

Capítulo 3

Infraestructura

Una vez introducidos en el proyecto y con los objetivos marcados se hablara 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 posé.

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 línea de comandos, nos descargamos todas las dependencias contenidas en nuestro package.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 característica 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)

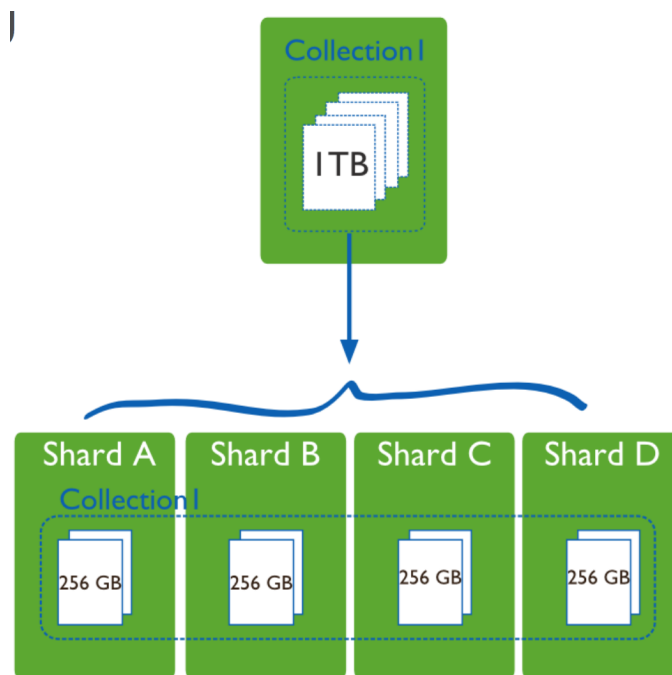


Figura 3.2: Sharding

¿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



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 "miniaplicación", 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 encontrasen 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 de 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



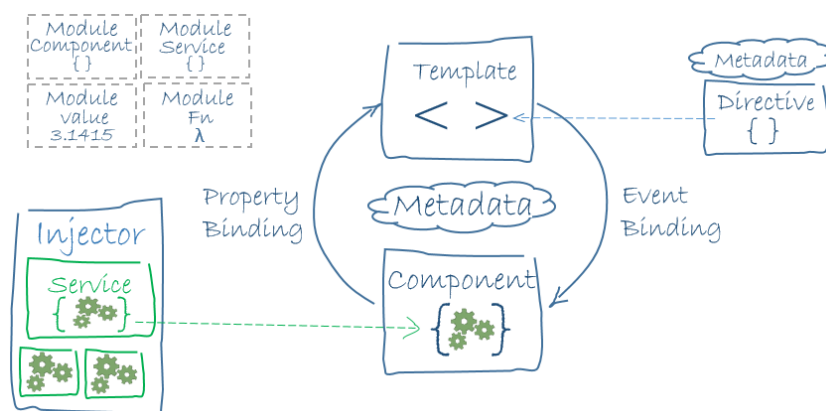
Angular es un framework JS para la parte cliente o Frontend de una aplicación web, que respeta el paradigma MVC y permite crear Single-Page Applications (Aplicaciones web que no necesitan recargar la página), de manera más o menos sencilla. Es un proyecto mantenido por Google y que actualmente está muy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

- `*ngForg`
 - `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';

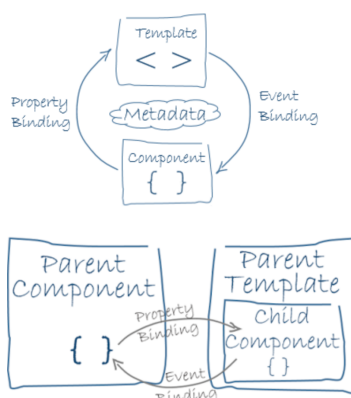
@Component({
  selector:      'todo-list',
  templateUrl:  'todo-list.component.html',
  styleUrls:    ['todo-list.component.css'],
  moduleId:     module.id,
  directives:   [TodoDetailComponent],
  providers:    [TodoService]
})
export class TodoListComponent { ... }
```

- **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 árbol 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 pensando “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)**