

Capítulo 1

Introducción

En este TFG se ha desarrollado una aplicación web, que pone en contacto profesores y alumnos para dar clases particulares, utilizando tecnologías web de última generación.

1.1. Tecnologías web de última generación

En este capítulo se introducirá en el contexto de las tecnologías web que marcan el panorama actual.

Las aplicaciones web han ido evolucionando a lo largo de la historia de internet, pero todas ellas se basan en un modelo cliente-servidor, es decir, para poder lograr la comunicación necesitamos un cliente, normalmente un navegador y un servidor web capaz de atender nuestras solicitudes.

Todas ellas se basan en el protocolo HTTP, el cual permite las transferencias de información en la World Wide Web y define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse.

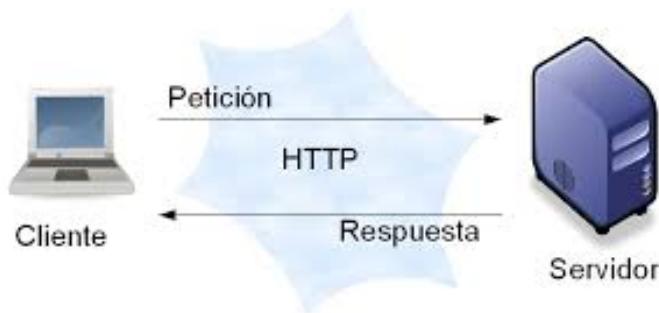


Figura 1.1: Modelo cliente-servidor

HTTP/2

HTTP/2 es la versión más actualizada de Hypertext Transfer Protocol, el protocolo de comunicación que permite la transferencia de información en la red. Es un protocolo orientado a transacciones y que sigue un esquema petición-respuesta entre un cliente y un servidor.

HTTP define una serie predefinida de métodos de petición(verbos) que pueden utilizarse, teniendo la flexibilidad de ir añadiendo nuevos métodos con sus nuevas funcionalidades.

Entre todos los métodos de petición de HTTP/2 destacamos los siguientes:

- **HEAD:** El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.
- **GET:** El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.
- **POST:** El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.
- **PUT:** El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
- **DELETE** El método DELETE borra un recurso en específico.

HTTP es un protocolo sin estado, es decir, no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener estado por lo que se utilizan las cookies, que es información que un servidor puede almacenar en el sistema cliente para simular la noción de la sesión.

API REST

API REST se define por ser un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP. La arquitectura de un sitio Web tiene tres componentes principales:

- **Un servidor Web:** Distribuye páginas de información formateada a los clientes que las solicitan. Los requerimientos son hechos a través de una conexión de red, y para ello se usa el protocolo HTTP.
- **Una conexión de red**
- **Uno o más clientes:** Una vez que se solicita esta petición mediante el protocolo HTTP y la recibe el servidor Web, éste localiza la página Web en su sistema de archivos y la envía de vuelta al cliente que la solicitó.

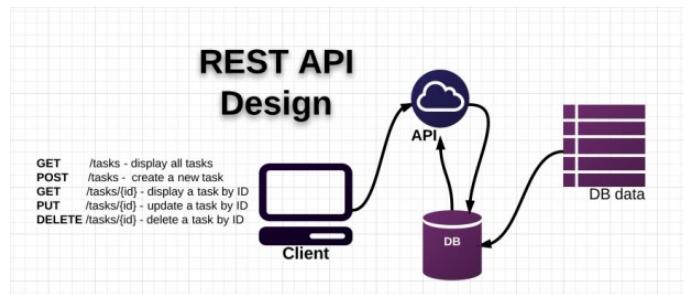


Figura 1.2: API-REST

Para comprender el concepto API-REST, primero debemos entender el concepto API. Una API es una interfaz de programación de aplicaciones (del inglés API: Application Programming Interface), que en su conjunto de rutinas provee acceso a funciones de un determinado software.

REST(Transferencia de Estado Representacional) es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad pero también mucha complejidad. A veces es preferible una solución más sencilla de manipulación de datos como REST.

Las reglas que definen una API-REST son las siguientes:

- **Interfaz uniforme:** para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI.
- **Peticiones sin estado** cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla.
- **Cacheable** existe la posibilidad de definir algunas respuestas a peticiones HTTP concretas como cacheables, con el objetivo de que el cliente pueda ejecutar en un futuro la misma respuesta para peticiones idénticas.
- **Separación de cliente y servidor**
- **Sistema de Capas** arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.

La arquitectura API-REST donde las comunicaciones son más ligeras entre productor y consumidor, mantenibles y escalables, hacen de REST un estilo de construcción popular para APIs basadas en la nube, como las proporcionadas por Amazon, Microsoft y Google.

El estilo REST hace énfasis en que las interacciones entre los clientes y los servicios se mejoran al tener un número limitado de operaciones (verbos). La flexibilidad se obtiene asignando recursos a sus propios identificadores de recursos universales únicos (URI). Debido a que cada verbo tiene un significado específico (GET, POST, PUT y DELETE), evitando la ambigüedad.

- **GET** Se usa GET para obtener un recurso
 - **POST** Se usa POST para crear un recurso en el servidor
 - **PUT** Se usa PUT para cambiar el estado de un recurso o actualizarlo
 - **DELETE** Se usa DELETE para eliminar un recurso

Evolución

Web 1.0

Actualmente la web se ha convertido en algo cotidiano entre los mas de 600 millones de usuarios que componen internet, suponiendo un impacto en la economía mundial incalculable. Todo empezó en 1990 cuando Tim Berners-Lee creó lo que hoy concebimos como web 1.0, una versión inicial de la web que nada tiene que ver con lo que conocemos hoy en día.

Nace como un sistema de hipertexto para compartir información en internet, con la finalidad de publicar documentos. Cuando las empresas empiezan a darse cuenta del potencial de la web, empiezan a incorporar información corporativa con la finalidad de ser más próximos a los clientes y como un canal mas para promocionarse. Pero pronto se dan cuenta de las limitaciones que la web 1.0 contiene:

- El contenido publicado no se actualizaba constantemente por lo que podían pasar largos períodos de tiempo sin que esa información se modificase.
 - Las páginas eran estáticas y no permitían interacción de ningún tipo
 - La principal desventaja es que eran difícil de manejar y solo podían publicar contenido los más entendidos en el tema o web masters.

World Wide Web

The WorldWideWeb (W3) is a wide-area [hypermedia](#) information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an [executive summary](#) of the project, [Mailing lists](#), [Policy](#), November's [W3 news](#), [Frequently Asked Questions](#).

[What's out there?](#) Points to the world's online information, subjects, Web servers, etc.

Pointers to the world's online information, [subjects](#), [W3 servers](#), etc.
[Help](#)

on the browser you are using
Software Products

[Software Products](#) A list of W3 project components and their current status.

Technical Details of protocols, formats, program internals

Bibliography

Paper documentation on W3 and references.

People A list of some people involved in the project
History

History

How can I help?

If you would like to support the web..
[Getting code](#)

Figura 1.3: Ejemplo web 1º generación

Web 2.0

Debido a las limitaciones que ofrece la web 1.0, nace una nueva forma de concebir la web donde se valora las reacciones de los usuarios. Surgen aplicaciones y páginas que utilizan la inteligencia colectiva, consecuencia de ello las páginas pueden ser personalizadas convirtiéndose en una herramienta dinámica que permite el intercambio de información. Es por eso que la información se transforma en comunicación gracias a la interacción y a la incorporación de textos, vídeos, chats... Con esta nueva forma de concebir la web nacen los blogs, las redes sociales, los wikis... Este cambio ha supuesto una gran revolución, puesto que permite devolver la información de los usuarios y poder procesarla con el objetivo de controlar mejor la demanda.



Figura 1.4: Ejemplo web 2º generación

Web 3.0

La conocida web 3.0 dará paso a otro tipo de web donde pondrá su objetivo en la inteligencia artificial, un método para que los usuarios puedan no solo encontrar la información sino comprenderla. Este control está en manos de motores informáticos y procesadores de información, que tratan de analizar nuestro perfil y nuestra actividad en red para enviarnos información de nuestro interés. Es por esto que la web 3.0 es definida por el concepto "personalización", ya que pretende devolver al usuario una información lo más afinada posible, filtrada a sus gustos y preferencias, evitando información que no sea de su interés.

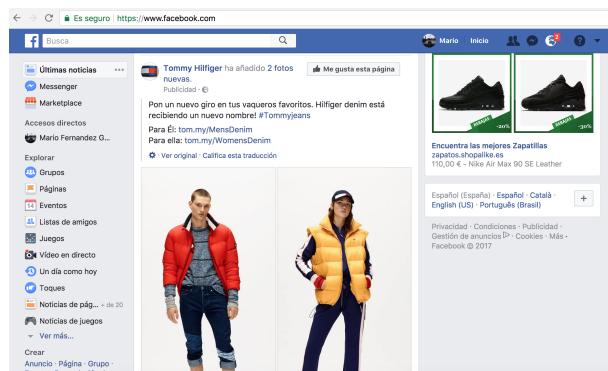


Figura 1.5: Ejemplo web 3º generación

Con la aparición de la web 3.0, nuevos términos tecnológicos aparecen como:

- **Web SPA:** Una web SPA es una aplicación de una sola página en la que la carga de datos es asíncrona y la página no se recarga en casi ningún momento, pese a cambiar de ruta o url para navegar entre las secciones de la aplicación, es una nueva tendencia en el desarrollo web.
- **Big Data:** Big data o macrodatos es un término que hace referencia a una cantidad de datos tal que supera la capacidad del software convencional para ser capturados, administrados y procesados en un tiempo razonable. El volumen de los datos masivos crece constantemente. En 2012 se estimaba su tamaño de entre una docena de terabytes hasta varios petabytes de datos en un único conjunto de datos.

Aplicaciones web

La aplicación de este TFG ha sido desarrollada basándose en modelos de aplicaciones que hoy en día están funcionando, como por ejemplo:

1.1.1. Airbnb

Es una empresa y una plataforma de software dedicada a la oferta de alojamientos a particulares y turísticos. El nombre es un acrónimo de airbed and breakfast (colchón inflable y desayuno). Airbnb tiene una oferta de unas 2.000.000 propiedades en 192 países y 33.000 ciudades. Desde su creación en noviembre de 2008 hasta junio de 2012 se realizaron 10 millones de reservas.

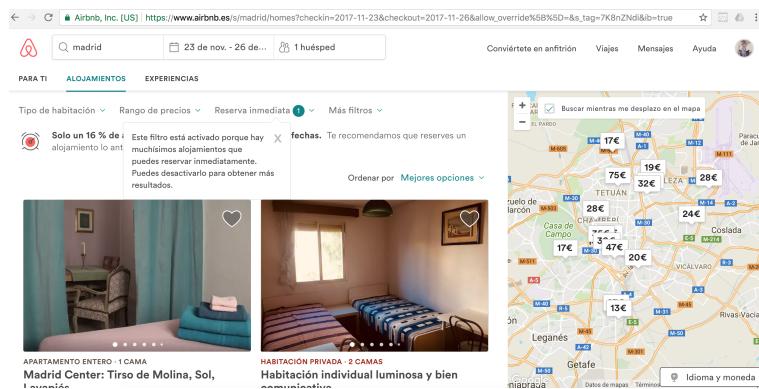


Figura 1.6: Aplicación Airbnb

1.1.2. Blablacar

Es un servicio de vehículo compartido que hace posible que las personas que quieren desplazarse al mismo lugar al mismo momento puedan organizarse para viajar juntos. Permite compartir los gastos puntuales del viaje (combustible y peajes) y también evitar la emisión extra de gases de efecto invernadero, al permitir una mayor eficiencia energética en el uso de cada vehículo.



Figura 1.7: Aplicación Blablacar

1.1.3. Wallapop

Es una empresa española fundada en 2013, que ofrece un website dedicado a la compra y venta de productos de segunda mano entre usuarios a través de Internet, con un uso centrado en smartphones. Utiliza la geolocalización para que los usuarios puedan comprar y vender en función de su proximidad geográfica



Figura 1.8: Aplicación Wallapop

1.2. Técnologia en el lado Cliente

El Front-end se desarrolla normalmente en HTML, CSS o Javascript, lo cual implica que los programadores se especialicen en estos tres lenguajes. Además de que el código sea correcto, la web debe tener un diseño atractivo y funcional, que permita que la experiencia del usuario sea lo suficientemente cómoda, intuitiva y agradable para que continúe navegando.

HTML 5

HTML5 es la quinta versión del lenguaje básico de la World Wide Web, publicado en Octubre de 2014. Al no ser reconocido en viejas versiones de navegadores por sus nuevas etiquetas, se recomienda al usuario común actualizar su navegador a la versión más reciente, para poder disfrutar de todo el potencial que provee HTML5.

HTML5 incluye significativas novedades en diversos áreas, ya que no incorpora solo nuevas etiquetas o elimina otras, sino que mejora áreas que estaban fuera del alcance del lenguaje:

- **Responsive:** Permite desarrollar aplicaciones que se adaptan fácilmente a distintas resoluciones, tamaños de pantallas, relaciones de aspectos y orientaciones.
- **Geolocalización:** Permite localizar gráficamente las páginas web por medio de una API de geolocalización.
- **Canvas:** Nuevo componente que permitirá dibujar en la página todo tipo de formas, que podrán estar animadas y responder a interacciones del usuario por medio de las funciones de un API.
- **WebSockets** tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP.
- **Aplicaciones web Offline** API que permite el trabajo con aplicaciones web, que se podrán desarrollar para que funcione también en local y sin estar conectados a internet.

ECMAScript 6

ECMAScript es una especificación estándar de un lenguaje desarrollado por Brendan Eich. Inicialmente se llamaba Mocha, luego LiveScript, y finalmente Javascript. Debido al gran éxito de Javascript como lenguaje de scripting del lado del cliente para páginas web, Microsoft desarrolló un dialecto compatible del lenguaje llamado JScript, para evitar problemas legales con la marca.

La primera versión de JavaScript, ECMAScript 1, se lanzó en Junio de 1997, y desde entonces han existido las versiones 2, 3 y 5, que es la más usada actualmente (la 4 se abandonó). Sobre la versión de ECMAScript 6, podemos decir que desde 2015 ya es un estándar cerrado, tratándose de una evolución del lenguaje JavaScript para dotarlo de características avanzadas que se echaban mucho en falta y que sí estaban disponibles en otros lenguajes populares, como por ejemplo:

- **Mejoras de sintaxis:** parámetros por defecto, variables let, plantillas...
- **Módulos para organización de código**
- **Verdaderas clases para programación orientada a objetos**
- **Promesas:** para programación asíncrona.

- **Mejoras en programación funcional:** expresiones lamda, iteradores, generadores...

Una cuestión muy importante es que ECMAScript 6 es totalmente compatible hacia atrás con versiones anteriores, por lo que no tenemos que preocuparnos por nuestro código actual, el cual funcionará perfectamente en motores de JavaScript que usen la próxima versión.

Centrando el foco en el lenguaje de programación javascript, aparecen multitud de framework que facilitan su programación como son Angular2+, Vuejs y React entre otros.

CSS3

CSS o Cascading Style Sheet (Hoja de estilos en cascada) es el lenguaje de diseño de la web. Su estandarización y especificación corre por parte del W3C, que lo incluyó a partir de la versión 4 de HTML.

Gracias a este lenguaje, podemos indicar donde se colocan los elementos, su color, apariencia, etc.

Al tratarse de estilos en cascada, los estilos que definamos en un elemento que contenga a otros, estos podrán propagarse hacia abajo. Por ejemplo: Si cambiamos el tamaño de fuente al elemento `<body>` (Padre de todo el documento), todos los párrafos y links de la página tendrán ese tamaño de fuente a menos que indiquemos lo contrario.

En esta última versión se han incluido grandes mejoras, podemos hacer uso de transformaciones 2D y 3D así como animaciones aceleradas por GPU, lo que hace que sean mucho más suaves y vistosas que programándolas como hasta ahora, en JavaScript.

Frameworks de javascript en el lado cliente

A continuación se van a describir los principales frameworks de JavaScript para el lado del cliente que existen en la actualidad.

- **Angular:** es el framework estrella hoy en día en demanda de ofertas de trabajo y en comunidad detrás de él. La gran crítica de los desarrolladores sobre Angular es su gran curva de aprendizaje, ya que empezar con Angular implica tener unos amplios conocimientos de diferentes tecnologías.
- **React:** es la segunda gran apuesta en el desarrollo de aplicaciones del lado de cliente. Ha sido creado por Facebook para el desarrollo de interfaces de usuario en aplicaciones Web. Constantemente se compara React con Angular pero sus objetivos son diferentes: React no es un framework sino una biblioteca que se centra en crear interfaces de usuario, a diferencia de Angular, que trata de abarcar mucho más.
- **Vue.js:** El tercer framework en esta lista es el proyecto de código abierto denominado Vue.js, el cual ha tenido una gran popularidad en los últimos tiempos

con la aparición de la versión 2.0 en 2016. Este framework trata de tomar lo mejor de cualquier framework e implementarlo, de hecho, en amplias comparativas entre diferentes frameworks ha conseguido unos resultados extraordinarios en velocidad y ligereza de peso

1.3. Técnologia en el lado Servidor

Un servidor web o servidor HTTP es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o Aplicación del lado del cliente. El código recibido por el cliente es renderizado por un navegador web.

La principal razón para usar servicios Web es que se pueden utilizar con HTTP sobre Transmission Control Protocol (TCP) en el puerto de red 80. Dado que las organizaciones protegen sus redes mediante firewalls (que filtran y bloquean gran parte del tráfico de Internet), cierran casi todos los puertos TCP salvo el 80, que es, precisamente, el que usan los navegadores web. Los servicios Web utilizan este puerto, por la simple razón de que no resultan bloqueados. Es importante señalar que los servicios web se pueden utilizar sobre cualquier protocolo, sin embargo, TCP es el más común.

Frameworks en el lado servidor

Los frameworks de lado servidor, es software que hacen más fácil escribir, mantener y escalar aplicaciones web.

- **Express:** es un framework web veloz, no dogmático, flexible y minimalista para Node.js. Proporciona un conjunto de características para aplicaciones web y móviles. Express es extremadamente popular, en parte porque facilita la migración de programadores web de JavaScript de lado cliente a desarrollo de lado servidor, y en parte porque es eficiente con los recursos
- **Django:** es un Framework Web Python de alto nivel que promueve el desarrollo rápido y limpio y el diseño pragmático. Es también veloz, seguro y muy escalable. Al estar basado en Python, el código de Django es fácil de leer y de mantener.
- **Ruby on rails:** es un framework web escrito para el lenguaje de programación Ruby. Rails sigue una filosofía de diseño muy similar a Django. Como Django proporciona mecanismos estándar para el enrutado de URLs, acceso a datos de bases, generación de plantillas y formateo de datos como JSON o XML.

1.4. Antecedentes

EL TFG centra su desarrollo en el campo de las tecnologías Web y como a través de ella permiten generar diversas aplicaciones con las que el usuario puedan interactuar. A continuación, se presentan varios ejemplos de TFGs dentro del Laboratorio de Robótica de la URJC que emplean estas tecnologías y han sido el punto de partida de este TFG.

1.4.1. Prácticas docentes de desarrollo web

Este TFG fué realizado por Walter Cuenca, con el objetivo de servir como apoyo en las prácticas de la asignatura de LTAW del grado de ingeniería de sistemas audiovisuales y multimedia. Consta de cuatro prácticas que utilizan diferentes tecnologías Web, herramientas y bibliotecas muy acentuadas en este campo. Algunas de las tecnologías empleadas son Javascript en el cliente, NodeJS y Django en el servidor, Base de datos como MySQL y por último Web-Socket y WebRTC como tecnologías de comunicación fluida.

<https://github.com/RoboticsURJC-students/2015-TFG-Walter-Cuenca>

1.4.2. Suveillance 5.1

Edgar barrero desarrollo en Ruby sobre Rails Suveillance 5.1, una aplicación web que ofrece un flujo de vídeo desde una cámara web, un flujo de imagen de profundidad procedente de un sector Kinect y su representación en 3D, además de un sensor de humedad y un actuador.

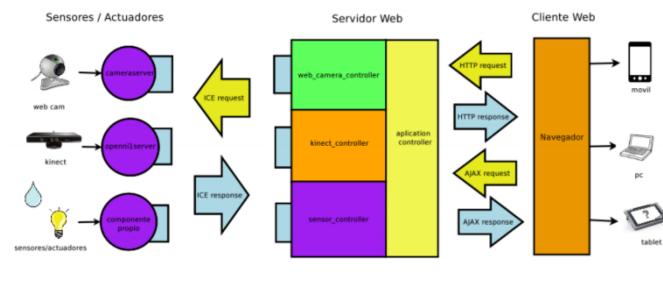


Figura 1.9: TFG Edgar Barrero

1.4.3. JdeRobotWebClients (URJC)

JdeRobotWebClients fue desarrollado por Aitor Martínez Fernández como trabajo fin de grado. La aplicación Web consiste en crear seis versiones web de herramientas utilizadas por JdeRobot (CameraViewJS, RGBDViewerJS, KobukiViewerJS,) que estaban programadas en C++ o Python con su propio interfaz gráfico provocando que sean ejecutables solo en Linux. Estas nuevas versiones son multiplataforma (Linux, Android, IOS, Windows) y accesibles desde un navegador web como interfaz gráfico permitiendo acceder a los sensores y actuadores sin un servidor intermedio.

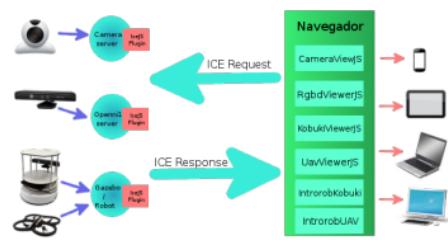


Figura 1.10: TFG Pablo Parejo

Capítulo 2

Objetivos

Una vez que hemos enfocado el contexto en el que se va a desarrollar este trabajo, pasamos a definir el objetivo general y los subobjetivos que se pretender cubrir en este TFG.

El objetivo principal es desarrollar Classcity, una aplicación web desarrollada con tecnología de última generación, cuya funcionalidad es facilitar el contacto entre alumnos y profesores para dar clases particulares. Classcity utiliza la geolocalización del alumno para proporcionarle los profesores más próximos a él, además de poder filtrar por diferentes parámetros como curso, asignatura y distancia.



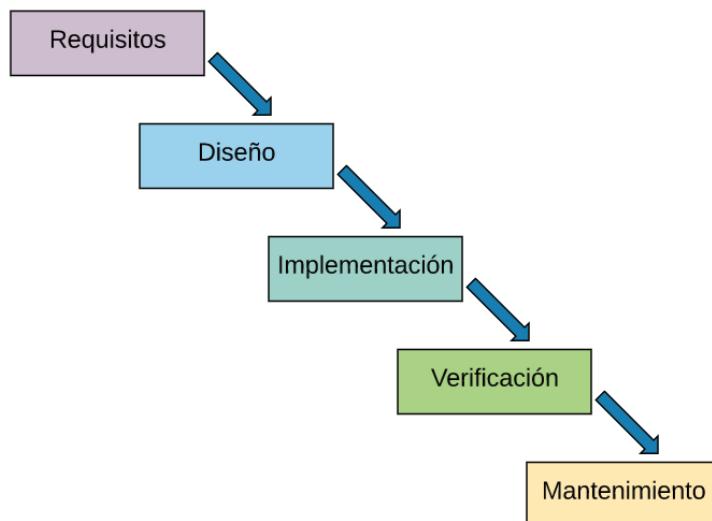
Para la realización de esta aplicación hemos decidido dividir nuestro objetivo en sub-objetivos mas sencillos con la finalidad de que quitemos complejidad al proyecto. Estos sub-objetivos son:

- **Front-End:** La parte del cliente es quizás la más tediosa por su dificultad a la hora de desarrollar una interfaz lo suficientemente ligera y adaptable para diferentes tipos de dispositivo. Es por esto que la elección del framework en el cliente nos puede cambiar por completo la estructura de nuestra aplicación web, además de reducir mucho los tiempos de desarrollo.
- **Back-End:** El segundo sub-objetivo es plantearnos cual de los diferentes frameworks del backend se adaptan mejor a nuestras necesidades. La elección de este framework viene condicionado en gran medida por el framework seleccionado para el cliente.

- **BBDD:** La base de datos por lo general pueden ser de dos tipos SQL y NoSQL, es por esto que debemos elegir cual de los dos tipos de bases de datos satisface mas con nuestras necesidades.
- **Despliegue en la nube:** Una vez que tengamos nuestra aplicación funcionando correctamente en local, es momento de desplegarlo en alguno de los cloud que nos ofrecen lo proveedores mas importantes como son: AWS, Azure o GoogleCloud

2.1. Metodología

En la realización del proyecto se ha necesitado definir una metodología que permita planificar las tareas necesarias para llegar a nuestro objetivo. El modelo seleccionado para la realización del TFG ha sido de tipo cascada, un proceso de desarrollo secuencial, en el que el desarrollo de software se concibe como un conjunto de etapas que se ejecutan una tras otra. Se le denomina así por las posiciones que ocupan las diferentes fases que componen el proyecto colocadas una encima de otra, y siguiendo un flujo de ejecución de arriba hacia abajo, como una cascada.



Como parte de la metodología, durante el tiempo que ha durado el proyecto se acordaron reuniones semanales con el tutor de forma presenciales o por Vídeo-Conferencia en las que se revisaba los objetivos semanales y se definían los nuevos hitos.

2.2. Plan de trabajo

Para la realización de todo el proyecto he seguido una metodología de trabajo que ha consistido en cinco diferentes fases:

- **Primera fase:** Es una fase de iniciación cuyo objetivo principal es el de aprender todo lo que tenga que ver con el desarrollo web. En esta fase deberíamos de dejar

conceptos básicos aclarados y empezar a manejar alguna herramienta de control de versiones como Git. Es muy recomendable en esta primera fase empezar a manejar los lenguajes de programación que quieras utilizar en el futuro.

- **Segunda fase:** Una vez que tenemos cierta destreza con el desarrollo, empezamos a enfocar nuestra aplicación decidiendo que tecnologías son las que mejor nos van a venir para nuestro modelo de aplicación. Esta fase es vital para la continuación del proyecto, ya que la mala elección de una tecnología nos puede llevar mucho tiempo.
- **Tercera fase:** Una vez que tenemos claro que tecnologías vamos a utilizar en nuestra aplicación, comenzamos con una sencilla aplicación que utilice todas las tecnologías que estarán implicadas en nuestra aplicación. Esto nos servirá para tener una sencilla estructura de lo que queremos montar.
- **Cuarta fase:** Cuando tengamos claros los conceptos, manejemos los lenguajes de programación necesarios y tengamos montado una sencilla aplicación con las tecnologías que hemos seleccionado para nuestro proyecto, es momento de empezar a dar forma a nuestras ideas. Esta fase es quizás la más emocionante de todas ya que empiezas a dar cuerpo a lo aprendido hasta ahora.
- **Quinta fase:** Cuando tengamos nuestra aplicación completamente desarrollada y haciendo lo que nosotros queremos, es el momento de subirla a alguna plataforma de computación en la nube. Esta ultima fase es quizás la mas sencilla y mas gratificante del proceso ya que es el momento de que tu trabajo sea contemplado por el resto del mundo.

Capítulo 3

Infraestructura

Una vez introducidos en el proyecto y con los objetivos marcados se hablará de las tecnologías usadas para el desarrollo de la aplicación. Las cuatro principales tecnologías en las que gira el proyecto son: Mongodb, Express, Angular2 y Node.

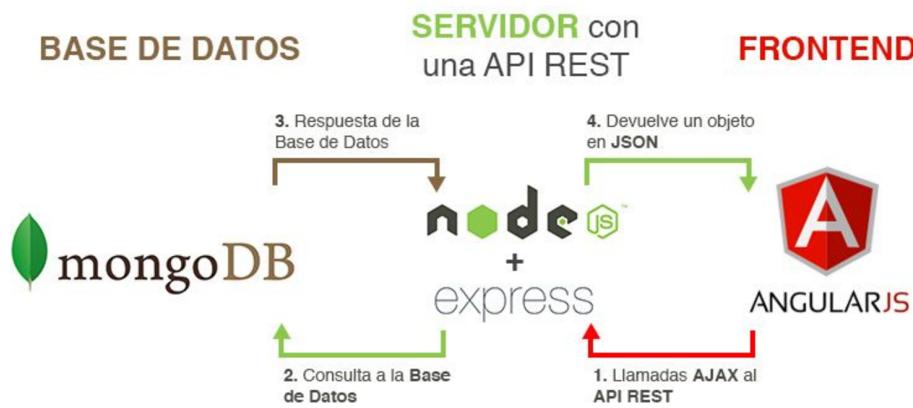
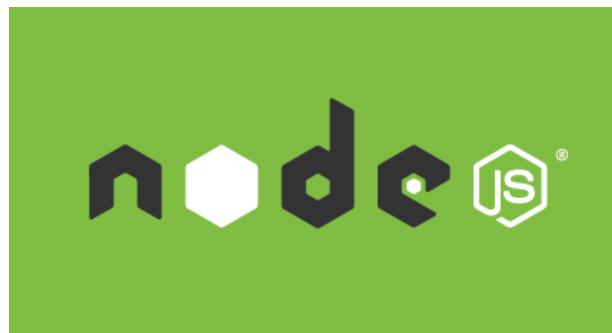


Figura 3.1: Esquema MEAN

En este esquema podemos ver como se comporta el stack MEAN, el cual hemos utilizado para el desarrollo de nuestra app. En este capítulo vamos a explicar como se comporta este paquete de 4 tecnologías:

3.1. Node



Node es un entorno de programación en JavaScript para el Backend basado en el motor V8 del navegador Google Chrome y orientado a eventos, no bloqueante, lo que lo hace muy rápido a la hora de crear servidores web y emplear tiempo real. Fue creado en 2009 y aunque aún es joven, las últimas versiones lo hacen más robusto además de la gran comunidad de desarrolladores que posee.

Uno de los beneficios de Node es su gestor de paquetes, npm (node package manager), el cual nos permite gestionar todas las dependencias y módulos de una aplicación. Al igual que Ruby tiene RubyGems y PHP tiene Composer, Node tiene npm. Viene ya incluido con Node y permite que nos bajemos una serie de paquetes para satisfacer nuestras necesidades.

Este sistema de paquetes es lo que hace a Node tan potente. La capacidad de tener una serie de códigos que puedes reutilizar en todos tus proyectos hace que el desarrollo sea mucho más sencillo, ya que puedes combinar varios paquetes para crear aplicaciones complejas.

```
> npm install && npm start
```

Con esta instrucción en linea de comandos, nos descargamos todas las dependencias contenidas en nuestro packcage.json, además de arrancar nuestra aplicación.

3.2. MongoDB



3.2.1. Introducción

MongoDB es la base de datos que he elegido para mi aplicación, debido a sus grandes ventajas cuando se manejan ingentes cantidades de información. MongoDB nace en octubre de 2009 y a día de hoy innumerables empresas ya disponen de esta base de datos en sus aplicaciones como por ejemplo:

- **Bosh:** Utiliza MongoDB ya que esta poniendo a prueba una aplicación que es capaz de capturar datos de vehículos, como el sistema de frenado, la dirección asistida, los limpiaparabrisas ... Con todos estos datos se pueden hacer diagnósticos de necesidad de mantenimiento preventivo.
- **Forbes:** Construyo todo un sistema de gestión de contenidos en MongoDB. Además utiliza MongoDB para analítica en tiempo real. Cuando algún artículo se hace viral, Forbes detecta la forma en que se está compartiendo entre los usuarios y de este modo sabe qué tipo de contenido le debe ofrecer a sus lectores.

3.2.2. Características

MongoDB es una base de datos no relacional (NoSQL) de código abierto que guarda los datos en documentos tipo JSON (JavaScript Object Notation) pero en forma binaria (BSON) para hacer la integración de una manera más rápida. Se pueden ejecutar operaciones en JavaScript en su consola en lugar de consultas SQL. Además tiene una gran integración con Node.js con los driver propio y con Mongoose, framework que explicaremos más adelante. Debido a su flexibilidad es muy escalable y ayuda al desarrollo ágil de proyectos web.

MongoDB esta orientado para servicios que necesiten una persistencia basada en documentos, al contrario que otros sistemas de base de datos noSQL como Cassandra, el cual esta orientado para logs, o como Redis que necesita una persistencia basada en colas de mensajes.

Estamos ante la era de lo que Martin Fowler llama “Polyglot persistence”. Hay que decidir el tipo de persistencia a utilizar para después usar el tipo de persistencia que más se amolde a nuestras necesidades.

Las características que hacen tan importante a esta base de datos son las siguientes:

- Está orientada a documentos. Lo que quiere decir que en un único documento es capaz de almacenar toda la información necesaria que define un producto, un cliente, etc, aceptando todo tipo de datos sin tener que seguir un esquema predefinido.
- Da respuesta a la necesidad de almacenamiento de todo tipo de datos: estructurados, semi estructurados y no estructurados.
- Tiene un gran rendimiento en cuanto a escalabilidad y procesado de la información.
- Da respuesta a la necesidad de almacenamiento de todo tipo de datos: estructurados, semi estructurados y no estructurados.

- Puede procesar la gran cantidad de información que se genera hoy en día..
- Permite a las empresas ser más ágiles y crecer más rápidamente, creando así nuevos tipos de aplicaciones.

3.2.3. Documento en MongoDB

MongoDB esta escrito en C++, su versión de 32 bits solo puede alcanzar 2GB, por este motivo la versión de 32 bits no es recomendable usarla en producción.

```
{
  name: "mario",
  age: 25,
  preferences: [
    "programming",
    "nosql",
    "javascript"
  ]
}
```

Esto es un documento en Mongo, los cuales se almacenan en colecciones y estas a su vez en bases de datos. Estas colecciones poseen un esquema flexible y totalmente dinámico lo que hace que la velocidad de computo sea muy alta. Las bases de datos no se crean manualmente, primero se define la base de datos a usar y luego se inserta un documento en alguna colección.

```
> show dbs
> use pruebanosql
> show collections
> db.users.insert({ "name": "mario", "age": 24})
```

3.2.4. Inconvenientes de MongoDB

Un problema que tiene Mongo, es que no soporta transacciones de múltiples documentos, sin embargo puede proporcionar operaciones atómicas en un solo documento. A menudo, estas operaciones atómicas de nivel de documento son suficientes para resolver los problemas que requerían transacciones en una base de datos relacional. Por ejemplo en Mongo se pueden incrustar datos relacionados en matrices anidadas o documentos anidados dentro de un solo documento y actualizar todo el documento en una sola operación atómica. Por este motivo los servicios que requieren de transacciones como los bancos o entidades económicas, no utilizan Mongo debido a que es sensible a Hacker, debido a que no es capaz de hacer una sola operación atómica en dos documentos.

Otro posible problema podría ser la excesiva cantidad de memoria RAM que puede consumir MongoDB, aunque es posible ejecutar MongoDB en una maquina con una pequeña cantidad de memoria RAM libre. Pero si es cierto que MongoDB usa automáticamente toda la memoria libre del equipo como su cache, es por esto por lo que los monitores de recursos muestran que MongoDB utiliza una gran cantidad de memoria, pero su uso es dinámico. Es decir que si otro proceso de repente necesita

mayor espacio de memoria RAM, MongoDB liberara parte de su memoria asignada para el otro proceso.

3.2.5. Fragmentación (Sharding)

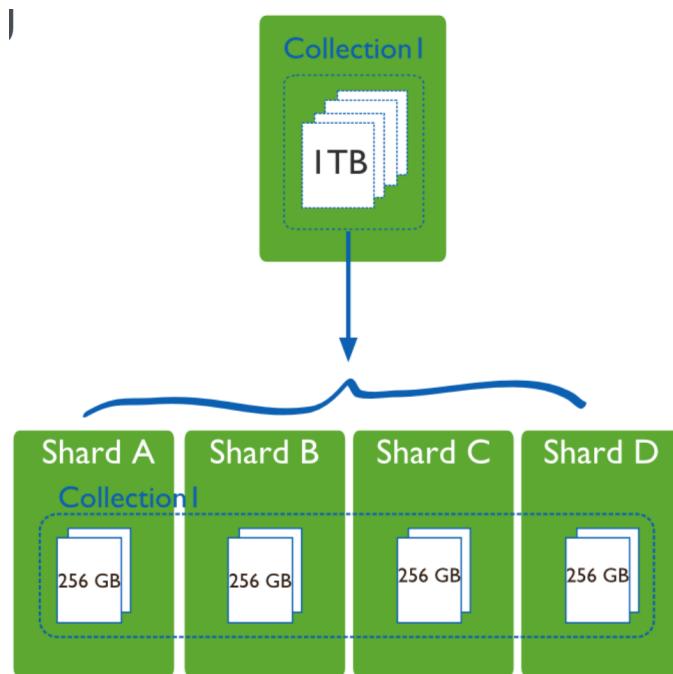


Figura 3.2: Sharding

¿Que es el Sharding? Cuando el proyecto que estas llevando a cabo empieza a tener un numero de peticiones de acceso elevado , empiezas a notar que tu base de datos va mas lento de lo normal. Para este problema tienes dos soluciones, una actualizar toda la infraestructura para soportar la demanda o empezar a utilizar el sharding.

El sharding, es el modo en el que hacemos nuestra base de datos escalable. En lugar de tener una colección en una base de datos, la tendríamos en varias bases de datos distribuidas, de modo que a la hora de consultar los datos de dicha colección, los recuperaremos como si de una única base de datos se tratase. Todo esto de encontrar la base de datos lo hace MongoDB de forma transparente. Cuando hacemos consultas, tenemos un enrutador llamado "MongoS", el cual mantendrá un pequeño pull de conexiones a los distintos host.

Los fragmentos estarán formados por replica set, de modo que si creamos tres fragmentos, cada uno de los cuales tiene una replica set con tres servidores, estaríamos hablando de un total de nueve servidores. Para saber en que fragmento debe consultar para recuperar datos de una colección ordenada, se utilizan rangos y shard key, de modo que se trocea la colección en rangos y se les asigna un id a cada rango, de este modo que cuando se consulte la colección debemos proporcionar el shardKey.

3.3. Express

express

3.3.1. Introducción

Express es un framework de aplicaciones web para Node.js, que permite crear servidores web y recibir peticiones HTTP de una manera sencilla, lo que permite también crear APIs REST de forma rápida.

En la web de ExpressJS, lo describen como “un framework de desarrollo de aplicaciones web minimalista y flexible para Node.js”. Sin duda el éxito de Express radica en lo sencillo que es usarlo, y además abarca un sin número de aspectos que muchos desconocen pero son necesarios.

La referencia de la API se divide en 5 grandes módulos:

- **express():** La función express () es una función de nivel superior exportada por el módulo express.

```
express.json()  
express.static()  
express.Router()  
express.urlencoded()
```

- **Application:** El objeto app se crea llamando a la función express () de nivel superior exportada por el módulo Express

```
Properties -->| app.locals || app.mountpath |  
Events -->| mount |  
Methods -->| app.all() | app.delete() | app.disable() | app.listen() |
```

- **Request:** El objeto req representa la solicitud HTTP y tiene propiedades para la cadena de consulta de solicitud, parámetros, cuerpo, encabezados HTTP, etc.

```
Properties -->| req.body | req.cookies |  
Methods -->| req.accepts() | req.acceptsCharsets() |
```

- **Response:** El objeto res representa la respuesta HTTP que envía una aplicación Express cuando recibe una solicitud HTTP.

```
Properties -->| res.app | res.headersSent | res.locals |  
Methods -->| res.cookie() | res.clearCookie() |
```

- **Router:** El objeto Router es una instancia aislada de middleware y rutas. Puede considerarlo como una ”miniacplación”, que solo puede realizar funciones de enrutamiento y middleware. Cada aplicación Express tiene un router de aplicaciones integrado.

```
Methods -->|router.all()|router.METHOD()|router.param()|
```

3.3.2. Estructura de una app.js

En una aplicación escrita con express existe una estructura interna bien definida y es como sigue:

- **Módulos o archivos externos** Importamos todos los módulos o archivos externos que nuestra app vaya a necesitar. El bloque, o mejor dicho la línea, que viene a continuación es la más importante de todas, ya que se encarga de instanciar Express y asignarlo a la variable app, la cual se utilizará a partir de ahora para configurar los parámetros de Express.

```
var app = express();
```

El siguiente bloque sirve para configurar e iniciar algunos componentes de Express. Destacar la línea, app.use(express.static(...)), en la cual se configuran los objetos estáticos (imágenes, hojas de estilo, etc.) que debe servir Express, los cuales se encuentran en la carpeta public. Esta configuración permite también que los elementos estáticos puedan ser accedidos como si se encontraran en el directorio raíz del proyecto, de forma que para acceder a las imágenes ubicadas en /public/images se haría con la URL <http://localhost:3000/images>.

```
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));
```

- **Conectamos a la base de datos** La segunda parte de nuestra app express, es conectarnos a la base datos.
- **Importamos controladores y modelos de la base de datos** Una vez conectados a la base de datos importamos los controladores y los modelos de nuestra base de datos.
- **Rutas** Las rutas son definitivamente la parte más importante de tu aplicación, ya que son las encargadas de invocar las funciones que se encuentran en el controlador.

Como podemos ver una ruta esta especificada de la siguiente forma:

```
app.VERBO(PATH, ACCION)
```

VERBO: Puede ser: GET, POST, PUT, DELETE y así para cada uno de los verbos HTTP.

PATH: Define la dirección de acceso.

ACCION: Que es lo que se tiene que hacer.

- **Listen** Por último es importante que tu aplicación este disponible en algún puerto.

```
app.listen(8000);
```

Express esconde muchas funcionalidades internas de Node, lo que te permite sumergirte en el código de tu aplicación y conseguir tus objetivos de forma muy rápida. Es fácil de aprender y te deja cierta flexibilidad con su estructura. Por algo es el framework más popular de Node.

3.4. Angular



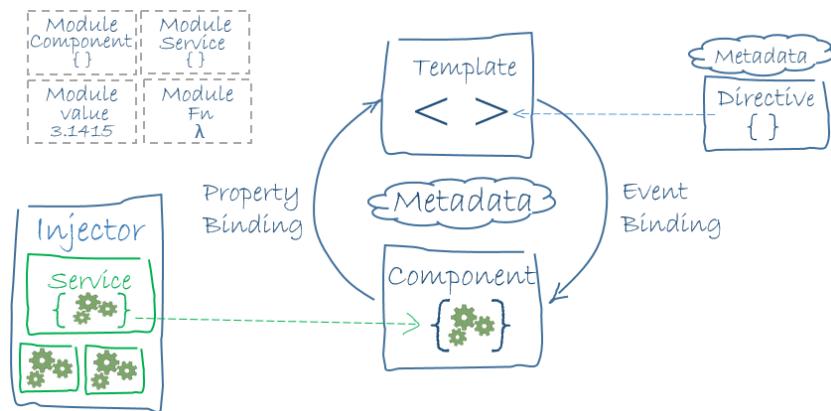
Angular es un framework JS para la parte cliente o Frontend de una aplicación web, que respeta el paradigma MVC y permite crear Single-Page Applications (Aplicaciones web que no necesitan recargar la página), de manera más o menos sencilla. Es un proyecto mantenido por Google y que actualmente está muy en auge.

Angular es un framework completo para construir aplicaciones en cliente con HTML y Typescript, es decir, con el objetivo de que el peso de la lógica y el renderizado lo lleve el propio navegador, en lugar del servidor.

Para crear apps en Angular 2 necesitamos:

- **Plantillas HTML (templates)**
- **Componentes para gestionar esas plantillas**
- **Servicios para gestionar la lógica de la aplicación**
- **El componente raíz de la app al sistema de arranque de Angular 2 (bootstrap).**

Veamos como se relacionan estos elementos en el diagrama de arquitectura típico, sacado de la web de Angular 2:



Podemos identificar los 8 bloques principales de una app con Angular 2:

- **Módulo** Igual que con las versiones anteriores de Angular, las aplicaciones de Angular en su versión más actualizada son modulares. Un módulo, típicamente es un conjunto de código dedicado a cumplir un único objetivo. El módulo exporta algo representativo de ese código, típicamente una única cosa como una clase. Los módulos se pueden exportar e importar:

```
//app/app.component.js
  export class AppComponent {
    //aqui va la definicion del componente
  }

//app/main.js
  import { AppComponent } from './app.component';
```

Hay módulos que son librerías de conjuntos de módulos. Las librerías principales de Angular son:

- @angular/core
 - @angular/common
 - @angular/router
 - @angular/http
- **Componente** Un Componente controla una zona de espacio de la pantalla que podríamos denominar vista. El componente define propiedades y métodos que están disponibles en su template, pero eso no te da licencia para meter ahí todo lo que te parezca. Haciendo un símil con AngularJS (Angular en su primera versión), un componente vendría a ser un controlador que siempre va ligado a una vista.

- **Template** El Template (cuyo concepto ya existía en AngularJS), es lo que nos permite definir la vista de un Componente. Igual que su predecesor, el template de Angular es HTML, pero decorado con otros componentes y algunas directivas: expresiones de Angular que enriquecen el comportamiento del template.

Como vemos, además de elementos HTML normales como <h2> y <div>, hay otros elementos desconocidos en nuestro lenguaje de markup:

- *ngFor
- todo.subject
- (click)
- [todo]
- todo-detail

- **Metadatos:** La forma de añadir metadatos a nuestra clase en TypeScript es mediante el patrón decorador justo antes de la declaración de la clase. Veamos:

```
import { Component } from '@angular/core';

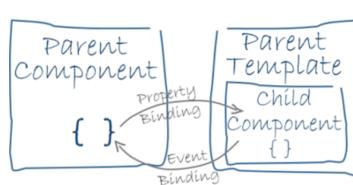
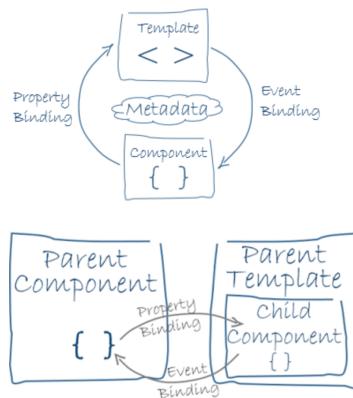
@Component({
  selector:      'todo-list',
  templateUrl:  'todo-list.component.html',
  styleUrls:    ['todo-list.component.css'],
  moduleId:     module.id,
  directives:   [TodoDetailComponent],
  providers:    [TodoService]
})
export class TodoListComponent { ... }
```

- **Data Binding** Uno de los principales valores de Angular es que nos abstrae de la lógica pull/push asociada a insertar y actualizar valores en el HTML y convertir las respuestas de usuario (inputs, clicks, etc) en acciones concretas. Escribir toda esa lógica a mano (lo que típicamente se hacía con JQuery) es tedioso y propenso a errores, y Angular 2 lo resuelve por nosotros gracias al Data Binding.

- **Interpolación** Hacia el DOM. todo.subject
- **Property binding** Hacia el DOM. [todo] = "selectedTodo"
- **Event binding** Desde el DOM. (click) = "selectTodo(todo)"
- **Two-way binding** (Desde/Hacia el DOM) input [(ngModel)] = "todo.subject"

Angular procesa los data binding una vez por cada ciclo de eventos JavaScript, desde la raíz de la aplicación siguiendo el arbol de componentes en orden de profundidad.

Los siguientes gráficos de la documentación de Angular ilustran la importancia del data-binding para la comunicación entre componentes, así como componente-template.



- **Directiva:** Los templates de Angular son dinámicos: Cuando Angular los renderiza, transforma el DOM en base a las instrucciones que encuentra en las directivas

Cuando hemos hablado de los componentes y hemos dicho que eran similares a un controlador con una vista, muchos habréis pensado “más bien se parece a una directiva...”. Es cierto, un Componente es un caso concreto de directiva que siempre va asociado a un template y al que por ser un elemento tan importante en Angular 2 se le ha dado un decorador propio.

Tenemos dos tipos de directivas:

- **Las directivas estructurales** comienzan por asterisco y sirven para alterar el DOM.

```
<div *ngFor="let todo of todos"></div>
<todo-detail *ngIf="selectedTodo"></todo-detail>
```

- **Las directivas Atributo** alteran la apariencia o comportamiento de un elemento del DOM

```
<input [(ngModel)]="todo.subject" >
```

- **Servicio:** Los servicios son imprescindibles en Angular, si bien en Angular se definen a través de simples clases. Todo valor, función o característica que nuestra aplicación necesita, se encapsula dentro de un servicio.

Los Componentes son grandes consumidores de servicios. No recuperan datos del servidor, ni validan inputs de usuario, ni logean nada directamente en consola. Delegan todo este tipo de tareas a los Servicios.

- **Dependency Injection** Una dependencia en tu código se produce cuando un objeto depende de otro. Hay diferentes grados de dependencia, pero tenerla en exceso hace que testear tu código sea complicado o que algunos procesos se ejecuten más tiempo de la cuenta. La inyección de dependencias es un método por el cual damos a un objeto las dependencias que requiere para su funcionamiento.

Angular permite extender el vocabulario de tu HTML con directivas y atributos para crear componentes dinámicos. Si alguna vez has hecho una página web dinámica sin Angular te habrás dado cuenta de ciertas complicaciones frecuentes, como el data binding, validación de formulario, manejador de eventos con DOM (Document Object Model) y otras muchas. Angular presenta una solución “todo-en-uno” a esos problemas. La curva de aprendizaje para Angular es muy pequeña, lo que explica que mucha gente se este pasando a este framework. La sintaxis es simple y sus principios básicos como el data binding (vinculación de elementos de nuestro documento HTML con nuestro modelo de datos) y la inyección de dependencias son sencillas de entender.

3.5. Cuándo usar el stack MEAN

Las ventajas del stack MEAN provienen de la robustez de Node. Node nos proporciona su API abierta en real-time (tiempo real) la cual podemos usar libremente con nuestro código frontend en Angular. Podemos usarlo para transferir datos para aplicaciones como chats, actualización de estados, o cualquier otra situación que requiera mostrar datos rápidamente en tiempo real:

1. Chat
2. Actualización de estados en tiempo real por el usuario
3. Tienda online
4. Polling app (aplicación para votaciones)

Capítulo 4

Diseño e implementación

4.1. Introducción

En esta sección describiremos mas en profundidad el diseño y la implementación realizados en el proyecto. Antes, comenzaremos con una breve introducción al mismo para tener una visión más global y poder entender más adelante las partes por separado.

4.2. Estructura básica

Como he explicado en el capítulo anterior la aplicación que hemos desarrollado se divide en 4 grandes bloques: Node, MongoDB, Express y Angular. Cada uno de ellos se encarga de realizar una función dentro de la aplicación.

Antes de profundizar en cada bloque, todos los proyectos que utilizan el stack MEAN, siguen una estructura similar a la siguiente.



Figura 4.1: Estructura proyecto MEAN

4.3. Node

De los 4 grandes bloques he decidido empezar con Node, ya que es imprescindible para poder construir el proyecto.

Como ya he comentado en el capítulo anterior, Node es un sistema innovador, puesto que es la plataforma encargada del funcionamiento del servidor, y funciona totalmente con JavaScript, como bien es sabido Javascript es un lenguaje de programación que en un principio era dedicado a correr en los navegadores, o en el lado del cliente, pero como se puede ver con el MEAN stack y sus subsistemas el uso de JavaScript se ha ampliado considerablemente a todos los aspectos de una página web.

Cabe destacar, sin duda, la irrupción en los últimos años de servidores interactivos asíncronos, como es el caso de NodeJS. Este tipo de servidores realizan tareas de forma asíncrona bloqueante, por lo que el tiempo de respuesta es, en muchos casos, menor que en servidores multihilos(servidores concurrentes). Por estos motivos, en la actualidad, múltiples proyectos enfocados al tiempo real, utilizan este tipo de servidores asíncronos.

De la mano a Node tenemos NPM, es el manejador de paquetes por defecto para Node.js. Para poder arrancar cualquier proyecto en MEAN, debemos utilizar npm para bajarnos todas las dependencias del proyecto.

Una vez tengamos Node y NPM instalados, lo utilizaremos en los siguientes casos:

- **Instalar dependencias(Cliente y Servidor)**

```
sudo npm install
```

- **Correr nuestro servidor**

```
sudo node server.js
```

- **Arrancar el cliente:**

```
sudo npm start
```

4.4. Back-End: Express MongoDB

Nuestro lado servidor se compone de dos tecnologías muy importantes e innovadoras en el mercado de las aplicaciones web como son: Express, que como ya hemos comentado antes es un framework de desarrollo de aplicaciones web para Node.js y MongoDB que consiste en una base de datos NoSQL,el cual utiliza la librería mongoose para poder conectar node.js con la base de datos en MongoDB.

Comenzaré explicando el backend con el primer fichero que ejecutamos cuando queremos arrancar nuestro servidor Server.js

4.4.1. Server.js

Para poder empezar analizando el código implementado en nuestro server.js, tenemos que saber como arrancar nuestro servidor.

```
sudo node server.js
```

Gracias a Node, simplemente con una linea en la terminal seremos capaces de generar procesos que escuchen en el puerto que queramos.

A continuación vamos a enumerar todas las funciones contenidas en nuestra server.js

- 1. Open mongo** Llamamos a la librería mongoose encargada de unir a MongoDB con Node.js, a continuación le decimos a que data base apuntar y si el resultado es satisfactorio abrirá la conexión en caso contrario saltará la excepción.

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/classcity');
var db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function() {
  console.log('Connected to Database');
});
```

- 2. Parser body** Analizamos los body de las request en un middleware antes que llegue a sus manejadores.

```
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
```

- 3. control errores** En caso de tener algún error en alguna solicitud, el manejador de errores se lanzará sin bloquear el resto del servicio.

```
app.use(function(err, req, res, next) {
  if (err.name === 'StatusError') {
    res.send(err.status, err.message);
  } else {
    next(err);
  }
});
```

- 4. control imágenes** Para poder controlar la ingestión de imágenes en nuestras bases de datos, hemos usado Multer, un middleware de node.js para el manejo multipart/form-data, que se usa principalmente para cargar archivos.

```
var storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, './uploads')
  },
  filename: function (req, file, cb) {
    console.log(file.fieldname);
    var name = file.fieldname + '-' + Date.now() + '.jpg';
    cb(null, name)
  }
})
var upload = multer({ storage: storage });
app.use(multer(upload).single('file'));
```

5. **Server sockets** Corremos nuestro chat en el puerto 8080, independientemente de nuestra aplicación que corre en el puerto 8000

```
var socketServer = require('../controllers/socket');
socketServer.start();
```

6. **Rutas** Nuestra aplicación se compone de multitud de rutas que invocan a funciones contenidas en nuestro controlador que mas tarde explicaremos.

```
//Rutas
app.use('/', users);
app.use("/uploads", express.static(__dirname + '/uploads'), img);
img.route('/:id').get(Ctrlprofesor.getimg)
users.route('/loginalumno').post(Ctrlalumno.loginalumno);
```

7. **Start server** Arrancamos nuestra aplicación en el puerto 8000

```
app.listen(8080, '0.0.0.0', function() {
  console.log("Node server running on http://localhost:8080");
});
```

4.4.2. Estructura de la base de datos

Como bien sabemos MongoDB es una base datos no relacional, es decir no es como las típicas bases de datos SQL donde existen relaciones entre una tabla y otra.

La estructura de la base de datos que he elaborado consiste en 4 modelos, los cuales se relacionan dos a dos por medio de referencias y el metodo populate en MongoDB.

Analizamos la estructura de los modelos:

- **Modelo Alumnos** Consiste en un modelo simple donde tenemos tres campos predefinidos de tipo string.

```
var alumnoSchema = new Schema({
  nombre: { type: String },
  apellidos: { type: String },
  edad: { type: String }
});
module.exports = mongoose.model('Alumno', alumnoSchema);
```

- **Modelo Login Alumnos** Este modelo encapsula dentro de él al anterior, y lo hace a partir de una llamada de referencia. Dentro del campo data tendremos el modelo del alumno.

```
var loginSchema = new Schema({
  email: { type: String },
  password: { type: String },
  data: { type: Schema.ObjectId, ref: "Alumno" },
});
module.exports = mongoose.model('LoginAlumno', loginSchema);
```

- **Modelo Profesores** Este modelo corresponde al del profesor.

```
var profesorSchema = new Schema({
  nombre: { type: String },
  apellidos: { type: String },
  edad: { type: String },
  curso: { type: String, enum:
    ['Primaria', 'ESO', 'Bachillerato', 'Universidad', 'FP',
    'EXAMENES LIBRES', 'FRACASO ESCOLAR'] },
  asignaturas: { type: String },
  location: {
    type: { type: String },
    coordinates: {type: []}
  },
  path: {type: String},
  notification: [
    type: [
      alumno: { type: Schema.ObjectId, ref: "Alumno" },
      leido: {type: Boolean},
      _id: false
    ]
  ]
});
});
```

- **Modelo Login Profesores** Modelo que vuelve a anidar otro modelo en el campo data.

```
var loginSchema = new Schema({
  email: { type: String },
  password: { type: String },
  data: { type: Schema.ObjectId, ref: "Profesor" },
});
module.exports = mongoose.model('LoginProfesor', loginSchema);
```

La idea de tener estos modelos relacionados, es porque puede no interesarnos enviar toda la información en una llamada, es decir si un usuario hace introduce su email y su password en la ventana de login, no necesitaríamos buscar entre todos los datos de los usuarios, simplemente con tener un modelo con el login y la password de los usuario para poder hacer la verificación sería más que correcto.

4.4.3. Controladores

Un controlador es un archivo donde tenemos diversas funciones que son invocadas a partir de las rutas que tenemos configuradas en el server.js. Dependiendo del modelo de la base de datos que utilicemos para guardar, editar o eliminar datos, he decidido organizar las funciones en tres controladores diferentes:

- **Controllers Alumnos** Es el fichero en el que están todas las funciones que usan los modelos alumno.js y loginalumno.js

1. **registeralumno:** Función cuyo comportamiento consiste en comprobar que el email que introduce el alumno al registrarse no esta en nuestra base de datos, y que los campos password y email no están vacíos, en tales casos el servidor devolverá un 400 al cliente.

Si el alumno es registrado con éxito, se guardará en la base de datos y se enviarán las credenciales con un 201 en forma de token para una mayor seguridad.

```
alumno.save(function(err, datasave) {
  if(err) return res.send(500, err.message);
  var profile = _.pick(req.body, 'Email', 'Password', 'extra');
  profile.id = datasave.data;
  res.status(201).send({ id_token: createToken(profile) });
});
```

2. **loginalumno:** Si el alumno ya ha sido registrado en nuestra base de datos, y lo que quiero en hacer loguin, esta función sera invocada y comprobará que el email y la password del alumno coinciden con los credenciales, en caso de ser aceptado se le enviará sus credenciales en forma de token con un 201 y en caso de ser rechazado se le enviara un 400 con el mensaje: "The username or password don't match".

- **Controllers Profesores**Es el fichero en el que estan todas las funciones que usan los modelos profesor.js y loginprofesor.js

1. **registerprofesor:** El comportamiento es idéntico a registeralumnos, con la única salvedad de que el modelo que utilizamos es profesor.js
2. **loginprofesor:** También mismo comportamiento que en loginalumnos, pero utilizando el modelo de datos de loginprofesor.js
3. **savenotificacion:** Cuando un alumno quiere contactar con un profesor vía chat, antes tiene que enviarle una petición de contacto. De esto consiste savenotification, es una función que se encarga de almacenar en la base de datos del profesor los alumnos que le han enviado una petición de contacto.

```
exports.savenotificacion = function(req, res){
  DataProfesor.findOneAndUpdate(
    {_id: req.body._id},
    {$addToSet: {notification: {alumno: req.body.id,
      leido: false, _id: false}}},
    {safe: true},
    function(err, model) {
      if (err == null){
        res.status(200).send("La notificacion ha
          sido recibida");
      }else{
        res.send(500, err.message);
      }
    }
);
```

```
};
```

4. **readynotificacion:** Es una función que se encarga de comprobar si el profesor a aceptado o rechazado la solicitud de contacto del alumno. En caso de ser aceptado, el chat se habilitará y el alumno y el profesor podrán tener un primer contacto.
5. **getallprofesores:** Función que se encarga de enviar al cliente todos y cada uno de los profesores que integran Classcity sin ningún tipo de requisito.
6. **getimg:** Como podemos ver en el código, es una función muy simple que se encarga de enviar al cliente la imagen que solicita.

```
exports.getimg = function(req, res){
    res.sendFile('uploads/' + req.params.id)
};
```

7. **postimg:** Función que se encarga de actualizar las imágenes de los profesores que editan su perfil.

```
exports.postimg = function(req, res){
    ProfesorScheme.findOne({ "email" : req.file.originalname },
        function(err, data) {
            DataProfesor.findById(data.data, function(err, dataext) {
                {
                    dataext.path = req.file.path;
                    dataext.save();
                });
            });
            res.end('File is uploaded');
    };
};
```

8. **queryprofesores:** Continuamos con la función mas compleja de todas. Su misión consiste en filtrar los profesores que encajen con la solicitud, es decir como he comentado al principio, un alumno puede buscar a su profesor por tres argumentos diferentes: Curso, Asignatura y Distancia.

Por este motivo si nos fijamos en la función, hacemos un find con tres argumentos de entre los que destaca el argumento Location. La magia de mongoDB nos permite hacer query tan impresionantes como esta, donde tenemos una base de datos de ubicaciones de profesores más la ubicación que introduce el alumno somos capaces de devolverle aquellos profesores que se encuentren a un radio de él.

```
exports.queryprofesores = function(req, res) {
    DataProfesor.find({ "curso" : req.body.Curso, "asignaturas"
        ": req.body.Clase,
        location:{$geoWithin:{$centerSphere: [ [ req.body.Loc.lat,
            req.body.Loc.lng],
        req.body.Radio / 6378100 ] } } }, function(err, dataprod)
    {
        res.status(200).jsonp(dataprof);
    });
};
```

9. **getdetail:** Por último concluimos con la función que se encarga de encontrar el perfil del profesor que el alumno solicita.

```
exports.getdetail = function(req, res){
    DataProfesor.findOne({ "_id" : req.params.id}, function(err
        , dataprod) {
        res.status(200).send(dataprof);
    });
};
```

- **Controllers Socket** Socket.js es el fichero que se encarga de gestionar el chat en la parte del servidor. Si comenzamos analizando el código de socket.js, lo primero que hacemos es que el servidor escuche en el puerto 8000.

```
server.listen(8000, '0.0.0.0');
```

Una vez que el servidor esta escuchando en el puerto 8000, debemos utilizar la librería io para establecer la conexión con el usuario que intenta tener la comunicación.

```
io.on('connection', function(socket) {}
```

Como tenemos que manejar tantos hilos de chat como profesores tengamos, necesitamos un control de canales. Por eso cada 'room' nueva viene asociado con el identificador de cada profesor, es decir cuando un profesor se registra, un nuevo canal es creado y los alumnos tienen la oportunidad de poder hablar con el profesor en esa room sin que otros profesores se tengan constancia de ello.

Las 'room' son cada uno de los canales abiertos en la comunicación del chat, donde los alumnos son libres de elegir a que 'room' entrar. Cada vez que un alumno entra en el perfil de un profesor, entra en una 'room' donde solo los que estén en el perfil del profesor podrán enterarse de lo que se comente por esa 'room'.

```
socket.on('room', function(_room) {
    room = _room.roomName;
    user = _room.userName;
    socket.join(room);
    if (room in rooms)
        rooms[room]++;
    else
        rooms[room] = 1;
    io.to(room).emit('intro', {'userName': user, 'text': "ha
        entrado en la sala"});
});
```

Cuando un alumno o un profesor que ya se encuentran en una 'room' concreta empiezan a enviarse mensajes, la forma que tenemos para gestionarlo es la siguiente:

1. El mensaje enviado por el emisor es recibido por el servidor

2. El servidor analiza el mensaje enviado por el emisor y lo trata reconociendo a que 'room' pertenece.
3. El servidor reenvía el mensaje a todo el mundo que se encuentre en esa 'room', excluyendo al emisor.

```
socket.on('newMessage', function(_room) {
    user = _room.userName;
    text = _room.text;
    io.to(room).emit('message', {'userName': user, 'text':
        text});
});
```

En caso de que alguno de los integrantes de la 'room' decida abandonar el chat, realizaremos los siguientes pasos:

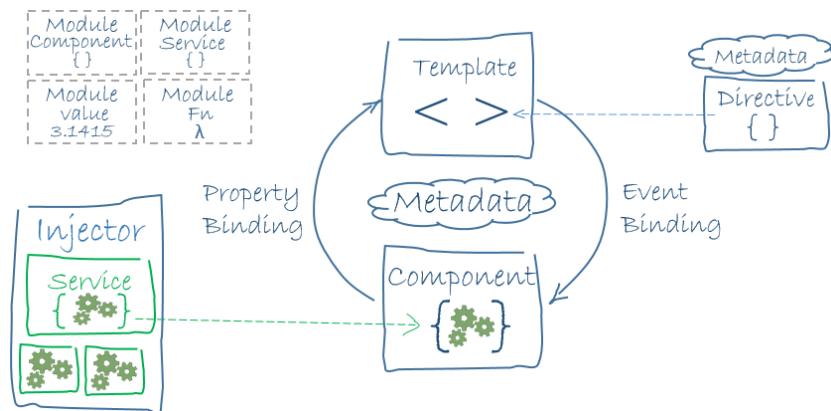
1. Recibiremos el disconnect del usuario que abandona la conexión.
2. A continuación informaremos al resto de los integrantes de la 'room' que usuario en concreto abandonó la sala.
3. Y en el caso de que todos los integrantes incluyendo el profesor no se encuentren en la sala, la eliminaremos de nuestros datos de control.

```
socket.on('disconnect', function() {
    leaveRoom();
});

var leaveRoom = function() {
    rooms[room]--;
    io.to(room).emit('client left', {'userName': user, 'text': "
        dejo la sala"});
    if (rooms[room] === 0)
        delete rooms[room];
};
```

4.5. Front-End: Angular

De los 8 bloques principales de una app en Angular, vamos a ir identificando uno a uno y que uso se le da en nuestra aplicación.



4.5.1. Modulos:

Como ya sabemos las aplicaciones en Angular son modulares y un modulo es el conjunto de código dedicado a cumplir un único objetivo, en esta sección vamos a hablar de los modulos utilizados en nuestra aplicación.

- **NgModule from '@angular/core'** Es el modulo principal, el cual recibe un objeto que define el módulo. Los metadatos más importantes de un NgModule son:
 1. Declarations: Las vistas que pertenecen a tu módulo. Hay 3 tipos de clases de tipo vista: componentes, directivas y pipes.
 2. Exports: Conjunto de declaraciones que deben ser accesibles para templates de componentes de otros módulos.
 3. Imports: Otros NgModules, cuyas clases exportadas son requeridas por templates de componentes de este módulo.
 4. Providers: Los servicios que necesita este módulo, y que estarán disponibles para toda la aplicación.
 5. Bootstrap: Define la vista raíz. Utilizado solo por el root module.
- **RouterModule from '@angular/router'** es uno de los módulos más importantes de Angular, se encuentra dentro de la librería @angular/route, gracias a el cada vez que cambiemos de dirección URL cambiaremos de página sin necesidad de tener que interactuar con el servidor.
- **HttpModule, Http, RequestOptions from '@angular/http'** Otro modulo imprescindible en una aplicación Angular, se encuentra dentro de la librería @angular/http. Gracias a este módulo podemos hacer cualquier petición AJAX sin apenas tener que escribir código.
- **FormsModule from '@angular/forms'** Modulo encargado de añadir formularios personalizados.

- **FileUploadModule from 'ng2-file-upload'** Es el modulo que nos permite subir imágenes y enviarlas a nuestro servidor, para luego poder almacenarlas de forma ordenada en nuestra base de datos.
- **AgmCoreModule from 'angular2-google-maps/core'** Gracias a este modulo, podemos utilizar la API de google maps en nuestra aplicación.
- **BrowserModule from '@angular/platform-browser'** este modulo es necesario en cualquier app que se renderice en el navegador.
- **AuthGuard from './common/auth.guard'** Gracias a este modulo, podemos conservar las credenciales de un usuario durante un tiempo determinado en el navegador.
- **ProvideAuth, AuthHttp, AuthConfig from 'angular2-jwt'** Este modulo proporciona seguridad a nuestra aplicación, generando un token encriptado para cada usuario que se registre en nuestra aplicación.
- **AppComponent, Intro, LoginAlumno, LoginProfesor , HomeAlumno, HomeProfesor, SignupAlumno, SignupProfesor, ProfesorDetail** Estos son los módulos propios que desarrollamos para nuestra aplicación, cada uno con su función independiente que contaremos a continuación.

4.5.2. Componentes:

Continuamos con los componentes de nuestra aplicación, como ya sabemos los componentes son como etiquetas nuevas, que podemos inventarnos para realizar las funciones que sean necesarias para nuestro negocio.

Nuestra aplicación se compone de 1 componente principal y 8 componentes que derivan de él. AppComponent es el componente principal y tiene la siguiente apariencia:

AppComponent

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'ClassCity';
}
```

El selector app-root o el nombre de la etiqueta que se usará cuando se desee representar, con la propiedad templateUrl asociamos un archivo .html que se usará

como vista del componente. Por último se define su estilo mediante la propiedad StyleUrls, indicando a un array de todas las hojas de estilos que deseamos.

Componentes secundarios

- **Intro** Este componente corresponde con la pagina introductoria a la aplicación donde podemos elegir entre que perfil de usuario queremos adoptar: Profesor o Alumno.
- **LoginAlumno, LoginProfesor** Componentes encargados de realizar la función de loguin del alumno o del profesor. Hemos desarrollado una función que se encarga de realizar una petición POST al servidor, si la respuesta es aceptada se almacenarán las credenciales en el "localStorage" con un timeout de 1 hora. Mientras que si la petición es rechazada, el servidor nos enviará un mensaje avisando de que: "The username or password don't match"
- **SignupAlumno, SignupProfesor** Estos componentes se van a encargar de registrar a profesores y alumnos en nuestra base de datos, para ellos hemos desarrollado una función en cada componente, que simplemente se encarga de enviar al servidor una petición POST con un body donde se encuentran los datos personales del profesor o el alumno.
- **HomeAlumno, HomeProfesor** Cuando un profesor o un alumno, es aceptado dentro de nuestra base de datos y consigue entrar en el home de la aplicación, puede realizar diferentes funciones dependiendo de si entro como alumno o como profesor.
 1. **Alumno** Un alumno, puedo realizar la búsqueda del profesor que mas le interese por diferentes parámetros:
 - El curso en el que esta el alumno
 - La asignatura que quiere cursar
 - La distancia a la que se encuentre el profesor
 2. **Profesor** El home del profesor consiste en un chat realizado con websocket, donde el profesor podrá entablar relación con cualquier alumno que este interesado en él. Aparte de poder personalizar su perfil, cambiando la foto que cada profesor tiene como avatar.
- **ProfesorDetail** Como último componente tenemos ProfesorDetail, cuando un alumno encuentra a su profesor particular ideal desde el home del alumno y hace click sobre él profesor interesado, el componente ProfesorDetail se lanza y consiste en una ficha técnica del profesor particular, así como un chat donde el alumno podrá comunicarse con el profesor para poder quedar y acordar el precio de la clase.

4.5.3. Templates:

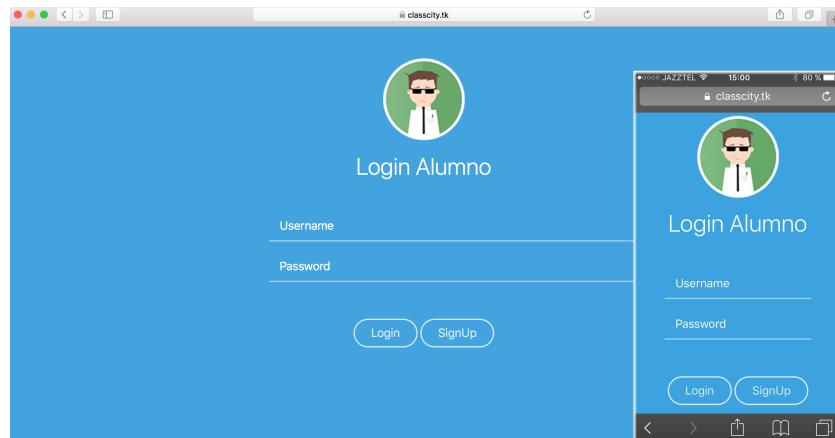
Como bien sabemos, los templates son las plantillas que se utilizan para dar forma a las aplicaciones, es la parte más visual de una aplicación web y es la propia magia de

Angular quien se encarga de renderizar estas plantillas, haciendo aplicaciones mucho mas personalizadas para el usuario. A continuación vamos a ir analizando una a una las diferentes templates que forman la aplicación:

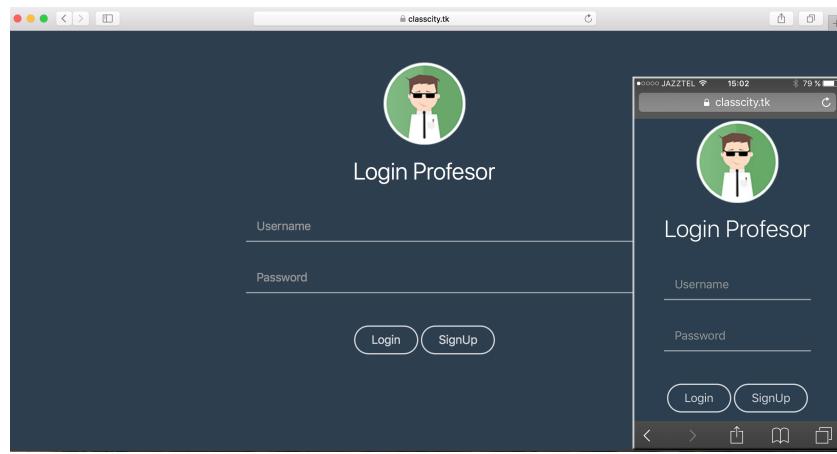
intro.html Cuando accedemos a <https://www.classcity.tk>, la primera plantilla que se nos presenta es Intro.html. Según podemos ver en la imagen, consiste en una pagina introductoria donde podemos elegir si somos alumnos o profesores.



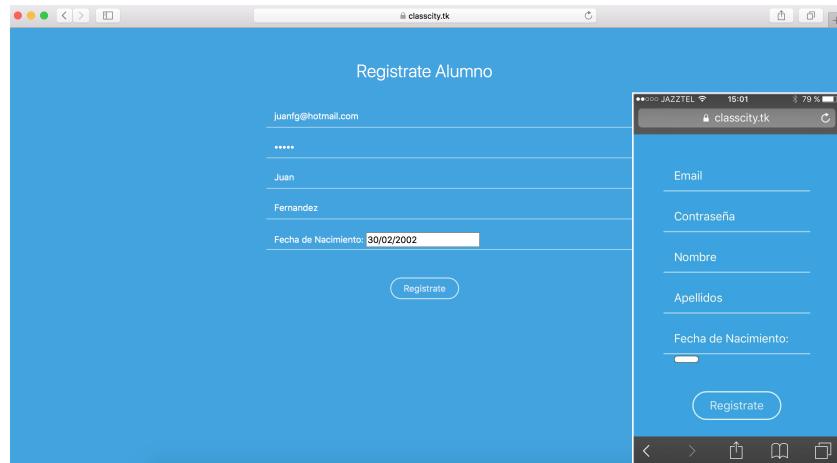
loginalumno.html Al seleccionar dentro de Intro.html en Alumno, accedemos a la siguiente plantilla donde podemos visualizar un formulario con sus campos username y password.



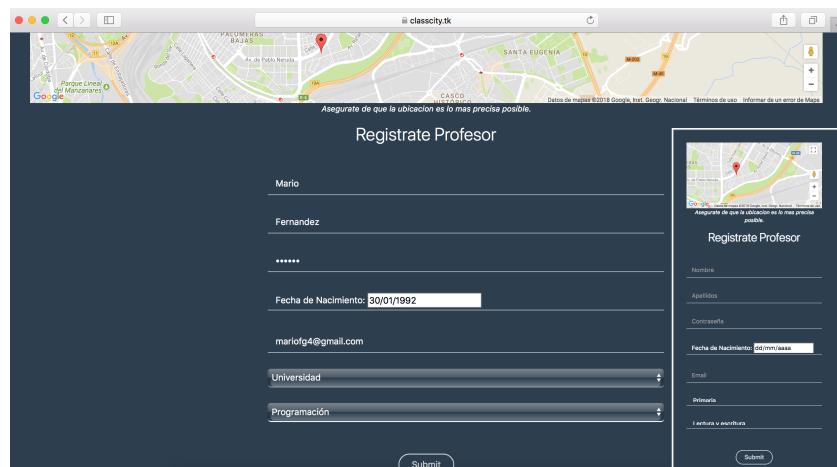
loginprofesor.html Si en vez de seleccionar Alumno hubiésemos seleccionado Profesor, hubiésemos entrado en otro formulario donde los profesores ya registrados pueden acceder a la aplicación.



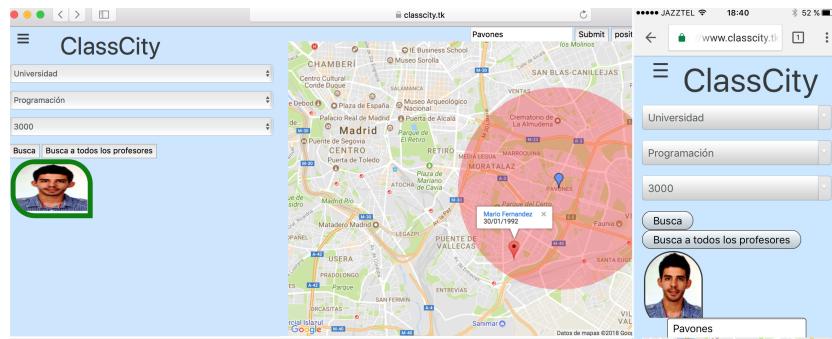
registeralumno.html Si un alumno quiere registrarse simplemente debe entrar en SignUp, donde tendrá un formulario para poder completar todos los campos necesarios.



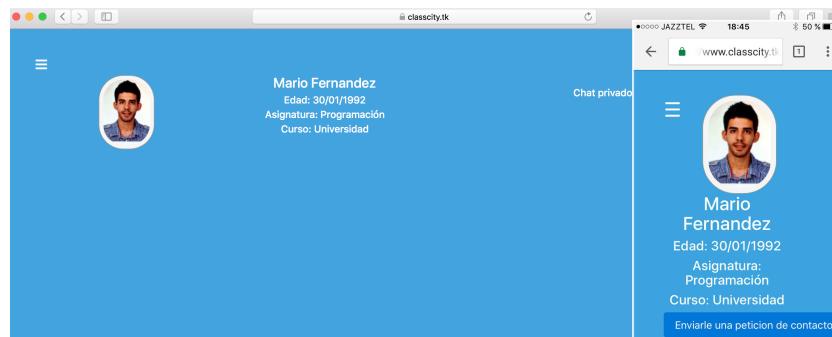
registerprofesor.html Si es el profesor es quien quiere registrarse en nuestra aplicación, entrará en SignUP de profesores e introducirá los datos necesarios.



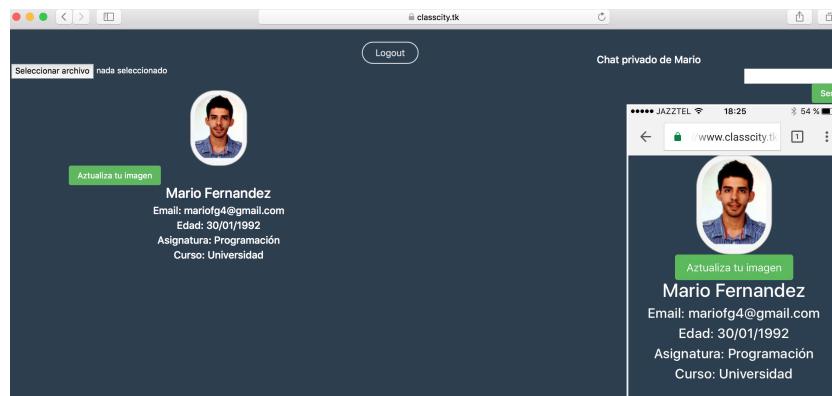
homealumno.html Una vez que el alumno ya ha sido registrado en nuestra base de datos, la interfaz con la que se encontrará el alumno es la que podemos ver en la siguiente imagen. Donde podemos encontrarnos con un Input para introducir la ubicación donde queremos buscar, por defecto nos ubicará en Madrid, Centro. También podemos diferenciar los filtros de búsqueda que tenemos donde marcamos: El curso, La asignatura y la distancia máxima en la que queremos buscar a nuestro profesor. Por último cuando el alumno busca con los parámetros que desee, les aparecerá pintados en el mapa tantos profesores como haya en nuestra aplicación con esas especificaciones.



detail.html Cuando el alumno encuentra algún profesor de su interés, puede hacer click sobre la imagen del profesor y así poder entrar en más detalle, viendo su ficha técnica y pudiendo establecer una conversación a partir del chat.



homeprofesor.html Cuando el profesor ya ha sido registrado en nuestra aplicación, el home del profesor es habilitado y en él puede hablar por un chat privado con todos los alumnos que le escriban por su canal, además de poder editar la foto de su perfil.



4.5.4. Metadatos:

4.5.5. Data Binding:

El Data Binding nos abstrae de la lógica pull/push asociada a insertar y actualizar valores en el HTML y convertir las respuestas de usuario(inputs, clicks, etc) en acciones concretas.

Como he comentado en el tema anterior Angular tiene 4 formas de Data Binding, las cuales hemos utilizado en nuestra aplicación para realizar las siguientes funciones:

Interpolación(Hacia el DOM) Angular evalúa todas las variables e introduce su resultado en el DOM. En el siguiente caso estamos insertando en el arbol DOM, todos los datos del usuario, consiguiendo una apariencia de la pagina mucho mas personalizada.

```
<code>{{decodedJwt.Email}}</code></pre>
<code>{{decodedJwt.id.nombre}}</code></pre>
<code>{{decodedJwt.id.apellidos}}</code></pre>
<code>{{decodedJwt.id.edad}}</code></pre>
```

Property binding: (Hacia el DOM) Este tipo de Data Binding, permite pasar los objetos que nosostros queramos de nuestro componente padre('home') a la propiedad (Latitud, Longitud, radius, fillColor) del componente hijo, en este caso sebm-google-map-circle. Para ello el componente hijo tiene que tener predefinido ciertas entradas en su directiva.

```
<sebm-google-map-circle
  [latitude]="query.Loc.lat"
  [longitude]="query.Loc.lng"
  [radius]="query.Radio"
  [fillColor]="'red'>
</sebm-google-map-circle>
```

Event binding: (Desde el DOM) Si queremos invocar a una función cuando se lance un evento, como por ejemplo cuando queremos hacer click en el botón de LogOut. En el momento que hacemos click sobre el botón de logout, la función logout es invocada y salimos de la sesión.

```
<button type="Submit" (click)="logout()">Logout </button>
```

Two-way binding: (Desde/Hacia el DOM)

Cuando queremos combinar el Event Binding y el Property Binding tenemos el binding bi-direccional, como podemos ver en el siguiente ejemplo.

```
<input [(ngModel)]="address">
```

En este caso queremos que el valor de address se actualice en el componente y que a su vez se introduzca dentro del input como en el caso de property binding.

4.5.6. Directiva:

Como ya he comentado las directivas son como los componentes, pero sin un template asociado. En nuestro caso hemos utilizado directivas para realizar según que funciones, por ejemplo:

FileSelectDirective, FileDropDirective, FileUploader Estas tres directivas proceden del modulo "FileUploadModule", y sirven para poder subir una imagen desde tu escritorio local al navegador, para luego poder enviar la imagen al servidor destino.

```
<input type="file" ng2FileSelect [uploader]="uploader" />
```

sebm-google-map, sebm-google-map-marker Estas son otras de las directivas procedentes del módulo ".gmCoreModule", el cual nos proporciona toda la api de GoogleMaps. Gracias a estas directivas podemos jugar con el Api de google en nuestra aplicación angular.

```
<sebm-google-map *ngIf="query.Loc"
  [latitude]="query.Loc.lat"
  [longitude]="query.Loc.lng"
  [scrollwheel]="false" [zoom] = "13">
  <sebm-google-map-marker
    [latitude]="query.Loc.lat"
    [longitude]="query.Loc.lng"
    [iconUrl] = "iconUrl">
  </sebm-google-map-marker>
</sebm-google-map>
```

También tenemos que hablar de los otros dos tipos de directivas que hemos usado en nuestra aplicación.

Directivas Estructurales: ***ngFor** repite su elemento en el DOM una vez por cada item que hay en el iterador que se le pasa, siguiendo una sintaxis de ES6.

Directivas Atributo: **ngModel** Implementa un mecanismo de binding bi-direccional. En este ejemplo el elemento HTML 'j'select'j', asigna la propiedad value a mostrar y además responde a eventos de modificación.

```
<div class="form-group">
  <select class="form-control" [(ngModel)]="query.Curso" name="curso">
    <option *ngFor="let p of curso" [value]="p">{{p}}</option>
  </select>
</div>
```

4.5.7. Servicio:

Como ya sabemos los servicios se definen a través de simples clases y son imprescindibles en Angular, ya que toda función o valor es encapsulado dentro de un servicio.

AlumnoService Este servicio se encarga de convertir una dirección física a una coordenada(latitud, longitud) para poder representarlo en el mapa.

Si observamos detenidamente el código, lo primero que hacemos en la función "getLatLan(address: string).^{es} una petición a la API de google maps con una dirección y esperamos a que nos responda. La respuesta puede ser satisfactoria o no, en caso de que sea OK las variables lat y lng serán actualizadas con los valores devueltos por google maps, si por el contrario no hubiesemos recibido respuesta, un mensaje de error sera mostrado por pantalla.

```
export class AlumnoService {
  getLatLan(address: string) {
    let geocoder = new google.maps.Geocoder();
    return Observable.create(observer => {
      geocoder.geocode( { 'address': address}, function(results,
        status) {
        if (status === google.maps.GeocoderStatus.OK) {
          let obj: Object = {lat: results[0].geometry.location.lat(),
            lng: results[0].geometry.location.lng() };
          observer.next(obj);
          observer.complete();
        } else {
          console.log('Error - ', results, ' & Status - ', status);
          observer.next({}); 
          observer.complete();
        }
      });
    });
  }
}
```

Capítulo 5

Despliegue en la nube con AWS

5.1. Amazon web services



Figura 5.1: AWS

5.1.1. ¿Qué es AWS?

Amazon Web Services es una colección de servicios de computación en la nube que en conjunto forman una plataforma de computación en la nube, ofrecidas a través de Internet por Amazon.com.

5.1.2. ¿Por qué AWS?

La cuestión de porque hemos elegido amazon web service como servicio de computación en la nube no es otra que por el gran impacto que esta teniendo en el mundo laboral, al margen de sus competidores directos como son Azure y Google Cloud.

Otro factor que juega a favor de AWS con respecto a sus competidores es que a la hora de desplegar una aplicación en la nube la experiencia de usuario me resulta mucho mas intuitiva que la de sus competidores. También debo destacar la importancia de tener un buen soporte como del que AWS dispone, donde en cualquier momento te resuelven las posibles dudas que tengas a la hora de desplegar la aplicación.

Por último y como factor de bastante importancia para aquellos pequeños emprendedores que quiera empezar a desarrollar sus ideas, es el bajo coste que supone tener una aplicación desplegada en la nube de amazon. Por el momento, el primer año de tu cuenta en AWS es gratuita, restringida a ciertas limitaciones.

5.1.3. ¿Quien lo utiliza?

Cada vez son más las empresas que deciden utilizar la computación en la nube ya que desde 2006, , Amazon Web Services proporciona servicios de infraestructura de TI para empresas en forma de servicios web.

Uno de los principales beneficios de la computación en la nube es la oportunidad de reemplazar importantes gastos de infraestructura con costes reducidos que se escalan dependiendo de la dimensión de su negocio.

Gracias a la nube, las empresas ya no tienen que planificar ni adquirir servidores y otra infraestructura de TI con semanas o meses de antelación. Pueden disponer en cuestión de minutos de cientos o de miles de servidores y ofrecer resultados más rápidamente.

Hoy en día, Amazon Web Services proporciona una plataforma de infraestructura escalable, de confianza y de bajo costo en la nube que impulsa cientos de miles de negocios de 190 países de todo el mundo. Con centros de datos en Estados Unidos, Europa, Brasil, Singapur, Japón y Australia.

A continuación enumeraremos algunas de las empresas con más éxito en AWS:

1. **Amazon.com:** Es el minorista online más grande del mundo. En 2011, Amazon.com pasó de utilizar el backup en cinta a usar Amazon S3 en la cloud para realizar copias de seguridad de la mayoría de las bases de datos de Oracle de las que se encarga. Mediante el uso de AWS, Amazon.com logró eliminar el software de backup y experimentó una mejora de desempeño 12 veces mayor, de forma que pudo reducir el tiempo de restablecimiento de 15 a 2,5 horas aproximadamente en situaciones seleccionadas.
2. **Netflix:** El referente de la televisión en streaming, usa AWS para proporcionar miles de millones de horas de vídeo casa mes a sus mas de 60 millones de suscriptores. Así puede hacer uso de miles de servidores y terabytes de almacenamiento en cuestión de minutos para que sus usuarios puedan ver series y películas desde cualquier parte del mundo en sus tabletas o teléfonos móviles.
3. **Dropbox:** El famoso y conocido servicio de alojamiento de archivos multiplataforma en la nube, utiliza hasta el momento AWS como repositorio para almacenar todos los archivos que los usuarios de Dropbox suben a la red.
4. **Bankinter:** Utiliza AWS como de su aplicación de simulación de riesgo crediticio, pasando de 23 horas a 20 minutos.
5. **FC Barcelona:** Su sitio web, que aloja más de 6 000 páginas y 12.000 fotos digitales, también usa AWS para su mantenimiento.
6. **Harvard Medical School** Desarrolla nuevos modelos de pruebas de genomas en tiempo récord.
7. **Mapfre:** Ahorró 1,3 millones de euros en infraestructura y redujo el desarrollo de semanas a días

5.2. Lanzar una instancia en AWS

Ahora que ya sabemos en detalle de lo que consiste AWS, antes de poder subir nuestra aplicación a producción debemos de crear una instancia en AWS.

Estos son los pasos a tener en cuenta a la hora de crear una instancia en AWS:

- **Paso1. Crear Key Pairs** Los Key Pairs se utilizan para iniciar sesión de forma segura en los servicios de AWS. Crearemos un Key Pairs para acceder a nuestra instancia de EC2.

1. Para crear nuevos pares de claves, navegue hasta AWS Console y luego haga clic en EC2.



Figura 5.2: Paso 1.1

2. En el panel izquierdo, haga clic en Key Pairs, luego haga clic en Crear Key Pairs

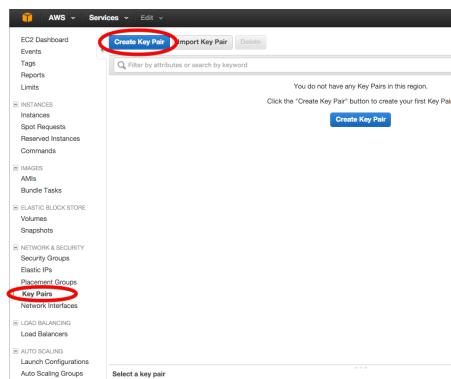


Figura 5.3: Paso 1.2

3. Ingrese un nombre para su clave, luego haga clic en Crear Key Pairs. El Key Pairs se descargará automáticamente. Debe mover esta clave a un directorio diferente.

Importante: Deberá cambiar los permisos de esta clave para que sean de solo lectura, consulte el siguiente código:

```
chmod 400 youKeyName.pem
```

- **Paso 2. Lanza una instancia de EC2 con Bitnami** En este paso, lanzaremos una instancia de EC2 desde Amazon Machine Image (AMI).

Con AMI, puede activar una instancia de EC2 que esté lista para el desarrollo sin demasiada configuración. Bitnami proporciona una imagen MEAN preconfigurada, que usaremos para configurarlo rápidamente.

1. Primero, navegue a la consola de AWS, haga clic en AWS Marketplace.

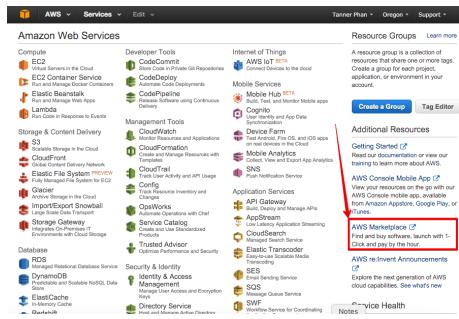


Figura 5.4: Paso 2.1

2. Search for MEAN powered by Bitnami, then select the 64-bit AMI to continue.

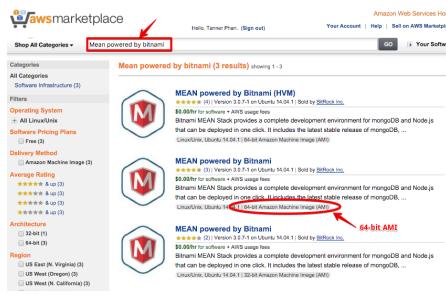


Figura 5.5: Paso 2.2

3. En Pricing Details con el fin de obtener la mejor velocidad de entrega, seleccione la región más cercana y luego haga clic en Continuar.

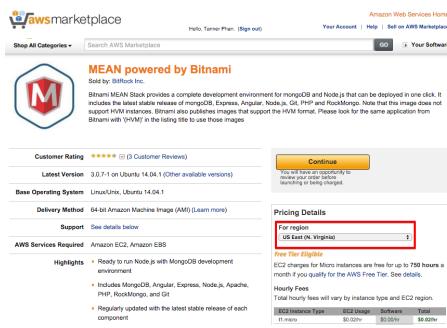


Figura 5.6: Paso 2.3

4. Para el grupo de seguridad, elija Crear nuevo según la configuración del vendedor.
5. Asegúrese de tener los siguientes métodos de conexión:

| |
|---------------------|
| 1. SSH , My IP |
| 2. HTTP , Anywhere |
| 3. HTTPS , Anywhere |

1-Click Launch
Review, modify, and launch **Manual Launch**
With EC2 Console, APIs or CLI

Click "Launch with 1-Click" to launch this software with the settings below

The default settings are provided by the software seller and AWS Marketplace.

Version
3.0.7-1 on Ubuntu 14.04.1, released 11/20/2015

Region
US West (N. California)

EC2 Instance Type
m1.small

VPC Settings
Will launch into: subnet-ddfb95b8

Security Group

Updated: Due to a change in other settings, security group settings is updated. A security group acts as a firewall that controls the traffic allowed to reach one or more instances. Learn more about Security Groups.

You can create a new security group based on seller-recommended settings or choose one of your existing groups.

Description:
A new security group will be generated by AWS Marketplace. It is based on recommended settings for MEAN powered by Bitnami version 3.0.7-1 on Ubuntu 14.04.1 provided by BiRock Inc..

| Connection Method | Protocol | Port Range | Source (IP or Group) |
|-------------------|----------|------------|-----------------------------|
| SSH | tcp | 22 - 22 | My IP 108.179.160.127/32 |
| HTTP | tcp | 80 - 80 | Anywhere 0.0.0.0/0 |
| HTTPS | tcp | 443 - 443 | Anywhere 0.0.0.0/0 |

Warning
Rules with source of 0.0.0.0/0 allows all IP addresses to access your instance. We recommend limiting access to only known IP addresses.

Figura 5.7: Paso 2.5

6. Seleccione la Key Pair que creaste en el paso 1.

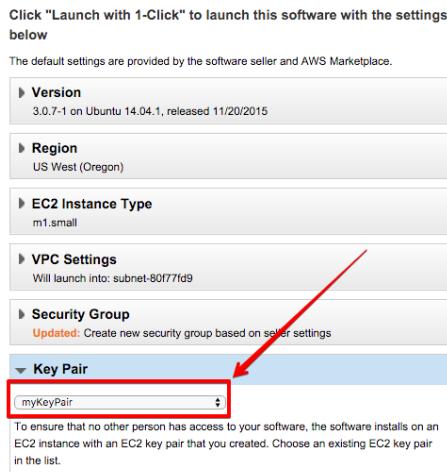


Figura 5.8: Paso 2.6

7. Finalmente, haga clic en Iniciar, para iniciar la instancia.

- **Paso 3. Conéctate a tu EC2** Para conectarte por SSH a su instancia, necesitará la IP pública de su instancia y el Key Pair que creó.

1. Dentro de EC2, haga clic en Instancias, seleccione la instancia recién iniciada en el paso anterior y luego haga clic en Conectar.
2. Copie el código que aparece marcado en rojo en la siguiente imagen.

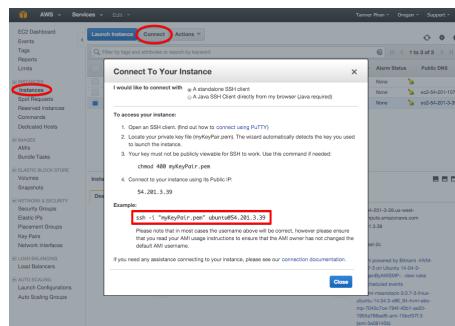


Figura 5.9: Paso 3.2

3. En su terminal, navegue hasta el directorio donde está guardado su par de claves, luego pegue el último paso del código.

```
The authenticity of host '54.201.14.233 (54.201.14.233)' can't be established.
ECDSA key fingerprint is SHA256:ITSoxli1GDHtY8Uhi1OB8NqM8tH3B807LzR8F/f1T/xE.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '54.201.14.233' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-71-generic x86_64)

*** Welcome to the Bitnami MEAN 3.0-7-3
*** Bitnami Wiki: https://wiki.bitnami.com/
*** Bitnami Forums: https://community.bitnami.com ***
bitnami@ip-172-31-1-86:~$
```

Figura 5.10: Paso 3.3

- **Paso 4. Conseguir un dominio gratuito** Una vez que ya tenemos la instancia creada y hemos sido capaces de conectarnos a ella por SSH, llega el momento de conseguir un dominio para que los usuarios se puedan conectar a la aplicación lo más fácil posible. El dominio que he utilizado es un dominio gratuito, proporcionado por el portal freedom.com, la forma de conseguirlo es la siguiente:

1. Vaya a <https://my.freedom.com> e inscríbase.

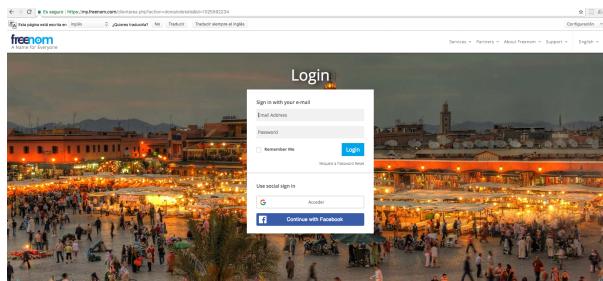


Figura 5.11: Paso 1.1

2. Una vez que nos hemos logueado, tenemos que ir a la sección de My Domains y allí elegir un dominio que este disponible.

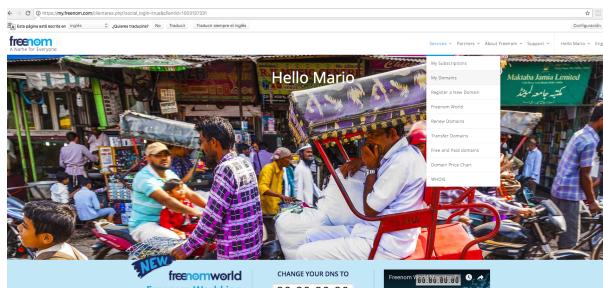


Figura 5.12: Paso 1.2

3. En nuestro caso, el dominio elegido es www.classcity.tk. Una vez creado nuestro dominio, comenzamos la configuración haciendo click en el botón Manage Domain.

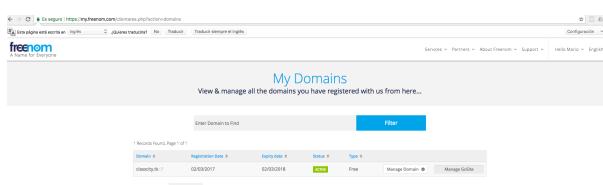


Figura 5.13: Paso 1.3

4. Dentro del Managin del dominio, nos vamos a la sección Manging Tools, y hacemos click en nameerver tal y como se puede ver en la siguiente imagen,

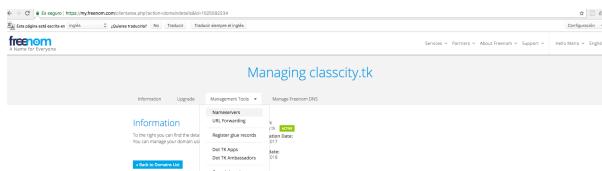


Figura 5.14: Paso 1.4

5. Debe cambiar los servidores de nombres y seleccionar usar servidores de nombres personalizados.

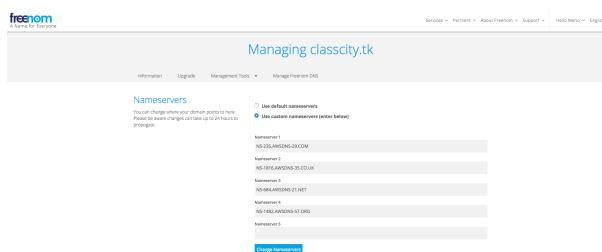


Figura 5.15: Paso 1.5

- **Paso 4. Obtener un certificado SSL gratis con AWS Certificate Manager para CloudFront**

Un certificado SSL es un fichero informático generado por una entidad de servicios de certificación que asocia unos datos de identidad a una persona física, organismo o empresa, confirmando de esta manera su identidad digital en Internet. Necesitamos un certificado SSL para que los usuarios que accedan a nuestra página lo hagan de una forma segura por el protocolo HTTPS.

Por otro lado tenemos Amazon CloudFront, es un servicio de red de entrega de contenido (CDN) global que proporciona datos, vídeos, aplicaciones y API de forma segura a sus espectadores con baja latencia y altas velocidades de transferencia.

Para conseguir el certificado SSL en AWS necesitas seguir los siguientes pasos:

1. Tienes que ir a AWS manager certificate desde la consola de Amazon web services y hacer click en get started.
2. A continuación te pedirá el dominio creado previamente y tendrás que hacer click en Review and request.
3. Una vez enviado la solicitud de certificado a la Autoridad certificadora, un email de confirmación será enviada a nuestra cuenta de correo vinculada con el dominio, es decir a admin@classcity.tk. En nuestro caso como disponemos de un dominio gratuito debemos hacer una redirección desde el portal <http://www.tkmailias.tk/es/pageA00E1.html>, y desde allí conseguir redireccionar nuestra cuenta admin@classcity.tk a un correo personal, para recibir la confirmación del certificado.

- Una vez que recibimos un correo como el que aparece en la siguiente imagen, realizamos la confirmación del certificado haciendo click en el enlace que viene en el correo

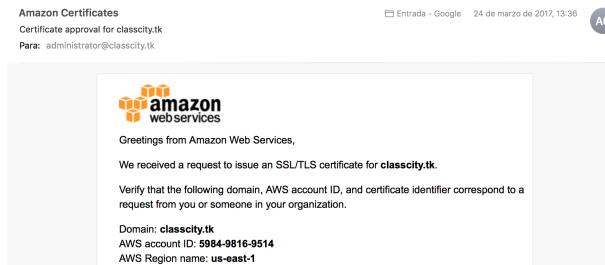


Figura 5.16: Correo de confirmación

Una vez que tenemos el certificado SSL generado es momento de configurar cloudfront proporcionándole de dicho certificado, para ello debemos seguir los siguientes pasos:

- Buscar cloudfront desde la consola de AWS.
- Una vez allí debemos crear una distribución
- En el momento de elegir la configuración del SSL certificate, marcaremos la opción Custom SSL Certificate, tal y como podemos ver en la siguiente imagen.

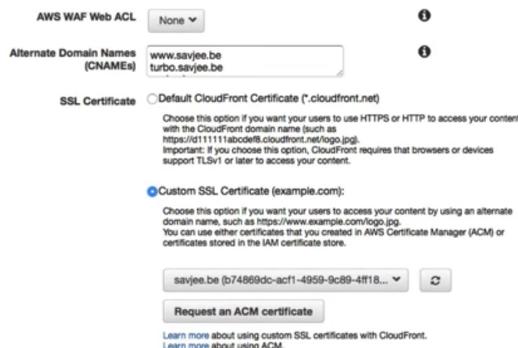


Figura 5.17: SSL

Si ahora vamos a nuestro dominio e introducimos antes el protocolo HTTPS, es decir en nuestro caso <https://www.classcity.tk>, veremos que efectivamente accedemos a nuestra aplicación de una forma segura.



Figura 5.18: SSL

5.3. Lanzar una aplicación MEAN a producción

Una vez que dispongamos de una instancia, un dominio, un certificado SSL y una distribución cloufront, tenemos todo listo para poder desplegar nuestra aplicación MEAN en producción. Para ello debemos seguir los siguientes pasos:

1. Conectarnos desde nuestra terminal por SSH a nuestra maquina virtual de AWS

```
sudo ssh -i /.ssh/keypair.pem root@ipinstance
```

2. Clonar en la maquina virtual de AWS, nuestro repositorio git donde almacenemos la aplicación MEAN.

```
rooti@ip sudo git clone https://github.com/RoboticsURJC-students/2016-tfg-Mario-Fernandez.git
```

3. Instalar dependencias de la aplicacion web por parte del Front-End

```
rooti@ip cd 2016-tfg-Mario-Fernandez
rooti@ip /2016-tfg-Mario-Fernandez sudo npm install
```

4. Correr Angular2 con angular-ci

```
rooti@ip /2016-tfg-Mario-Fernandez sudo npm run build
```

5. Una nueva carpeta se crea al terminar el proceso ./dist, debemos copiarla entera en nuestra carpeta httdocs de apache.

```
rooti@ip /2016-tfg-Mario-Fernandez sudo cp -r ./dist/* ../httdocs
```

6. Ahora deberíamos poder ir a <https://www.classcity.tk> y poder ver nuestra aplicación Angular2 corriendo en producción.



Figura 5.19: <https://www.classcity.tk>

7. No olvidemos que en el stack MEAN, aun falta que configuremos el servidor en producción. Para ello lo primero es instalar las dependencias necesarias para el backend.

```
rooti@ip /2016-tfg-Mario-Fernandez cd backend
rooti@ip /2016-tfg-Mario-Fernandez/backend sudo npm install
```

8. Antes de ponernos a ejecutar nada en la parte del backend, debemos configurar ciertas rutas en nuestro servidor apache que va ser el encargado de recibir la peticiones de entrada. Para ello debemos hacer lo siguiente:

- Editamos el archivo de configuración de apache llamado httpd.conf para que cuando llegue las solicitudes de tipo / app / * redirigir a localhost: 8080, que será donde estará escuchando nuestra aplicación.

```
ProxyPassMatch ^/app/(.*)$ http://localhost:8080/$1
ProxyPass /app/(.*)$ http://localhost:8080/
ProxyPassReverse /app/(.*)$ http://localhost:8080/
```

- Y cuando llega la solicitud de type / socket / * redirigir a localhost: 8000, que será el puerto donde este escuchando nuestro chat.

```
ProxyPassMatch ^/socket/(.*)$ http://localhost:8000/
$1
ProxyPass /socket/(.*)$ http://localhost:8000/
ProxyPassReverse /socket/(.*)$ http://localhost:8000/
```

9. A continuación en el backend correremos la base de datos con la propiedad screen, para que cuando se termine la conexión por ssh el proceso siga corriendo y no se corte.

```
rooti@ip /2016-tfg-Mario-Fernandez/backend sudo mkdir data
rooti@ip /2016-tfg-Mario-Fernandez/backend sudo screen
mongod --dbpath ./data
```

10. Y por último ejecutamos e fichero server.js también con la propiedad screen, y comprobaremos que funciona correctamente.

```
rooti@ip /2016-tfg-Mario-Fernandez/backend sudo node server
.js screen
```