Capítulo 1

Introducción

1.1. Motivación

1.1.1. Antes de empezar...

Antes de empezar a desarrollar una aplicación, debes saber que no será un camino sencillo, pero si muy gratificante. Cuando empecé a desarrollar la aplicación no sabía la cantidad de problemas a los que me iba a enfrentar, pero como tenía la ilusión de poder hacer realidad una idea todo fue más sencillo.

1.1.2. Classcity

La idea de la que hablo se llama Çlasscityz consiste en una aplicación web cuya función principal es contactar profesores con alumnos para que puedan quedar y dar clases particulares de cualquier asignatura y curso. Esta idea nace porque llevo muchos años dedicándome a la enseñanza de clases particulares y pienso que hay una gran oportunidad de organizar mejorar la comunicación profesor-alumno.

1.1.3. M.E.A.N.

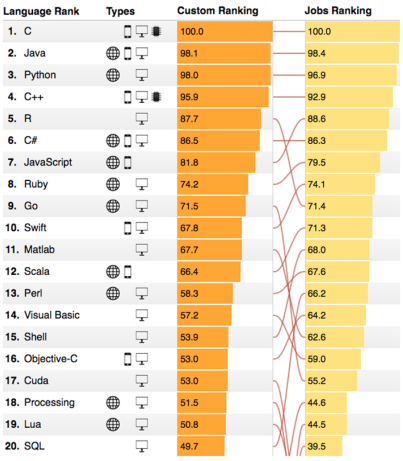
Classcity ha sido desarrollada con 4 tecnologías diferentes que forman el famoso stack M.E.A.N. haciendo referencia a las siglas Mongo, Express, Angular y Node. Desde el cliente al servidor pasando por la base de datos, todas con el mismo punto en común. Desarrollo end-to-end usando JavaScript tanto en el frontend, backend y la base de datos. Sin entrar en definir complementamente cada una de las tecnologías que lo componen nos encontramos con:

- MongoDB como base de datos que almacena documentos JSON.
- Express como web framework basado en Node.js que nos permite crear API REST, por ejemplo
- Angular como framework para crear la parte cliente de la aplicación en formato Single Page.

- Node.js como framework JavaScript basado en V8 que proporciona funcionalidades core para nuestra aplicación bajo un modelo asíncrono de eventos.

1.1.4. Javascript

MEAN como cualquier plataforma no puede vivir sin su comunidad, pero si tenemos en cuenta de que todas las tecnologías utilizan javascript como lenguaje de programación principal, podemos estar hablando de un stack bastante importante para la comunidad javascript, ya que todos los desarrolladores de front se podrán adentrar dentro del back y así poder hacer su propia aplicación.



Language Rank	Types	Custom Ranking	Jobs Ranking
1. C		100.0	100.0
2. Java		98.1	98.4
3. Python		98.0	96.9
4. C++		95.9	92.9
5. R		87.7	88.6
6. C#		86.5	86.3
7. JavaScript		81.8	79.5
8. Ruby		74.2	74.1
9. Go		71.5	71.4
10. Swift		67.8	71.3
11. Matlab		67.7	68.0
12. Scala		66.4	67.6
13. Perl		58.3	66.2
14. Visual Basic		57.2	64.2
15. Shell		53.9	62.6
16. Objective-C		53.0	59.0
17. Cuda		53.0	55.2
18. Processing		51.5	44.6
19. Lua		50.8	44.5
20. SQL		49.7	39.5

En este cuadro podemos encontrar información relevante de cómo están valorados los lenguajes de programación, en relación a la popularidad general de cada lenguaje de programación, pero también aplicando filtros por plataforma(computadores, móviles, web y sistemas embebidos) y por solicitud de empleo. Este estudio comenzó seleccionando unos 300 lenguajes en Github y escogiendo los más populares hasta quedarse con estos 20 candidatos para después estudiar la popularidad de esos lenguajes en 10 plataformas, que incluyen la propia Github, Google, Stack Overflow y paginas de oferta de empleo como Career Builder o Dice.

Javascript a pesar de estar bien posicionado en el ranking tiene ventajas como que es un lenguaje muy sencillo y versátil, puesto que es muy útil para desarrollar páginas web y aplicaciones web. Otra cualidad que hace importante a javascript es su rapidez de computo, pudiendo ejecutar funciones inmediatamente, y por último y para mi la ventaja más importante es que es el único lenguaje que te permite trabajar modo FullStack en cualquier tipo de desarrollo de programación. Pero no todo puede ser buenas noticias para javascript, también tiene inconvenientes como por ejemplo que en el FrontEnd sus códigos son visibles, por lo tanto pueden ser leídos por cualquier usuario, y por otro lado javascrip también tiende a introducir gran cantidad de fragmentos de código en los sitios web.

1.1.5. Plataforma Cloud

También tengo que añadir la complejidad de subir una aplicación en la nube a través de la plataforma Amazon Web Service. Hay muchas otras plataforma de cloud como Azure(Microsoft), Google Cloud... Pero elegí AWS debido a que tiene un periodo de prueba gratuito y es bastante mas sencillo de manejar que Azure o Google Cloud. Además AWS es el líder indiscutible, aunque sus competidores últimamente alcanzan mas importancia en el mundo del cloud.

Capítulo 2

Objetivos

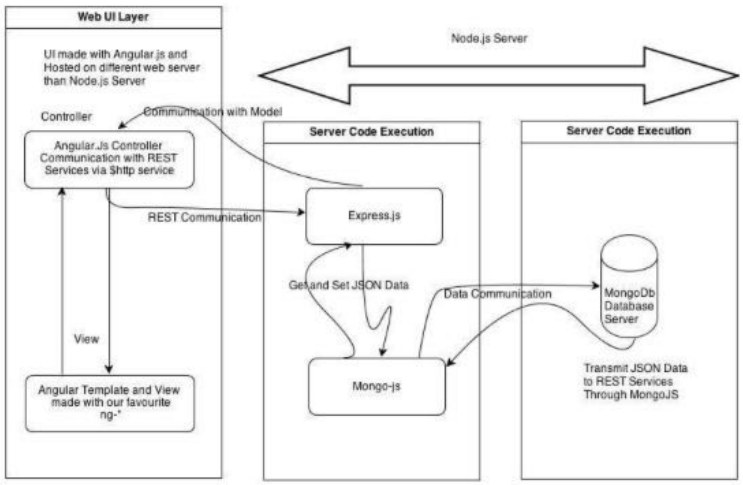
Cuando se empieza a desarrollar una aplicación se suelen cometer muchos errores que ralentizan el periodo de aprendizaje. Para intentar perder el menor tiempo posible es recomendable seguir ciertos pasos:

2.1. Familiarizarse con el lenguaje

Lo primero que debemos hacer antes de empezar a desarrollar nada, es relacionarnos con el lenguaje, en nuestro caso es bastante sencillo porque solo necesitamos manejar javascript para desarrollar la aplicación entera. Lo suyo es empezar haciendo pequeñas funcionalidades en javascript para quitarnos el miedo de encima. Recomiendo hacer pequeños programas que nos ayude a entender las particularidades de javascript.

2.2. Como funciona M.E.A.N.

Una vez que manejamos javascript con bastante soltura, podemos empezar a entender como funciona nuestros stack.



En este diagrama podemos apreciar el diagrama de bloque del stack MEAN, en el distinguimos tres grandes bloques:

- Web UI Layer: Este bloque hace referencia a la interfaz de usuario, o lo que podemos entender como el cliente(frontend) desarrolado en su totalidad por el framework Angular 2. Esta parte del stack es el que le llega a los usuarios, forma una parte esencial en la aplicación ya que es en todo sus efectos la cara visible de la aplicación.
- Server Code Execution : Este segundo bloque hace referencia al servidor encargado de atender a cualquier petición del cliente. Este bloque es vital para la inteligencia de la aplicación ya que es el encargado de hacer de intermediario entre el cliente y la base de datos.
- Database Server: Por último destacamos el bloque perteneciente a la base de datos, sin ella no podriamos tener una amplia lista de alumnos y profesores.

2.3. Pasos para desarrollar una app MEAN

Una vez que tengamos una buena soltura con javascript y entendamos el funcionamiento del stack MEAN, podemos empezar a meternos en materia.

- 1. Lo primero sería tener montado un "Hola Mundo.[en] MEAN, para tener organizada nuestro directorio y hacer modificaciones desde ahí.
- 2. Una vez tengamos nuestra aplicación web organizada, es típico error comenzar con la parte del cliente ya que parece una parte más vistosa y por lo tanto más amena. La apariencia de la app debe ser lo que menos te debe importar en este momento. Lo suyo es empezar por el desarrollo del back-end, aportandole la inteligencia necesaría para que con un cliente artificial, llamado postman podamos hacer pruebas de su funcionamiento.
- 3. Una vez tengamos un backend solido y consistente, empezamos con la implementación de controladores y modelos para la gestión de la base de datos. Esto permite que el backend pueda hacer cualquier petición a la base de datos y devolver el resultado a cliente. Volveremos hacer pruebas con Postman nuestro cliente artificial que nos permite saber si nuestro backend esta funcionando correctamente.
- 4. Por último y cuando estemos seguros de que nuestro backend y nuestra base de datos funcionan correctamente, empezaremos a desarrollar nuestra interfaz de usuario. Digamos que es la parte más bonita del desarrollo de una aplicación, debido a que los cambios en código suelen tener un cambio visual que hace que la programación sea más entretenida.

Capítulo 3

ClassCity

ClassCity es una aplicacion que consiste en facilitar la comunicacion entre profesores y alumnos para que puedan reunirse. Es el claro ejemplo de una aplicación API REST donde tenemos una parte cliente(front-end) y otra parte servidor(backend), las cuales se comunican a partir de ficheros en formato JSON.

3.1. Front-End

Esta parte de la aplicación corresponde al modelo cliente, el cual ha sido desarollado con el framework Angular2.

3.1.1. Fichero Raiz

Cuando el cliente accede a www.classcity.tk el fichero raiz de la app es index.html

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>ClassCity</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1"
     >
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs
     /twitter-bootstrap/4.0.0-alpha.5/css/bootstrap.css">
  <link rel="icon" type="image/x-icon" href="./assets/images/favicon.
     ico">

</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

Este fichero html no es el unico que se descarga cuando nos bajamos una aplicación en angular.

1. **Main.ts** Este es el fichero raiz encargado de montar todo el cliente de nuestra app.

```
import './polyfills';

import { platformBrowserDynamic } from '@angular/platform-
    browser-dynamic';
import { enableProdMode } from '@angular/core';
import { environment } from './environments/environment';
import { AppModule } from './app/app.module';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule);
```

2. **App.Module.ts** Llegamos a uno de los ficheros mas importantes de toda la aplicación, donde se concentra todo lo necesario para que la app funcione. Si analizamos el fichero nos encontramos con un monton de importaciones, las cuales las he intentado separar por espacios en blancos dividiendolas en tres partes: las primeras importaciones están relacionadas con el framework angular2, las segundas importaciones estas relacionadas con servicios que he utilizado y las ultimas importaciones corresponden a la diferentes partes de la aplicación.

   Si continuamos profundizando en el codigo encontramos una variable ROUTES, la cual se encargara de asignar a cada path un componente especifico.

   Por último encontramos el motor de la aplicación

```
  @NgModule({
declarations: [
  AppComponent,
  Intro,
  LoginAlumno,
  LoginProfesor,
  SignupAlumno,
  SignupProfesor,
  HomeAlumno,
  HomeProfesor,
  ProfesorDetail
],
imports: [
  BrowserModule,
  FormsModule,
  HttpModule,
  FileUploadModule,
  AgmCoreModule.forRoot({
    apiKey: 'AIzaSyCYUVL5zFNpT0vaziTcpEUUbsmqZ7YRERM'
  }),
  RouterModule.forRoot(ROUTES)
],
```

```
  providers: [AuthGuard, {provide: AuthHttp, useFactory:
      authHttpServiceFactory,
    deps: [ Http, RequestOptions ]}, AlumnoService],
  bootstrap: [AppComponent]
})
```

Si observamos las importaciones, encontramos modulos como:

1. **FormsModule** Lo utilizamos para los formularios.
2. **HttpModule** Agiliza las peticiones tipo AJAX.
3. **FileUploadModule** Libreria necesaria para el envio de imagenes.
4. **AgmCoreModule** Libreria GoogleMaps para Angular, apiKey es la clave necesaria para realizar peticiones al servidor de google.

Justo después de las importaciones llegan los "Providers", que como su propio nombre indica son "proveedores de servicios". En este caso utilizaremos servicios como AuthHttp para una autenticación mas segura.

3.1.2. Servicios

Classcity es una aplicación web que necesita de la comunicación con varios backends para poder funcionar. De aquí los servicios que interactúan con nuestro backend y con otros como el de google.

1. **Angular/router** Este es un servicio interno en la libreria de angular, el cual capacita a la aplicación a poder enrutar cualquier URL interna.
2. **Angular2-jwt y Authguard**
3. **Angular2-google-maps**
4. **Imagenes**
5. **Google Service Geolocation**

3.1.3. Componentes

A continuación se presentan los hitos en orden cronológico:

1. **Login**
2. **Sign-up**
3. **Buscar Profesores**
4. **Navigator Geolocation**
5. **Logout**
6. **Notificacion**
7. **Chat**

Capítulo 4

Tecnologías

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.

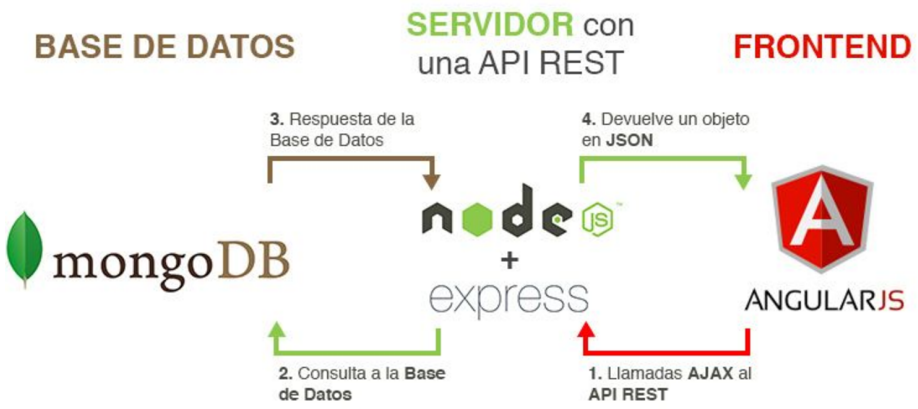


Figura 4.1: Esquema MEAN

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 tecnologias:

4.1. MongoDB

4.1.1. Introduccion

Mongo 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. 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 despues usar el tipo de persistencia que mas se amolde a nuestras necesidades.

Las caracteristica 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 asi nuevos tipos de aplicaciones.

4.1.2. MongoDB

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

```
{
    name: "mario",
    age: 24,
    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 dinamico 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 coleccion.

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

Un problema que tiene Mongo, es que no soporta transacciones de multiples documentos, sin embargo puede proporciona 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 economicas, no utilizan Mongo debido a que es sensible a Hacker, debido a que no es capaz de hacer una sola operación atomica en dos documentos.

Otro posible problema podria ser la excesiva cantidad de memoria RAM que puede consumir MongoDB, aunque es posible ejecutar mongo en una maquina con una pequeña cantidad de memoria RAM libre. Pero si es cierto que Mongo usa automaticamente toda la memoria libre del equipo como su cache, es por esto por lo que los monitores de recursos muestran que mongo utiliza una gran cantidad de memotria, 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.

4.1.3. Fragmentación (Sharding)

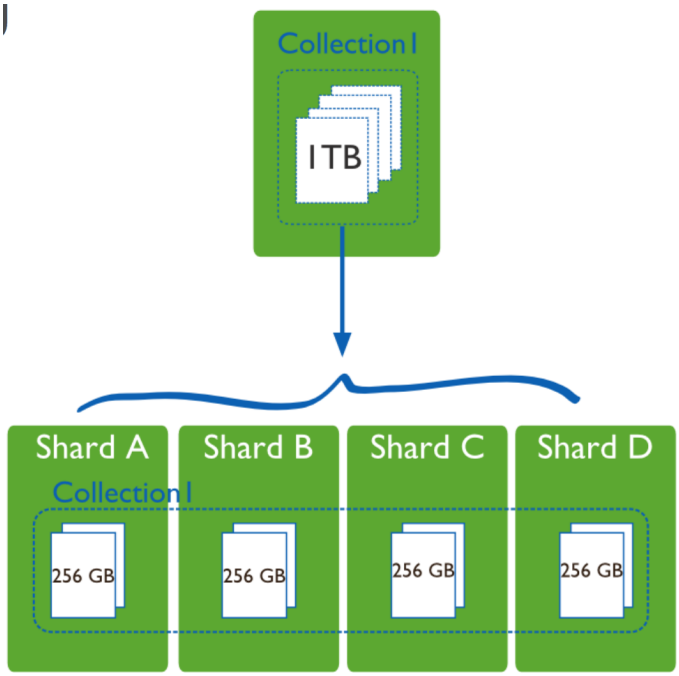


Figura 4.2: Sharding

¿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 insfraestruztura 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 coleccion en una base de datos, la tendriamos en varias bases de datos distribuidas, de modo que a la hora de consultar los datos de dicha coleccion, los recuperaremos como si de una unica base de datos se tratase. Todo esto de encontrar la base de datos lo hace Mongo de forma transparente. Cuando hacemos consultas, tenemos un enrutador llamado "MongoS", el cual mantendra 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 coleccion ordenada, se utilizan rangos y shard key, de modo que se torcea la colecicion en rangos y se les asigna un id a cada rango, de este modo que cuando se consulte la coleccion debemos proporcionar el shardKey.

4.2. Express

4.2.1. Introduccion

Express es un framework por encima de 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.

Express es una plataforma ligera para construir aplicaciones web usando NodeJS. Nos ayuda a organizar las aplicaciones web en el lado del servidor. En la web de ExpressJS, lo describen como ün framework de desarrollo de aplicaciones web minimalista y flexible para Node.js". Sin duda el exito de express radica en lo sencillo que es usarlo, y además abarca un sin número de aspectos que muchos desconocen pero son necesarios.

De entre las tantas cosas que tiene este framework podemos destacar:

- **Router**
- **Manejo de peticiones**
- **Configuracion de la aplicacion**
- **Middleware**

Para comenzar, y suponiendo que tienes ya instalado node y npm, lo único que es necesario hacer es lo siguiente:

```
> npm install -g express
```

Una vez instalado express puedes utilizar express NOMBRE-DEL-APP lo cual te creara una estructura personalizada y los archivos necesarios para comenzar a trabajar con el mismo. Haz cd NOMBRE-DEL-APP npm install para instalar automaticamente las dependencias.

4.2.2. Server.js

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

- **Modúlos 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 nuesta 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, porque si estas no están definidas, no existiria una interfaz para él cliente.

  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(3000);
```

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. Algunos nombres destacados que usan Express son:

- **MySpace**
- **LinkedIn**
- **Segment.io**

4.3. Angular 2

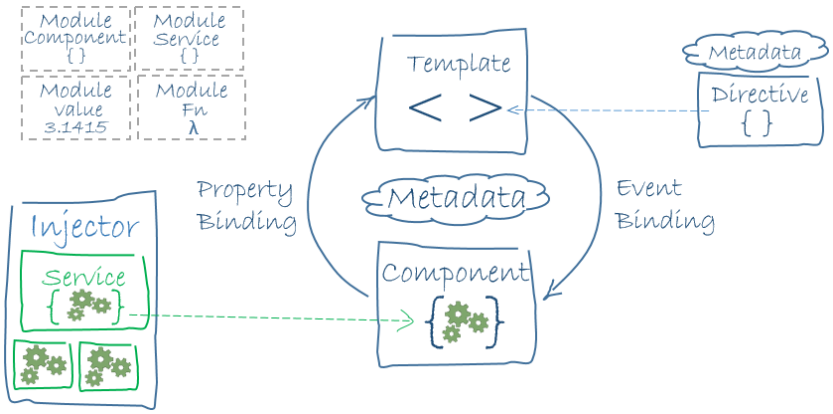
4.3.1. Introduccion

Angular 2 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.

Para crear apps en Angular 2 necesitamos:

- **Componemos plantillas HTML (templates) con el markup de Angular 2**
- **Escribimos Componentes para gestionar esas plantillas**
- **Encapsulamos la lógica de la aplicación en Servicios**
- **Entregamos 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:

- **Modulo** Igual que con su predecesor, las apps de Angular 2 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 modulos 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 2 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 1), un componente vendría a ser un controlador que siempre va ligado a una vista.

- **Template** El Template (cuyo concepto ya existía en Angular 1), es lo que nos permite definir la vista de un Componente. Igual que su predecesor, el template de Angular 2 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**

- **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”
- **Directiva**
- **Servicio**
- **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.

4.4. Node

Node es un entorno de programación en JavaScript para el Backend basado en el motor V8 de JavaScript 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.

- **NPM**.Uno de los beneficios de Node es su gestor de paquetes, npm (node package manager) y 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 packages/paquetes para satisfacer nuestras necesidades. Los packages amplían la funcionalidad de Node y este sistema de paquetes una de las cosas que hace a Node tan potente. La capacidad de tener una serie de códigos que puedes reutilizar en todos tus proyectos es increíble y hace que el desarrollo sea mucho más sencillo. Puedes combinar varios paquetes para crear aplicaciones complejas.