3.6.3:** Visualizar aceptados.

- **RF-3.6.4:** Visualizar rechazados.
- **RF-3.7:** Visualizar calendario.

RF-4. Información: Información de como usar la aplicación

Requisitos no funcionales

RNF-1 Seguridad se persigue preservar la seguridad impidiendo acceder a la aplicación sin logearse. O también añadiendo un *hash* para el almacenamiento de las contraseñas de los usuarios, preservando así la seguridad de los mismos.

RNF.2 Escalabilidad: la aplicación debe estar preparada para actuar con un gran número de usuarios.

RNF.3 Soporte: la aplicación debe ser soportable por diferentes navegadores (Restricción: lamentablemente *Angular 4* no es soportado actualmente por todos los navegadores del mercado)

RNF.4 Usabilidad: la aplicación debe ser fácil de usar.

B.4. Especificación de requisitos

En esta sección se explicará el diagrama de casos de uso y se desarrollará cada uno de los requisitos en función del esquema.

Diagrama casos de uso

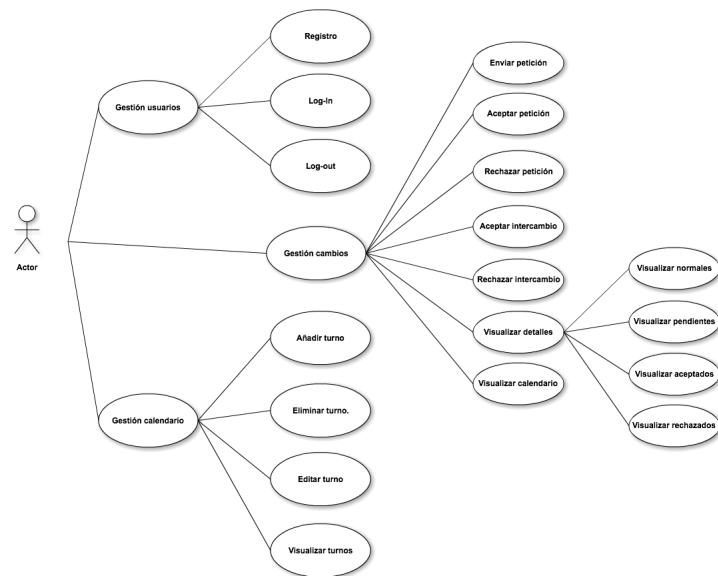


Figura B.1: Casos de uso. Fuente: Elaboración propia.

Actores

El actor es el usuario de la aplicación tenemos que tener en cuenta que para que se de un cambio de turno deben de existir dos actores sino sería imposible el intercambio.

Casos de uso

Nota: El editar usuarios y eliminar usuarios no esta implementado de cara al usuario. Está implementado para realizar en una segunda fase pero no se encuentra disponible en pantalla para realizarse directamente por parte del usuario.

CU-01	Gestión usuarios
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-1.1, RF-1.2, RF-1.3
Descripción	Permite registrarse, logearse al usuario o salir.
Precondición	La página está cargada, base de datos funcionando.
Acciones posibles	1. El usuario se registra. 2. El usuario se logea. 3. El usuario interactúa. 4. El usuario sale.
Postcondición	Se añade el usuario a la base de datos.
Excepciones	
Frecuencia	Alta
Importancia	Alta
Comentarios	

Tabla B.1: CU-0 Gestión usuarios

CU-02	Registro
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-1.1
Descripción	Permite registrarse.
Precondición	La página está cargada en la pestaña adecuada.
Acciones	1. El usuario introduce los datos, a saber: A. Nombre B. Apellidos B. Nombre usuario C. E-mail D. Nombre empresa E. Tipo de turno (a elegir) F. Contraseña
Postcondición	Se añade el usuario a la base de datos. Redirección a log-in.
Excepciones	Si no se introducen todos los campos obligatorios se notifica. También si el usuario ya existe o si todo ha ido correctamente.
Frecuencia	Alta
Importancia	Alta
Comentarios	

Tabla B.2: CU-02 Registro

CU-03	Log-in
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-1.2
Descripción	Permite logearse al usuario
Precondición	La página está cargada en la pestaña adecuada. Base de datos funcionando.
Acciones	1. El usuario se <i>logea</i> para ellos introduce: A. Correo electrónico B. Contraseña 2. Se cargan los turnos en el calendario.
Postcondición	El usuario entra en la aplicación.
Excepciones	Si la contraseña no es correcta, aviso. Si el correo no es correcto, aviso.
Frecuencia	Alta
Importancia	Alta
Comentarios	Si fuera el primer <i>log-in</i> el calendario el calendario estará vacío.

Tabla B.3: CU-03 Log-in

CU-04	Log-out
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-1.3
Descripción	Permite salir de la aplicación.
Precondición	El usuario está dentro de la aplicación.
Acciones	1. Presionar el botón <i>log out</i> .
Postcondición	Sale de la aplicación.
Excepciones	
Frecuencia	Alta
Importancia	Alta
Comentarios	

Tabla B.4: CU-04 Log out

CU-05	Gestión calendario
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-2.1, RF-2.2, RF-2.3, RF-2.4
Descripción	Gestión del calendario.
Precondición	El usuario está dentro de la aplicación.
Acciones	<ol style="list-style-type: none"> 1. Añadir turnos. 2. Eliminar turnos. 3. Editar turnos. 4. Visualizar turnos.
Postcondición	Turno deseado.
Excepciones	
Frecuencia	Media
Importancia	Alta
Comentarios	

Tabla B.5: CU-05 Gestión Calendario

CU-06	Añadir turno
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-2.1
Descripción	Añadir un turno determinado.
Precondición	El usuario está dentro de la aplicación.
Acciones	<ol style="list-style-type: none"> 1. Presionar botón <i>nuevo turno</i>. 2. Seleccionar título. 3. Seleccionar tipo. 4 .Seleccionar fecha.
Postcondiciones	<p>Turno deseado en calendario.</p> <p>Turno añadido en la base de datos asociado al usuario que lo ha añadido.</p> <p>Prioridad normal.</p>
Excepciones	No es posible más de un turno al día.
Frecuencia	Media
Importancia	Alta
Comentarios	

Tabla B.6: CU-06 Añadir turno

CU-07	Eliminar turno
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-2.3
Descripción	Eliminar un turno.
Precondición	El usuario está dentro de la aplicación.
Acciones	1. Presionar botón <i>eliminar turno</i> 2. Turno eliminado.
Postcondiciones	Turno eliminado del calendario. Turno eliminado de la base de datos.
Excepciones	
Frecuencia	Media
Importancia	Alta
Comentarios	

Tabla B.7: CU-07 Eliminar turno

CU-08	Editar turno
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-2.3
Descripción	Editar un turno.
Precondición	El usuario está dentro de la aplicación.
Acciones	1.El usuario visualiza el turno en la pestaña <i>normal</i> . 2. El usuario puede modificar : A.Titulo B.Tipo C.Fecha
Postcondición	Turno deseado editado.
Excepciones	
Frecuencia	Media
Importancia	Alta
Comentarios	El usuario puede modificar un turno siempre que así lo desee.

Tabla B.8: CU-08 Editar turno

CU-09	Visualizar turnos
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-2.4
Descripción	Ver de un simple vistazo todos los turnos.
Precondición	El usuario está dentro de la aplicación.
Acciones	1. Ver turnos.
Postcondición	
Excepciones	
Frecuencia	Media
Importancia	Alta
Comentarios	

Tabla B.9: CU-09 Visualizar turnos

CU-10	Gestión cambios
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-3.1, RF-3.2, RF-3.3, RF-3.4 RF-3.5, RF-3.6, RF-3.6.1, RF-3.6.2 RF-3.6.3, RF-3.6.4, RF-3.7
Descripción	Gestión de los cambios entre usuarios.
Precondición	El usuario está dentro de la aplicación.
Acciones	Hay más de un usuario. 1. Enviar petición de cambio. 2. Aceptar petición de cambio. 3. Rechazar petición de cambio. 4. Aceptar intercambio. 5. Rechazar intercambio. 6. Visualizar detalles. 7. Visualizar calendario.
Postcondición	Cambio turno.
Excepciones	Para diversos casos se muestran avisos.
Frecuencia	Media
Importancia	Alta
Comentarios	Se controlan que la petición se envíe una sola vez. Se muestra la lista de usuarios con turnos libres asociada a un día concreto

Tabla B.10: CU-10 Gestión Cambios

CU-11	Envío de petición de cambio
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-3.1
Descripción	Gestión de los turnos entre usuarios.
Precondición	El usuario está dentro de la aplicación. Hay más de un usuario.
Acciones	1. Enviar petición de cambio.
Postcondición	Ver si el usuario . destinatario a aceptado o rechazado la petición.
Excepciones	Aviso de envío de petición tan solo una vez. Si no hay usuarios libres no es posible enviar la petición
Frecuencia	Media
Importancia	Alta
Comentarios	Se controlan que la petición se envíe una sola vez. Se tienen en cuenta todos los usuarios con turnos libre.

Tabla B.11: CU-11 Enviar petición

CU-12	Aceptar de petición de cambio
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-3.1
Descripción	Aceptación de la petición
Precondición	El usuario está dentro de la aplicación. Hay una petición por parte de otro usuario
Acciones	1. Aceptar petición.
Postcondición	Cambio aceptado.
Excepciones	Petición única
Frecuencia	Media
Importancia	Alta
Comentarios	Se controlan que la petición sea única.

Tabla B.12: CU-12 Aceptar petición

CU-13	Rechazar de petición de cambio
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-3.2
Descripción	Rechazo de la petición
Precondición	El usuario está dentro de la aplicación. Hay una petición por parte de otro usuario
Acciones	1. Rechazar petición.
Postcondición	Fin de la lógica.
Excepciones	
Frecuencia	Media
Importancia	Alta
Comentarios	Se controlan que la petición sea única.

Tabla B.13: CU-13 Rechazar petición

CU-13	Aceptar intercambio
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-3.3
Descripción	Aceptación intercambio
Precondición	El usuario está dentro de la aplicación. Hay una petición por parte de otro usuario
Acciones	1. Aceptar cambio.
Postcondición	Se produce el cambio. Los eventos se intercambian tanto en la base de datos como en el calendario de ambos usuarios
Excepciones	
Frecuencia	Media
Importancia	Alta
Comentarios	

Tabla B.14: CU-14 Aceptar intercambio

CU-15	Rechazar de petición de cambio
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-3.4
Descripción	Rechazo de la petición
Precondición	El usuario está dentro de la aplicación. Hay una petición por parte de otro usuario
Acciones	1. Rechazar intercambio.
Postcondición	Fin de la lógica.
Excepciones	
Frecuencia	Media
Importancia	Alta
Comentarios	

Tabla B.15: CU-15: Rechazar intercambio

CU-16	Detalles peticiones
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-3.6, RF-3.6.1 RF-3.6.2, RF-3.6.3, RF-3.6.4
Descripción	Detalles de las peticiones mediante botones para una mejor comprensión por parte del usuario
Precondición	El usuario está dentro de la aplicación.
Acciones	1. Visualizar normales. 2. Visualizar pendientes. 3. Visualizar aceptados. 4. Visualizar rechazados.
Postcondición	Ver numero de cada parámetro
Excepciones	
Frecuencia	Media
Importancia	Media
Comentarios	No es estrictamente necesaria esta parte pero ayuda al usuario

Tabla B.16: CU-16: Visualizar detalles

CU-16	Detalles turnos
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-3.6.1
Descripción	Detalle de turnos
Precondición	El usuario está dentro de la aplicación y ha realizado algún turno
Acciones	1. Visualizar normales.
Postcondición	
Excepciones	
Frecuencia	Media
Importancia	Media
Comentarios	

Tabla B.17: CU-17: Visualizar normales

CU-16	Detalles peticiones pendientes
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-3.6.2
Descripción	Detalle de peticiones pendientes
Precondición	El usuario está dentro de la aplicación y ha solicitado algún cambio de turno (petición pendiente)
Acciones	1. Visualizar peticiones pendientes.
Postcondición	
Excepciones	
Frecuencia	Media
Importancia	Media
Comentarios	

Tabla B.18: CU-18 Visualizar pendientes

CU-16	Detalles peticiones
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-3.6.3
Descripción	Detalle de peticiones aceptadas
Precondición	El usuario está dentro de la aplicación y ha realizado alguna petición de cambio.
Acciones	1. Visualizar peticiones aceptadas .
Postcondición	Confirmar cambio
Excepciones	Si la petición se ha rechazado no se visualizará aquí.
Frecuencia	Media
Importancia	Media
Comentarios	

Tabla B.19: CU-17: Visualizar normales

CU-20	Detalles peticiones
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-3.6.4
Descripción	Detalle de peticiones rechazadas
Precondición	El usuario está dentro de la aplicación y ha realizado alguna petición de cambio.
Acciones	1. Visualizar peticiones rechazadas .
Postcondición	Rechazar cambio
Excepciones	Si la petición se ha aceptado no se visualizará aquí.
Frecuencia	Media
Importancia	Media
Comentarios	

Tabla B.20: CU-20 Visualizar rechazadas

CU-20	Visualizar calendario
Versión	1.0
Autor	Adrián Aguado
Requisitos asociados	RF-3.6.4
Descripción	Visualizar cambios en el calendario
Precondición	El usuario está dentro de la aplicación.
Acciones	1. Visualizar turnos 2. Click encima de turnos para ver duración 3, Color asociado a cada tipo de turno
Postcondición	
Frecuencia	Media
Importancia	Media
Comentarios	Todos los cambios de turno se deben de ver reflejados en el calendario del usuario en caso de ser aceptados por ambas partes

Tabla B.21: CU-21 Visualizar calendario

Apéndice C

Especificación de diseño

C.1. Introducción

Cada parte de la aplicación tiene características interesante de programación que merece la pena reflejar aquí: comprender qué datos se han usado, cómo se han pensado y el tipo de los mismos resulta fundamental para entender el comportamiento global de la aplicación.

C.2. Diseño de datos

Vamos a analizar los datos así como las técnicas o librerías más importantes de las que se han hecho uso para usar, enviar y recoger todos esos datos. Debemos recordar que se trata de una tecnología basada en componentes, todos los componentes en Angular tienen una estructura interna exactamente igual:

```
//app/todos/todos-list/todo-list.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'todo-list',
  templateUrl: 'todo-list.component.html',
  styleUrls: ['todo-list.component.css'],
  moduleId: module.id
})
export class TodoListComponent { ... }
```

Figura C.1: Componente Angular . Fuente: <http://blog.enriqueoriol.com/>.

En la imagen C.2 podemos ver que se define un **selector** que es la parte para introducir ese determinado componente en la plantilla, un **templateUrl** que define la propia plantilla html asociada a ese componente, **stylesUrls** que define la hoja de estilos asociada a esa plantilla y **module.id** que en nuestro

caso no tiene demasiada importancia. Los únicos estrictamente obligatorios son los dos primeros.

Si nos fijamos en las buenas prácticas de programación de Angular dictan que después del nombre de un componente viene la palabra *component*, pasa lo mismo con, por ejemplo, los servicios cuya palabra en este caso será *services*.

En la siguiente imagen C.2 podemos ver la relación entre un componente y su propia plantilla.

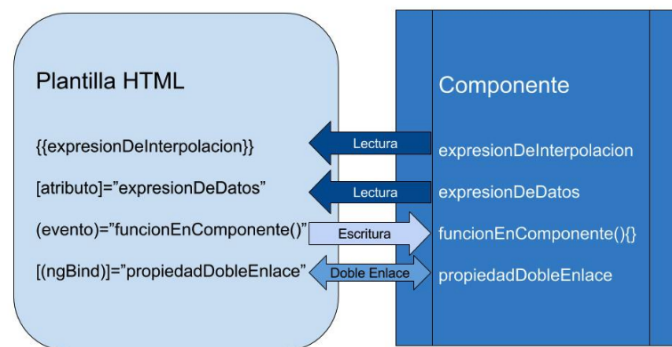


Figura C.2: Relación plantilla-componente. Fuente: <http://academia-binaria.com/>.

Vamos a analizar cada una de las partes esenciales de la aplicación:

- */app/directives* : las directivas en este caso se utilizan para lanzar alertas, concretamente en el registro o de la aplicación.
- */app/guards* : es la parte donde realmente se comprueba que un usuario este logeado en la web [18]. Me basé en un tutorial [19] de Jason Watmore, programador australiano.
- */app/models* : junto con *services* una de las partes fundamentales de la aplicación es dónde se describen los datos donde se declaran los atributos que va a tener cada entidad que vamos a usar. En mi caso: usuarios y eventos. Se ha usado aquí un DTO [20] para el caso de los eventos que van a ser intercambiados, es decir, ese turno que un usuario va a intercambiar con otro. Es habitual esta práctica de programación en java pero también empieza a serlo en Angular [7].

A continuación vamos a ver cada una de las clases que componen y en las que se basa la aplicación

Tipo de dato	Parámetros
Usuario [user]	id: string email: string password: string cname: string firstName: string lastName: string shift: string username: string

Tabla C.1: Clase Usuario

Tipo	Parámetros
Evento [event.dto]	id: string start: Date end: Date title: string primary_color: string secondary_color: string draggable: boolean resizable_beforeStart: boolean resizable_afterEnd: boolean company: string type: string username: string status: string sender: string turn_in_day: string

Tabla C.2: Clase Evento.

Tipo	Parámetros
Evento del usuario [my-event]	id: string id: string type: string username: string status: string sender: string turn_in_day: string

Tabla C.3: Clase Evento-usuario.

Tipo	Parámetros
Evento a intercambiar [interchange.dto]	id: string requestor: string acknowledger: string requestor_event_id: string acknowledger_event_id: string status: string;

Tabla C.4: Clase Intercambio.

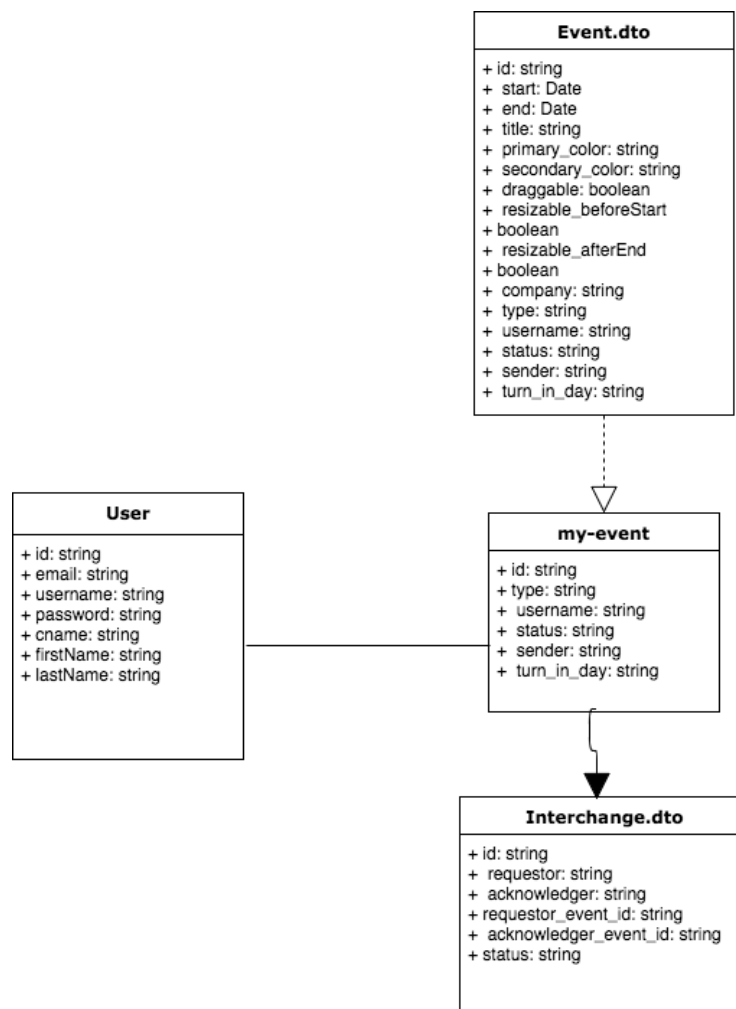


Figura C.3: Diagrama relacional. Fuente: Elaboración propia.

- `/app/services` : los servicios de la parte cliente, las alertas nombradas anteriormente o la autenticación tienen su propio servicio. Es interesante

el conocer aquí el concepto de *Observable* [4] en Angular, a la hora de las comunicaciones http es algo muy a tener en cuenta. La librería **@angular/http** trae por defecto el servicio http que es el cliente usado para enviar y recibir datos [6], pero sin duda la librería más potente actualmente sobre el tema es **RxJS**. Recomiendo encarecidamente el artículo: Javascript Reactivo y funcional [5], que aborda estos temas con gran entereza.

- */app/pages* : comprende las distintas páginas *físicas* de la web.
- */app/repository* : es la conexión de datos entre la parte del cliente y la parte servidor.

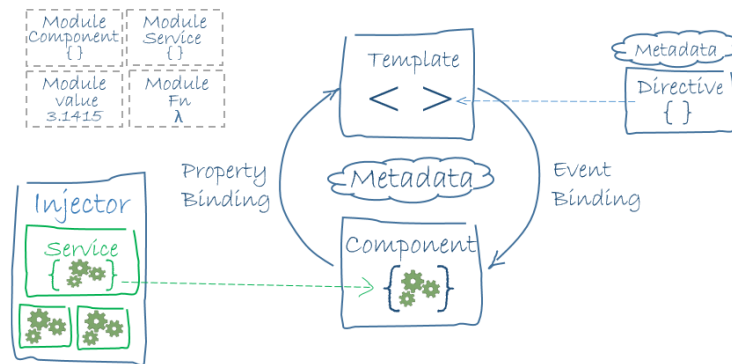


Figura C.4: Esquema Angular por dentro. Fuente: www.angular.io.

- */app/utils* : es una carpeta extra que pretende emplear el patrón *Singleton* definiendo datos inamovibles durante toda la vida de la aplicación. Se utiliza el nombre utils como buena práctica de programación ya que se trata de datos útiles que no van a cambiar nunca. De esta manera si queremos añadir datos nuevos en un futuro tan solo hay que añadirlos dentro de este componente.

Definición	Datos
Tipos de turno [shifts]	24 horas
	12 horas
	8 horas
	6 horas
Tipos de eventos [types]	Intocable
	Asignado conforme
	Asignado eliminar
	Día libre
Colores asociados a los eventos [colors]	Rojo (Intocable)
	Azul (Asignado conforme)
	Amarillo (Asignado eliminar)
	Verde (Día libre)
Turnos /día [turnInDay]	1 turno por día
	2 turnos por día
	3 turnos por día
	4 turnos por día
Estados de un evento [eventStatus]	Normal
	Pendiente
	Aceptado
	Rechazado
	Requerido

Tabla C.5: Datos.

En términos de programación todos son arrays excepto los turnos por día que se declaran como un mapa cuya clave es el argumento uno, dos, tres y cuatro del array *shifts* que corresponde al tipo de turno, y el valor son los turnos en los que se subdivide un día: jornada completa, dos turnos, tres turnos o cuatro turnos.

- *Calendario* : el calendario es la parte fundamental de la aplicación, se ha empleado un componente desarrollado por [Matt Lewis](https://github.com/mattlewis92/angular-calendar). Un desarrollador británico que tiene numerosas aportaciones al mundo *open source*. El calendario está en desarrollo actualmente como se puede ver en su página de [github](https://github.com/mattlewis92/angular-calendar).
 - <https://github.com/mattlewis92/angular-calendar>
- Otros detalles a tener en cuenta sobre el cliente.
 - *ngAfterViewInit* [8] : para intentar entenderlo diremos que es algo que es necesario que se inicialice antes de la vista principal.

- *PrimeNg*: se trata de una serie de componentes diseñados para Angular que resultan ser realmente útiles. Persiguen el diseño *responsive* de cara al usuario y tienen una comunidad enorme detrás [17]. Por ejemplo yo he empleado Growl [16] que se utiliza para lanzar al usuarios los mensajes de confirmación de las diferentes peticiones.
- */server/config* : se define aquí la configuración de la base de datos, local o remota, necesaria para almacenar los datos de la aplicación. Este fichero junto con **config.js**, que se encuentra en la raíz de la carpeta *server*, son los que hay que modificar para cambiar de una base de datos a otra. En un futuro no muy lejano se creará una variable de entorno para que sea por defecto capaz de detectar si es necesario una base de datos local o una remota.
- */server/controllers* : son los controladores de la parte del servidor. Emplean la librería *Router* [15] nativa en express, para poder funcionar.

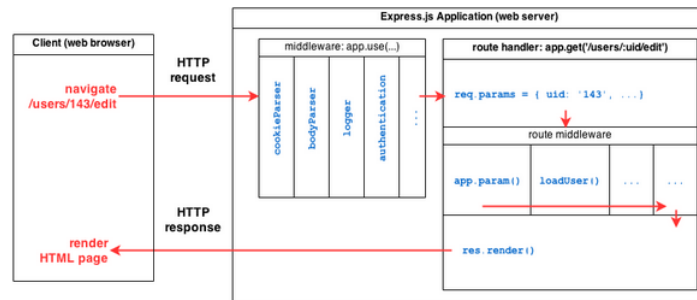


Figura C.5: Envío-respuesta http . Fuente: <http://expressjs.com/>.

- */server/routes* : son las rutas necesarias para realizar correctamente las llamadas *http* y que devuelvan en cada momento los datos correspondientes. Una vez el usuario se haya autenticado de forma correcta, ya puede acceder a las diferentes zonas de la aplicación. Cada sección dentro esta aplicación necesitará al menos una llamada a la API, situadas en rutas privadas, es decir, que necesitan autorización. Sin esa autorización no se pueden realizar las llamadas *http* necesarias.

En el siguiente par de páginas vamos a ver de un vistazo rápido los métodos empleados para extraer los datos y los métodos *http* que tiene asociado en cada ruta cada uno de ellos.

Tipo	Métodos	Método http asociado
Eventos	.create_event()	POST
	.create_events()	PUT
	.list_all_events()	GET
	.list_all_events_by_company()	GET
	.list_all_events_by_company_by_worker()	GET
	.list_all_free_events_by_company_by_day_by_shift_except_currentUser()	GET
	.update_event()	UPDATE
	.delete_event()	DELETE

Tabla C.6: Métodos para *event* y http asociados.

Tipo	Métodos	Método http asociado
Intercambio	.accept_shift()	POST
	.decline()	POST
	.activate_shift()	POST

Tabla C.7: Métodos para *interchange* y http asociados.

Tipo	Métodos	Método http asociado
Usuarios	.authenticate(email, password)	POST
	.register()	POST
	.getAll()	GET
	.getCurrent()	GET
	.update(id)	PUT
	.delete(id)	DELETE

Tabla C.8: Métodos para *users* y http asociados.

- `/server/services` : es la parte del servidor dónde realmente se producen todas las acciones, el intercambio entre usuarios esta implementado aquí. La idea es usar una **API**, es decir, *un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción* [21]. En definitiva es una forma de describir la forma en la que los sitios web intercambian datos, en este caso el intercambio de datos es mediante **JSON**. Si detallamos un poco más veremos que lo que realmente esta implementado en el servidor es una **API REST**, es decir un tipo de arquitectura (**RE**presentational **S**tate **T**ransfer) de desarrollo web que se apoya en una API y en el estándar *http*, por lo que todas las llamadas a la API se implementan como peticiones *http*.

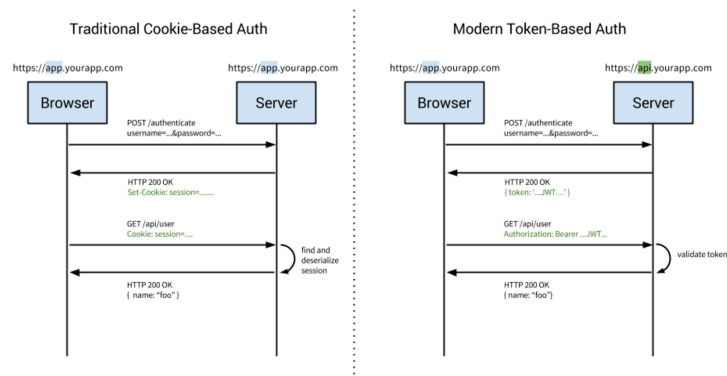


Figura C.6: Funcionamiento API. Fuente: <https://juanda.gitbooks.io/>.

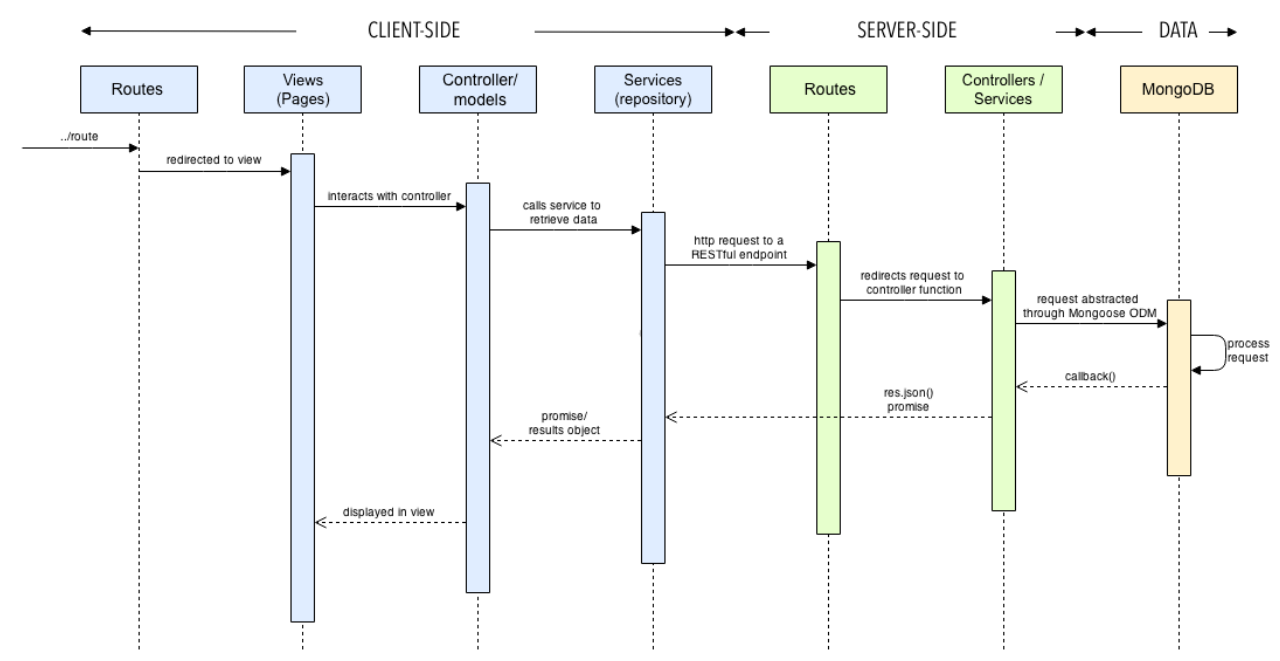


Figura C.7: Vista de la conexión entre el servidor y el cliente. Fuente: Elaboración propia.

C.3. Diseño arquitectónico

En cuanto a la arquitectura de Angular resulta, después de todo lo ya visto, entenderse de una manera fácil e intuitiva. La aplicación compuesta por tres capas siguiendo la arquitectura MVC (Modelo-Vista-Controlador):

- Modelo: es la capa que se comunica directamente con la base de datos. Se encarga de todos los procedimientos necesarios para crear, eliminar, recuperar y actualizar información de la base de datos.
- Vista: capa de presentación, implementa los métodos relacionados con los componentes de la interfaz de la aplicación.
- Controlador: se encarga de la comunicación entre la vista y el modelo. En la aplicación parte de la lógica se manejará dentro de la propia página utilizando JavaScript dentro de los controladores de Angular y otra parte se hará en el servidor de NodeJS.

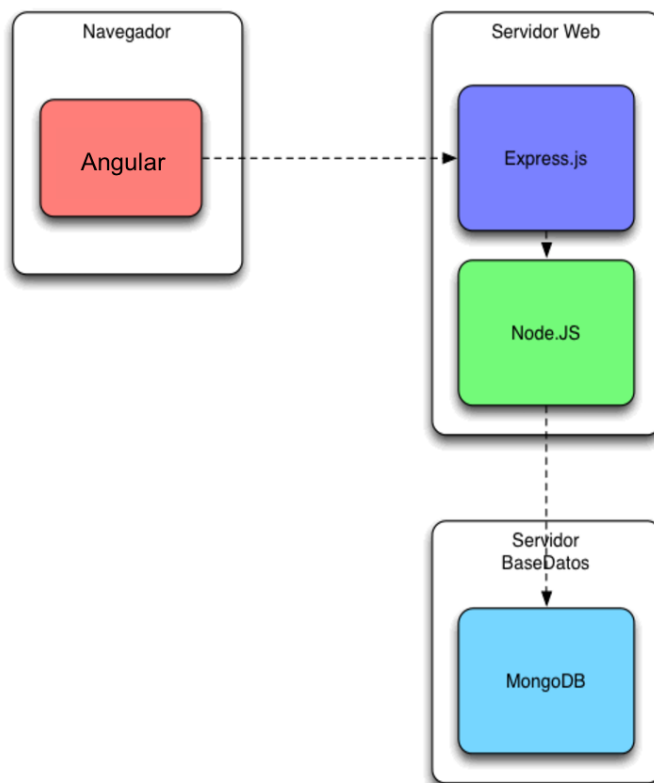


Figura C.8: Esquema Arquitectura. Fuente: <http://www.cantabriatic.com/>.

Seguir la estructura MVC es muy útil, ya que proporciona una idea mental de dónde situar cada elemento evitando así la tarea de investigación cada vez que se quiera ampliar el contenido de la aplicación.

Documentación técnica de programación

D.1. Introducción

Este capítulo vamos a dividirlo en dos partes: por un lado las condiciones específicas propias de la aplicación y por otro vamos a aprender de manera breve y sencilla a crear una aplicación gracias a *Angular CLI*, la interfaz de línea de comandos de Angular.

He decidido incluir la parte de *Angular CLI* porque considero que va a esclarecer muchas dudas acerca de cómo funciona esta tecnología y también para intentar comprender mejor como he realizado el proyecto. En diversas ocasiones para aprender a usar una tecnología tan solo hace falta tiempo pero en otras muchas el tiempo es limitado por lo que quizás con esta pequeña guía podemos comenzar a utilizar un *framework* cliente en muy pocos pasos y de una manera sencilla. Si bien es cierto que yo he necesitado mucho tiempo para comprender su funcionamiento y no lo he aprendido gracias a la interfaz de comandos. Ésta te permite agilizar los trámites de creación en un tanto por ciento considerablemente alto.

Este anexo por tanto tiene como objetivo analizar y documentar las necesidades funcionales que deberán ser soportadas por el sistema a desarrollar, es decir, en qué condiciones ha sido desarrollado, en qué condiciones se debe usar y cuáles son los requerimientos mínimos para que un futuro programador interactúe con la aplicación en caso de que así lo considere.

D.2. Requerimientos mínimos necesarios

Además de un *IDE* [22] para poder trabajar, los prerequisites mínimos para empezar a trabajar en el proyecto son:

1. Instalar [Nodejs](#) y [MongoDB](#).
2. Instalar [npm](#)
3. Instalar Angular CLI

```
npm i -g @angular/cli
```

Nota: es interesante, una vez instalados asegurarnos de que están correctamente instalados usando los comandos `node -v` y `npm -v` para saber si tenemos los requerimientos correctamente instalados.

D.3. AngularCLI

Sabiendo los requisitos mínimos que son necesarios para proceder vamos a realizar, como he nombrado antes una breve introducción a la línea de comandos de Angular, **Angular CLI**.

¿Qué es typescript?

Angular promueve el uso de TypeScript como lenguaje de partida, esto se ha nombrado antes pero resulta esencial para entender el código de Angular. Antes de nada decir que en Angular se puede trabajar con Javascript “clásico” (ES5) [23], así como ES6 y TypeScript. Desarrollado por Microsoft y como apuesta de Google vamos a ver el crecimiento exponencial de este lenguaje, que no deja de ser un super conjunto de JavaScript. TypeScript es por tanto un superset de ECMAScript 6, es decir, incluye todas las funcionalidades de ES6, y además incorpora una capa por encima con funcionalidades extra.

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. Any browser. Any host. Any OS. Open source.

Typescript Web [12]

La principal característica de TypeScript por encima de Javascript es que permite definir de qué tipo son las variables que se van a usar.

Comandos básicos

Angular CLI nos va a hacer la vida más fácil con Angular, ya que nos permite crear de forma sencilla un aplicación lista para funcionar después de la instalación por línea de comandos con *npm*, además incorpora las buenas prácticas de programación con Angular.

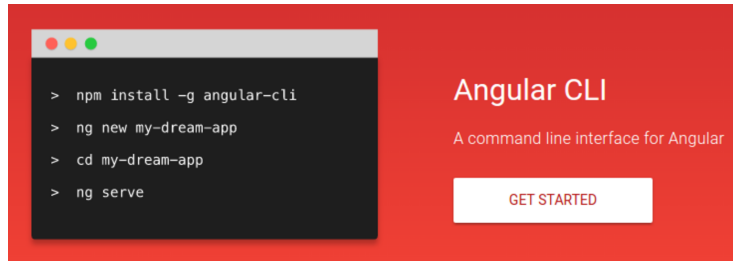


Figura D.1: Comandos *AngularCLI*. Fuente: <https://cli.angular.io/>.

Por lo tanto para crear nuestra aplicación con esta herramienta, una vez instalada, tan solo debemos ir a nuestra carpeta donde queremos crear el proyecto e introducir en la línea de comandos:

```
ng new angular-cli-primer-proyecto
```

Es un proceso que durará unos 2 minutos. Al finalizar seremos capaces de ejecutar directamente la aplicación por defecto que crea la herramienta. Para ejecutarla, al igual que antes accedemos a la carpeta del proyecto y ejecutamos:

```
npm start
```

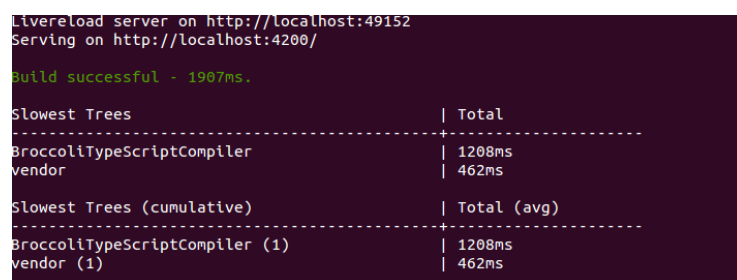


Figura D.2: Consola ejecución. Fuente: <http://codigoxules.org/>.

Se puede ver en la consola de ejecución (D.3) como se llama al comando **ng serve** que será el que utilizemos con Angular CLI para trabajar. Resulta realmente útil al introducir el último comando nombrado ya que una vez lanzado se detectan automáticamente los cambios a medida que vamos avanzando la página se recarga al guardar el componente que estemos modificando.



Figura D.3: Resultado después de creado. Fuente: <http://codigoxules.org/>.

Esta es la sencilla manera de crear un proyecto con **Angular CLI**. Después existen una serie de comandos para crear directamente componentes, servicios, clases, interfaces.. etcétera. Algunos de ellos son:

Tipo a crear	Comando
Component	<i>ng component my-new-component</i>
Directive	<i>ng g directive my-new-directive</i>
Pipe	<i>ng g pipe my-new-pipe</i>
Service	<i>ng g service my-new-service</i>
Class	<i>ng g class my-new-class</i>
Interface	<i>ng g interface my-new-interface</i>
Enum	<i>ng g enum my-new-enum</i>

Tabla D.1: Tabla comandos Angular CLI

D.4. Manual del programador

En esta sección hay que tener en cuenta que el autor de este trabajo proyecto a escogido una serie de herramientas, tanto para desplegar la app, como la base de datos como para desarrollar la aplicación pero que de ninguna manera resultan ser ni las únicas ni las mejores, simplemente son unas herramientas que ha considerado utilizar pero existen otras posibilidades que no son ni peores ni mejores.

Prerequisitos

1. Instalar [Nodejs](#) y [MongoDB](#).
2. Instalar [npm](#)

3. Instalar Angular CLI

```
npm i -g @angular/cli
```

4. Descargar el proyecto (*git clone*) o bien desde un soporte.
5. Una vez tenemos el proyecto descargado en nuestro sistema local desde el directorio raíz (vía línea de comandos) instalar las dependencias necesarias.

```
npm install
```

Estructura de directorios

La estructura de directorios general es la siguiente:

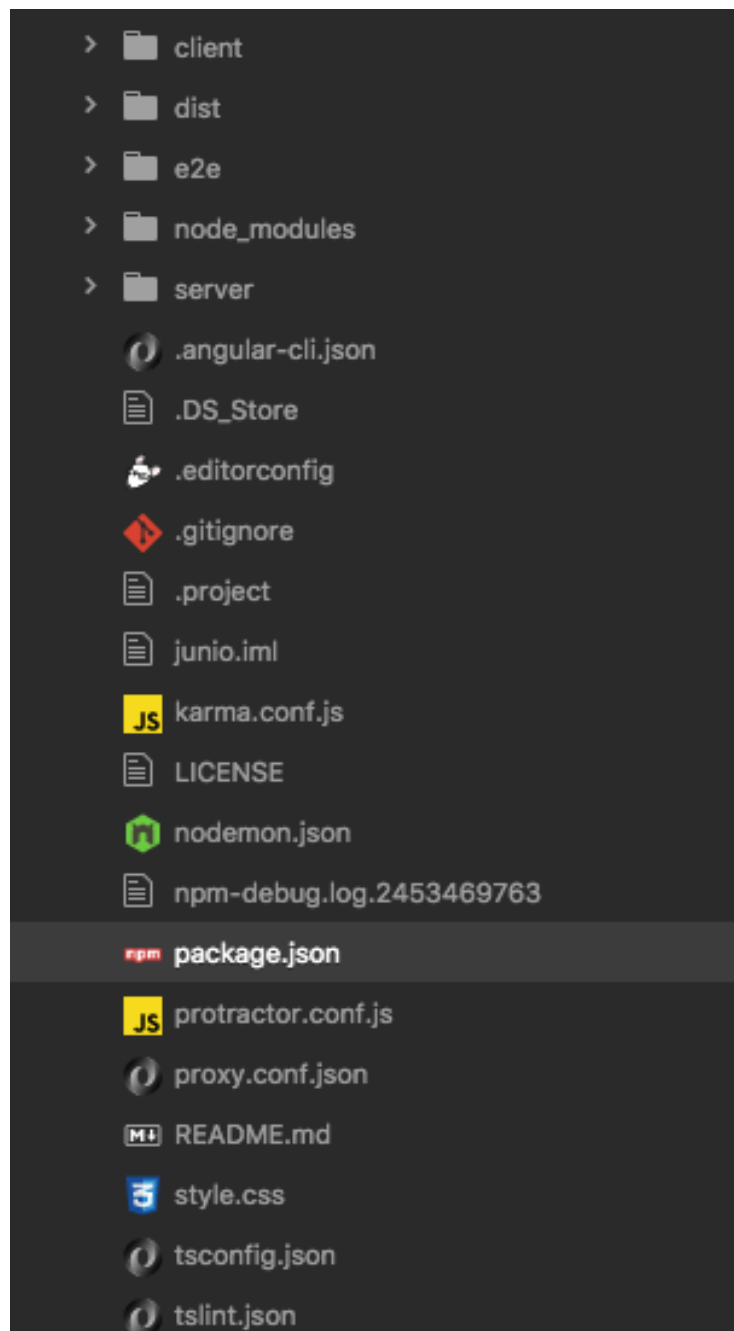
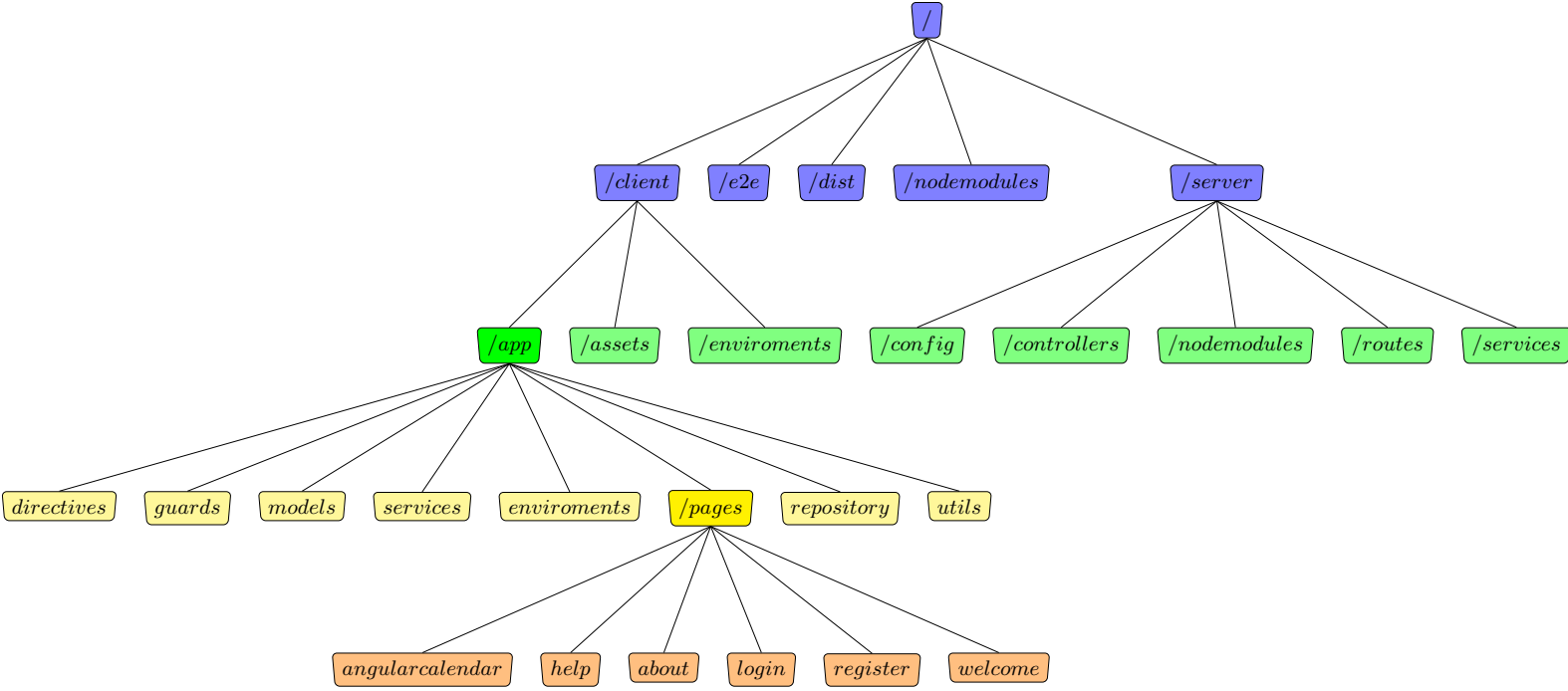


Figura D.4: Vista carpetas general *BarterAPP*. Fuente: Elaboración propia.

- */client/*: *Front-End*
 - */client/app*: Carpeta App

- */client/app/directives*: directivas
 - */client/app/guards*: rutas (autenticaciones)
 - */client/app/models*: modelos
 - */client/app/services*: servicios
 - */client/app/enviroments*: entornos
 - */client/app/pages/*: páginas
 - ◇ */client/app/pages/angularcalendar*: calendario
 - ◇ */client/app/pages/about*: about us
 - ◇ */client/app/pages/help*: help
 - ◇ */client/app/pages/login*: login
 - ◇ */client/app/pages/register*: registro
 - ◇ */client/app/pages/welcome*: vista inicial
 - */client/app/repository*: repositorio (conexión con el servidor)
 - */client/app/utils*: útiles
- */client/assets*: recursos
- */client/enviroments*: entornos
- */dist/*: ficheros para subir a servidor web
- */e2e/*: tests
- */nodemodules/*: módulos
- */server/*: *Back-End*
 - */server/config*: configuración
 - */server/controllers*: controladores
 - */nodemodules/*: módulos
 - */server/routes*: rutas
 - */server/services*: servicios



Modo desarrollador

Hay que tener en cuenta que tenemos diversas tecnologías dentro del proyecto por lo que resulta un poco costoso tener que ejecutar todas por separado, que es lo que hacía en un principio. Con el modo desarrollador (emula al **ng serve** nombrado antes) tan solo con el comando siguiente ejecutamos MongoDB, Angular, Express y el compilador de Typescript luego ya no nos hace falta nada más y podemos comenzar a trabajar.

```
npm run dev
```

Servidor (Heroku)

Para subir la aplicación al servidor una buena elección es [Heroku](#) y los pasos para lanzar la aplicación son los que se enumeran a continuación:

1. Acceder a la [web](#) de *Heroku*, registrarse y crear una nueva app.
2. Instalar [Heroku CLI](#)
3. Ejecutar los siguientes comandos:

```
heroku login
cd my-project/
git init
heroku git:remote -a your-app-name
```

4. Descargar el código fuente.
5. Editar la configuración(Ruta: *server/config/db.ts*) para introducir un servidor para una base de datos *real* con MongoDB (lo veremos en la sección siguiente)-
6. Ejecutar los siguientes comandos:

```
npm i
ng build -prod or ng build -aot -prod
tsc -p server
git add .
git commit -m "Going_to_Heroku"
git push heroku master
heroku open
```

7. Una venta se abrirá con tu aplicación ya online.

Resumiendo de esta manera utilizamos Heroku como si de github se tratase y vamos aplicando los cambios que hacemos a nuestra aplicación al repositorio

creado en Heroku, la ventaja que ya tenemos la aplicación lanzada en un entorno real. Es una restricción importante que ya no podemos trabajar con una base de datos local sino que tiene que ser un servidor remoto con MongoDB (en este caso) para que la aplicación funcione.

Base de datos (Mlab)

La base de datos escogida es MongoDB, la instalación es fácil y sencilla tan solo hay que ir a la [web](#) oficial y descargar la última versión estable disponible. Una vez en nuestro ordenador la ejecutamos y listo. Si surge algún problema recomiendo seguir alguno de los tutoriales que existen en la web ya que pueden aparecer errores comunes pero fácilmente solucionables [1].

Como sabemos podemos manejar la base de datos desde línea de comandos pero resulta un poco pesado cuando empezamos a tener muchas colecciones o un número de datos alto. Es por eso que yo recomiendo encarecidamente [Robomongo](#), la copia de SQL server para bases de datos NoSQL. Interfaz no muy lograda pero perfecta para visualizar nuestras bases de datos e información de una manera más intuitiva.

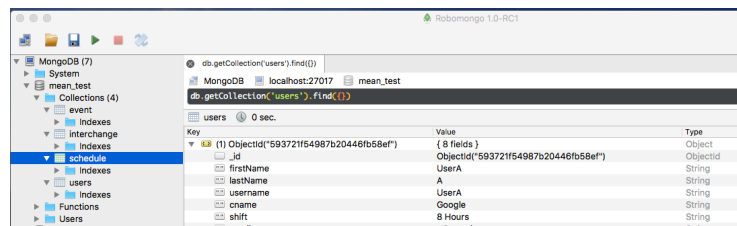


Figura D.5: Vista Robomongo. Fuente: Elaboración propia.

Esta claro que cuando trabajamos con un proyecto en local para realizar pruebas lo mejor es tener una base de datos en local pero cuando queremos algo un poco más serio resulta interesante el tener esa base de datos en un servidor real. Para ello he escogido [Mlab](#), los pasos para crear la base de datos se enumeran a continuación:

1. Acceder a la [Mlab](#), registrarse y crear una cuenta
2. Crear una subscripción a una base de datos (Tiene de pago pero también tiene una versión gratuita que nos vale para realizar pruebas a este nivel)
3. Ejecutar el siguiente comando para establecer la conexión con la base de datos creada

```
mongo ds012345.mlab.com:56789/dbname -u
dbuser -p dbpassword
```


4. Las colecciones las podemos crear directamente desde la propia web, o bien desde la línea de comandos
5. Para probar que la base de datos funciona (El primer comando es para insertar un elemento y el segundo para buscar. En este caso el segundo debería devolver el único elemento insertado.:

```
db.mynewcollection.insert({ "foo" : "bar" })
db.mynewcollection.find()
```

Nota: para empezar a trabajar con este proyecto es necesario crear las tres colecciones de las que ya hemos hablado en secciones anteriores: *Users*, *Events*, *Interchange*, y no hacer nada más puesto que según la configuración de la aplicación se debería añadir los usuarios, eventos y turnos de manera automática cuando el usuario hacer click en los botones adecuados.

Servidor (Nanobox)

La posibilidad de subir a un servidor como Heroku es fantástica pero he tenido algún que otro problema por lo que finalmente elegí otro servidor, en este caso [Nanobox](#) que funciona con [Digital Ocean](#). Los servicios no son gratuitos pero ha merecido la pena.

La curva de aprendizaje no es demasiado fácil [14] pero tampoco demasiado complicada, además el equipo humano que hay detrás de Nanobox te hace que los problemas que van surgiendo parezcan sencillos. Para subir a Nanobox tu aplicación hay que realizar los siguientes pasos:

1. Crear cuenta en Nanobox.
2. Elegir un plan entre los disponibles.
3. Elegir un *host*. Recomendando **Digital Ocean**.
4. Ahora, sabiendo la tecnología a usar, nodejs para el servidor en este caso, se crea un fichero de **boxfile.yml** en la raíz del proyecto[11] . Es el equivalente a el **Procfile** de Heroku.
5. A continuación los comandos:

```
-nanobox run
-nanobox remote barterapp
-nanobox deploy
```

Éstos son a grandes rasgos los pasos mínimos para desplegar una aplicación en esta herramienta. Una de las mejores cosas y por la cual conseguí solucionar mi problema con el servidor es el magnífico sistema de *logs* con el que cuenta la plataforma. La diferencia más destacable con Heroku es que en Nanobox subes absolutamente todos los ficheros con los que estas trabajando, por contra en Heroku solo la carpeta */dist* generada automáticamente por Angular. Cada uno tiene sus ventajas y sus inconvenientes.

Procfile to boxfile.yml

Heroku uses the `Procfile` to define the processes to include with your app and how to run them. Nanobox uses the `boxfile.yml` to define your app's runtime, configuration, and supporting services. Just as with the `Procfile` for Heroku, your Nanobox `boxfile.yml` needs to be included in the root of your project.

Figura D.6: Documentos configuración [10]. Fuente: <http://nanobox.io/>.

Documentación de usuario

E.1. Introducción

En este capítulo se detalla como un usuario puede comenzar a usa la aplicación deberemos diferenciar dos aspectos diferentes:

- Aplicación Web.
- Aplicación móvil.

E.2. Usuario Web

Se describe lo que puede hacer el usuario con la web.

Requisitos mínimos

Registro

Login

Añadir turnos al calendario

Intercambiar turno

- Enviar petición
- Aceptar/Rechazar petición
- Confirmar petición

Ayuda

E.3. Usuario Móvil

Registro

Login

Añadir turnos al calendario

Intercambiar turno

- Enviar petición
- Aceptar/Rechazar petición
- Confirmar petición

Ayuda

some thing	some thing	some thing	some thing
------------	------------	------------	------------

Bibliografía

- [1] Francisco Javier Martínez Páez Adictos al trabajo. Mongodb, primeros pasos., 2014. URL <https://www.adictosaltrabajo.com/tutoriales/mongodb/>. (Citado en página 52.)
- [2] Apache-Github. Apache license 2.0, 2004. URL <https://choosealicense.com/licenses/apache-2.0/>. (Citado en página 11.)
- [3] Ben Balter. Open source license usage on github.com, 2015. URL <https://github.com/blog/1964-open-source-license-usage-on-github-com>. (Citado en página 11.)
- [4] Alberto Basalo-Academia Binaria. Comunicaciones http observables con angular2, 2017. URL <http://academia-binaria.com/comunicaciones-http-observables-con-angular2/>. (Citado en página 34.)
- [5] CodekStudio. Javascript reactivo y funcional. conceptos: Observables, rxjs y angular 2., 2016. URL <https://codekstudio.com/post-blog/conceptos-observables-rxjs-y-angular-2-javascript-reactivo-y-funcional/>. (Citado en página 34.)
- [6] Stackoverflow Community. How to do polling with angular 2 observables, 2015. URL <https://stackoverflow.com/questions/41658162/how-to-do-polling-with-angular-2-observables>. (Citado en página 34.)
- [7] Stackoverflow Community. Dto design in typescript angular2, 2016. URL <https://stackoverflow.com/questions/39272947/dto-design-in-typescript-angular2>. (Citado en página 31.)

- [8] Angular Official Documentation. Afterviewinit : Interface, 2016. URL <https://angular.io/api/core/AfterViewInit>. (Citado en página 35.)
- [9] Github. Choose an open source license, 2016. URL <https://choosealicense.com/>. (Citado en página 11.)
- [10] Nanobox Official guide. Migrating apps from heroku to nanobox., 2017. URL <https://content.nanobox.io/migrating-apps-from-heroku-to-nanobox/>. (Citado en páginas III y 54.)
- [11] Nanobox Official guide Nodejs + Expressjs. Install and run a express app from scratch in 5 mins., 2017. URL <https://guides.nanobox.io/nodejs/express/>. (Citado en página 53.)
- [12] Microsoft. Offical web page typescript, 2012. URL <https://www.typescriptlang.org/>. [Online; Accedido 03-Mayo-2017]. (Citado en página 44.)
- [13] MIT-Github. Mit license, 2017. URL <https://choosealicense.com/licenses/mit/>. (Citado en página 11.)
- [14] Nanobox Official Page: node js. Examples of nodejs in nanobox, 2017. URL <https://guides.nanobox.io/nodejs/>. (Citado en página 53.)
- [15] Express Guía Oficial. Direccionamiento http., 2015. URL <http://expressjs.com/es/guide/routing.html>. (Citado en página 36.)
- [16] PrimeNG Official Page. Growl is used to display messages in an overlay., 2015. URL <https://www.primefaces.org/primeng/growl>. (Citado en página 36.)
- [17] PrimeNG Official Page. The most complete user interface suite for angular, 2015. URL <https://www.primefaces.org/primeng/>. (Citado en página 36.)
- [18] Thoughttram'. Protecting routes using guards in angular, 2016. URL <https://blog.thoughttram.io/angular/2016/07/18/guards-in-angular-2.html>. (Citado en página 31.)
- [19] Jason Watmore'. Angular user registration and login example. tutorial, 2017. URL <http://jasonwatmore.com/post/2016/09/29/angular-2-user-registration-and-login-example-tutorial>. (Citado en página 31.)
- [20] Wikipedia'. Objeto de transferencia de datos (dto)- wikipedia, 2015. URL [https://es.wikipedia.org/wiki/Objeto_de_Transferencia_de_Datos_\(DTO\)](https://es.wikipedia.org/wiki/Objeto_de_Transferencia_de_Datos_(DTO)). (Citado en página 31.)

- [21] Wikipedia'. Interza de programacion de aplicaciones- wikipedia, 2017. URL https://es.wikipedia.org/wiki/Interfaz_de_programaci\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{o\global\mathchardef\accent@spacefactor\spacefactor}\accent19o\egroup\spacefactor\accent@spacefactor\futurelet\@let@token\penalty\@M\hskip\z@skipn_de_aplicaciones. (Citado en página 39.)
- [22] Our Code World. Best free web development ide for javascript, html and css, 2016. URL <http://ourcodeworld.com/articles/read/200/top-7-best-free-web-development-ide-for-javascript-html-and-css>. (Citado en página 44.)
- [23] Juriy Zaytsev. Compatibility table es5, 2016. URL <http://kangax.github.io/compat-table/es5/>. (Citado en página 44.)