Captainer:

Gestión de Implantaciones y soporte de incidencias

Proyecto de fin de Ciclo: Desarrollo de Aplicaciones multiplataformas

Por: Juan José González – 2º DAM - 2019

Briefing de la presentación

- *Precedentes: motivaciones y planteamiento inicial*
- *Toma de decisiones: Plataformas de desarrollo*
- *Análisis de la estructura del proyecto*
- *Microframework Lumen y MySQL: Migraciones y modelos de tablas*
- *Angular I: Diseño de la interfaz y componentes iniciales de la SPA*
- *Angular II: Consumo del servicio REST mediante servicios HTTP*
- *Angular III: Enrutamiento y autenticación mediante JWT y Rol de usuario*
- *Valoraciones finales y pruebas de la aplicación*

Precedentes

Motivaciones y planteamiento inicial

La empresa necesita un software corporativo que requiere las siguientes características:

Gestión de incidencias e implantaciones por roles de usuario

65 %

Comunicación entre trabajadores de la empresa

50 %

Fácil acceso a las incidencias de sus clientes

80 %

Acceso desde cualquier dispositivo de la empresa

35 %

Empleo de lenguajes de programación multiplataforma:

Que nos permita la interacción con cualquier dispositivo.

Conexión en red con servidor local

Necesariamente, requerimos herramientas para trabajar en red

Entorno simple, ligero y seguro

Plataforma que sea fácil de manejar, segura a la hora de utilizarla y no sobrecargue el servidor.

TOMA DE DECISIONES

Plataformas de desarrollo

Framework Lumen 5.5:

Como ORM del la aplicación (PHP)

Framework Angular 7

Front-End (Typescript - HTML)

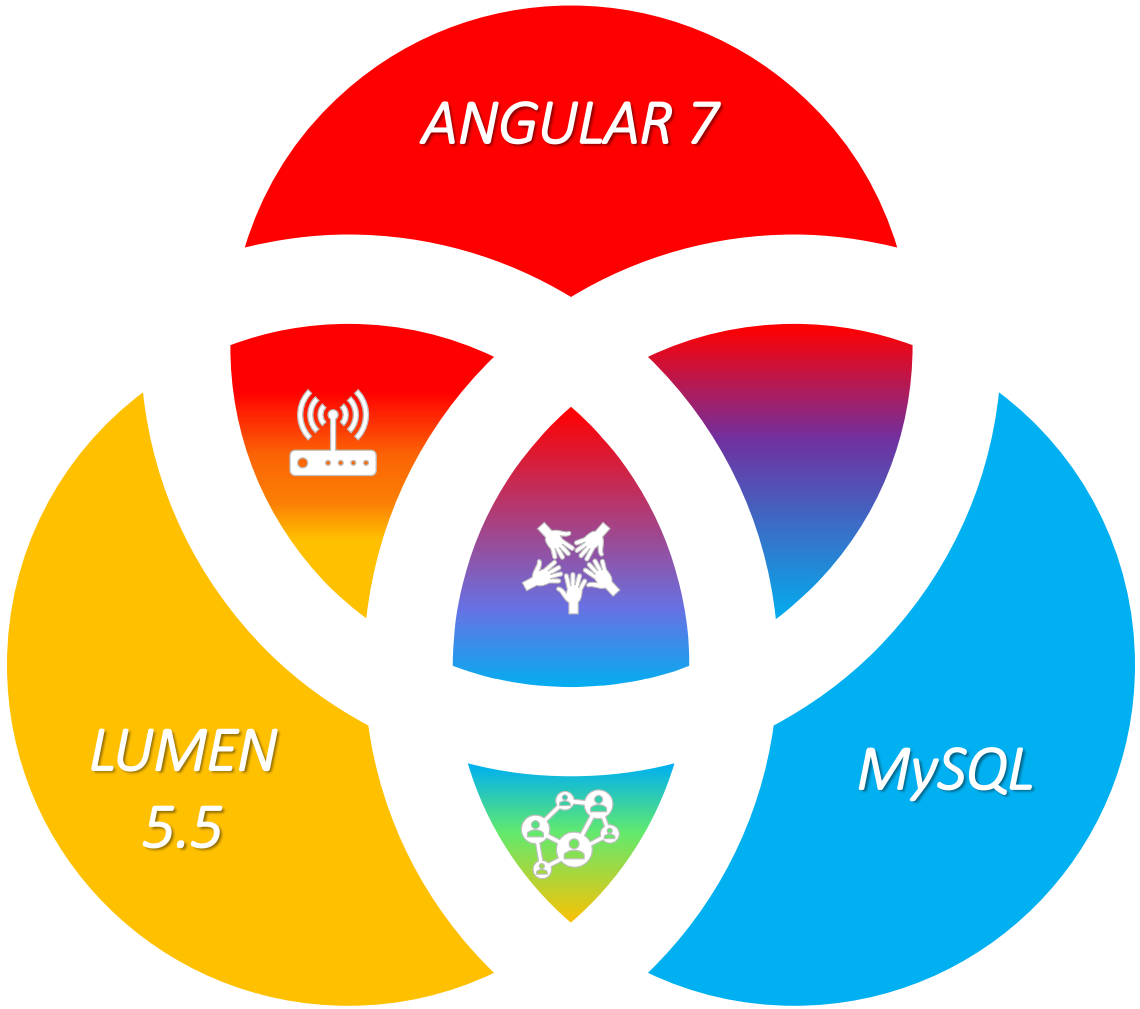
Base de datos MySQL:

Estructura relacional empleada por la empresa

Framework Bootstrap 4

Pautas de diseño

Equipo de Programación



Análisis de la estructura del proyecto

Comunicación mediante Servicios HTTP Angular

Migraciones, Seeders, Modelos y controladores

Equilibrio y sinergia en el modelo de negocio

¿Por qué Web y no Nativa?

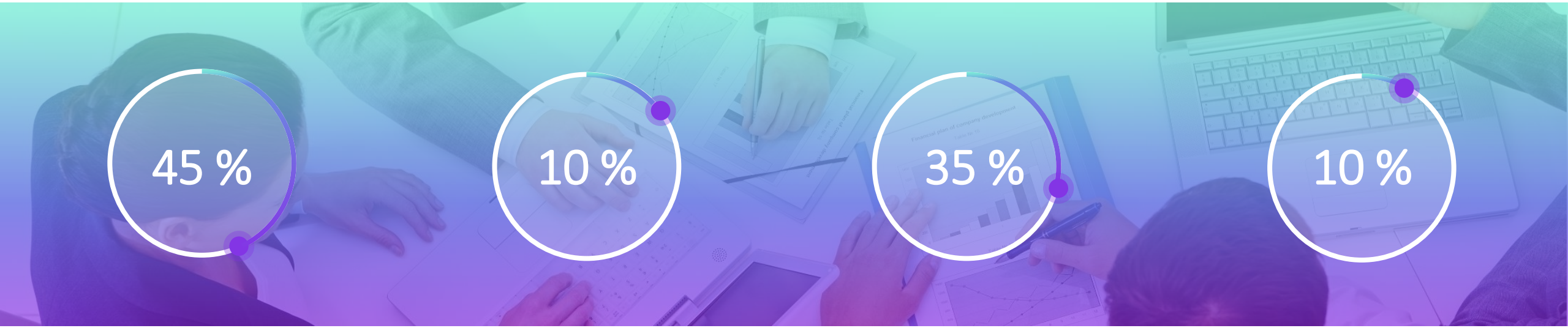
Veamos las principales ventajas de cada una de las modalidades a la hora de elegir una plataforma para desarrollar:

- ***Nativa***
- ***Híbrida***
- ***Web***

Debido a las siguientes ventajas y desventajas, resulta más apropiado el uso de las tecnologías híbridas y web con respecto a las nativas, ya que se adaptan mejor a las necesidades de la empresa

	Lenjuaje	Coste desarrollo	Interfaz usuario	Rendimiento	Multiplataforma	Tiempo desarrollo	App Stores
NATIVA	JAVA. -C, .NET	X	✓	✓	X	X	✓
HÍBRIDA	HTML, CSS, Javascript	–	✓	–	✓	–	✓
WEB	HTML, CSS, Javascript	✓	–	X	✓	✓	–

Ventajas de trabajar con Angular + Lumen y Visual Studio Code



Organización

La estructura de Angular y Lumen esta estandarizada. Cualquier programador que maneje dichas tecnologías es capaz de conocer su estructura

Código reducido

Ambos frameworks tratan de evitar la redundancia en el código. Esto es traduce en una aplicación ligera y veloz, con estructura SPA

Potencia y futuro

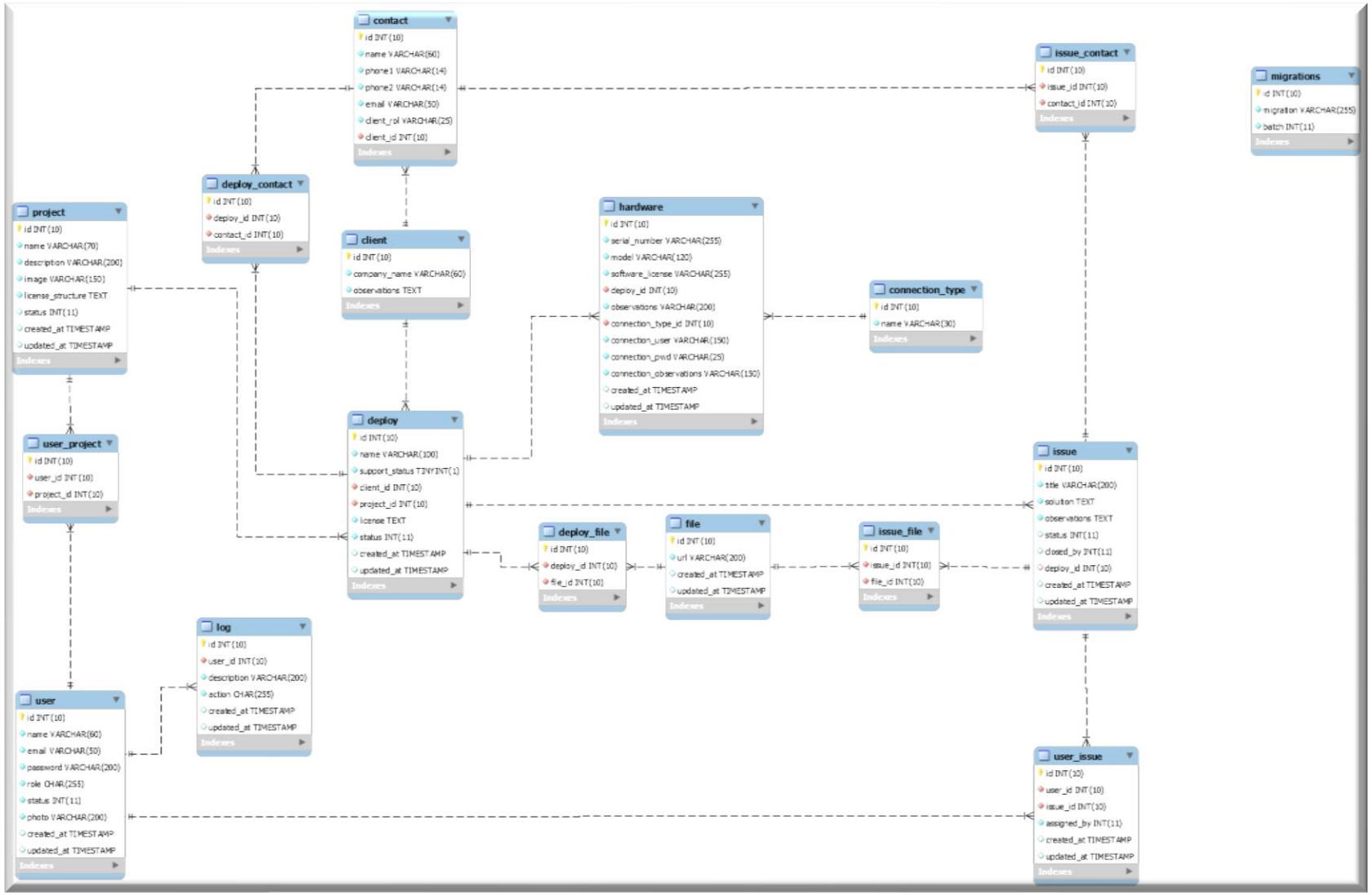
Ambos frameworks son versátiles y manejan multitud de funciones aplicables a las necesidades de programación actuales

Control de versiones

Visual Studio Code permite trabajar fácilmente con GIT y enlazar el proyecto con un repositorio remoto

MySQL

Utilizaremos MySQL como base de datos, ya que es la base de datos empleada en la empresa



Lumen 5.5

Es un micro-framework de Laravel, es decir
una versión reducida de este Framework PHP

¿Por qué Lumen y no Laravel?

ofrece menos características pero está optimizado para un rendimiento superior al eliminar características que no son esenciales.

Migraciones

Seeders

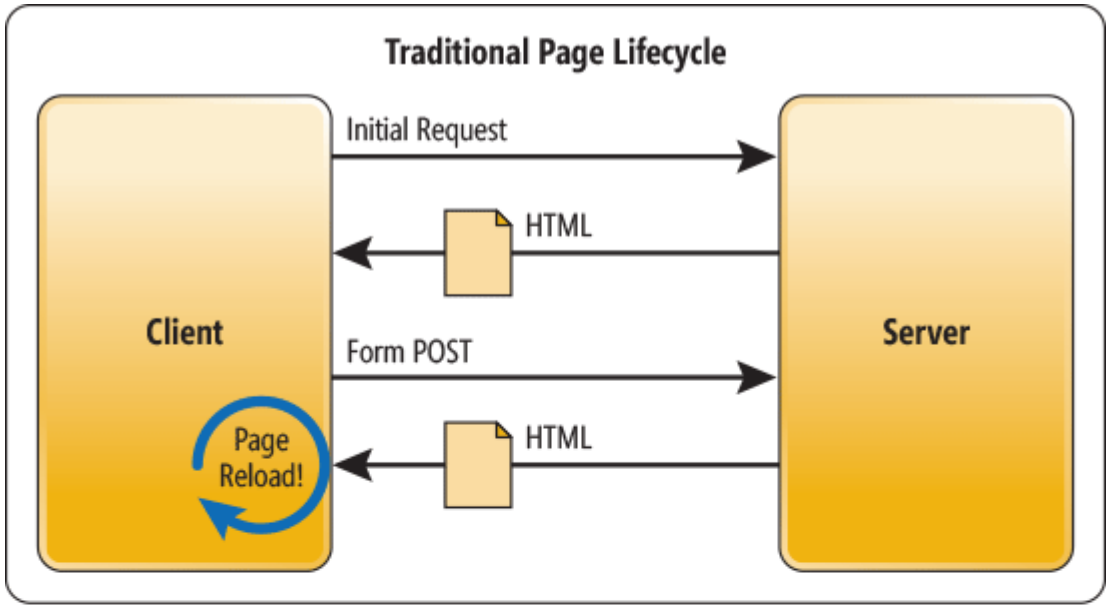
Modelos

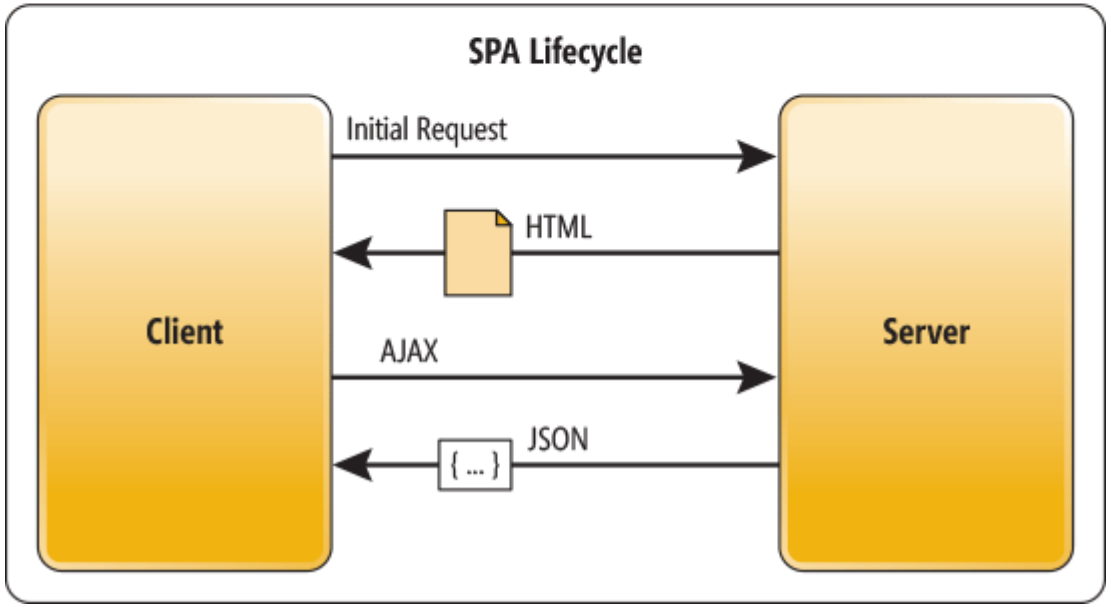
Controladores

Angular 7 y SPA

Angular es un *framework* JavaScript,
creado por Google y destinado
a facilitar la creación de aplicaciones web
modernas de tipo SPA (*Single Page Application*).

Su diseño SPA disminuye el número de ficheros requeridos para su funcionamiento y permite que la web sea asíncrona mediante peticiones AJAX



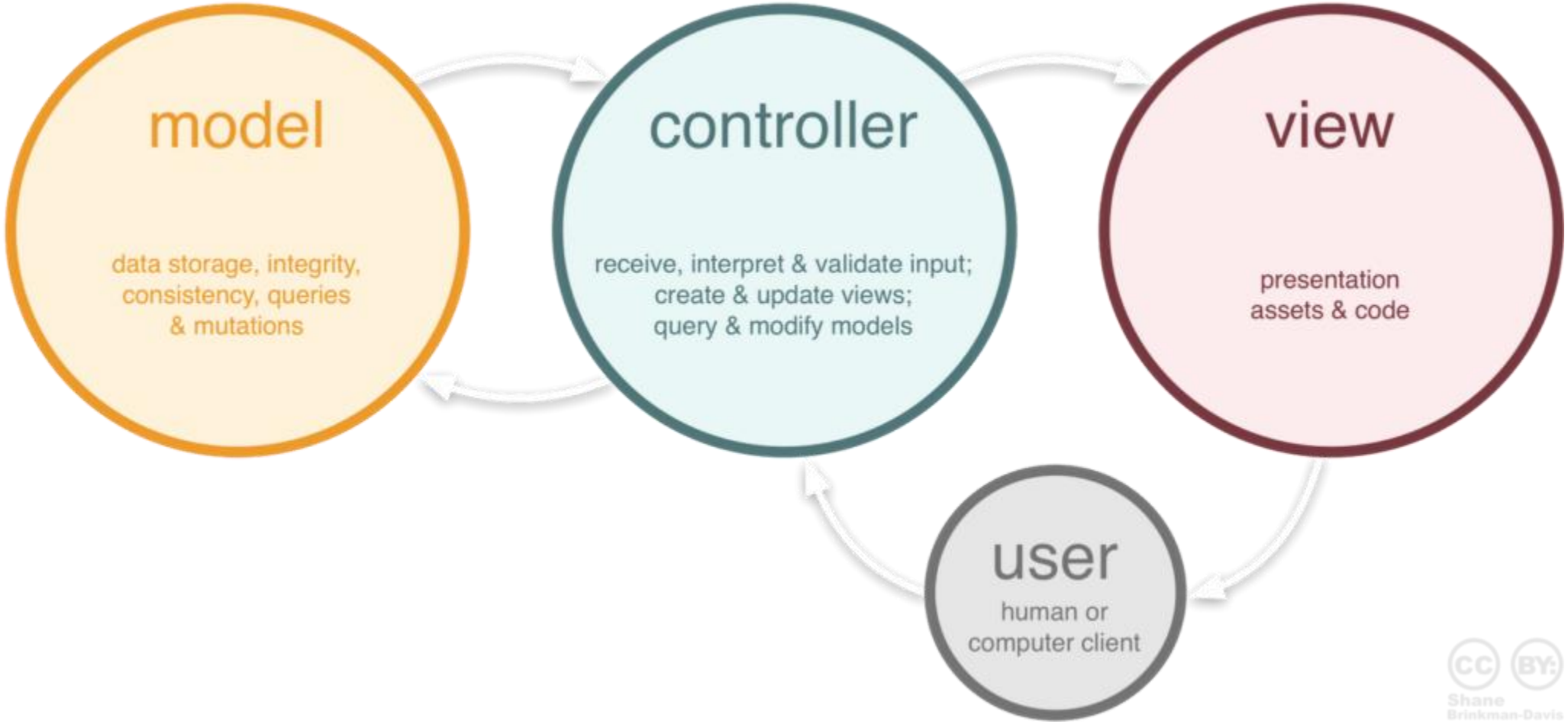


Migrando la base de datos e inyectando valores

Utilizaremos los ficheros 'Migrations' creados por nosotros Para desplegar las tablas en la base de datos, mediante el comando:

Creando modelos y controladores

Creamos los modelos de las tablas de la base de datos y los controladores necesarios para los CRUD de la API REST



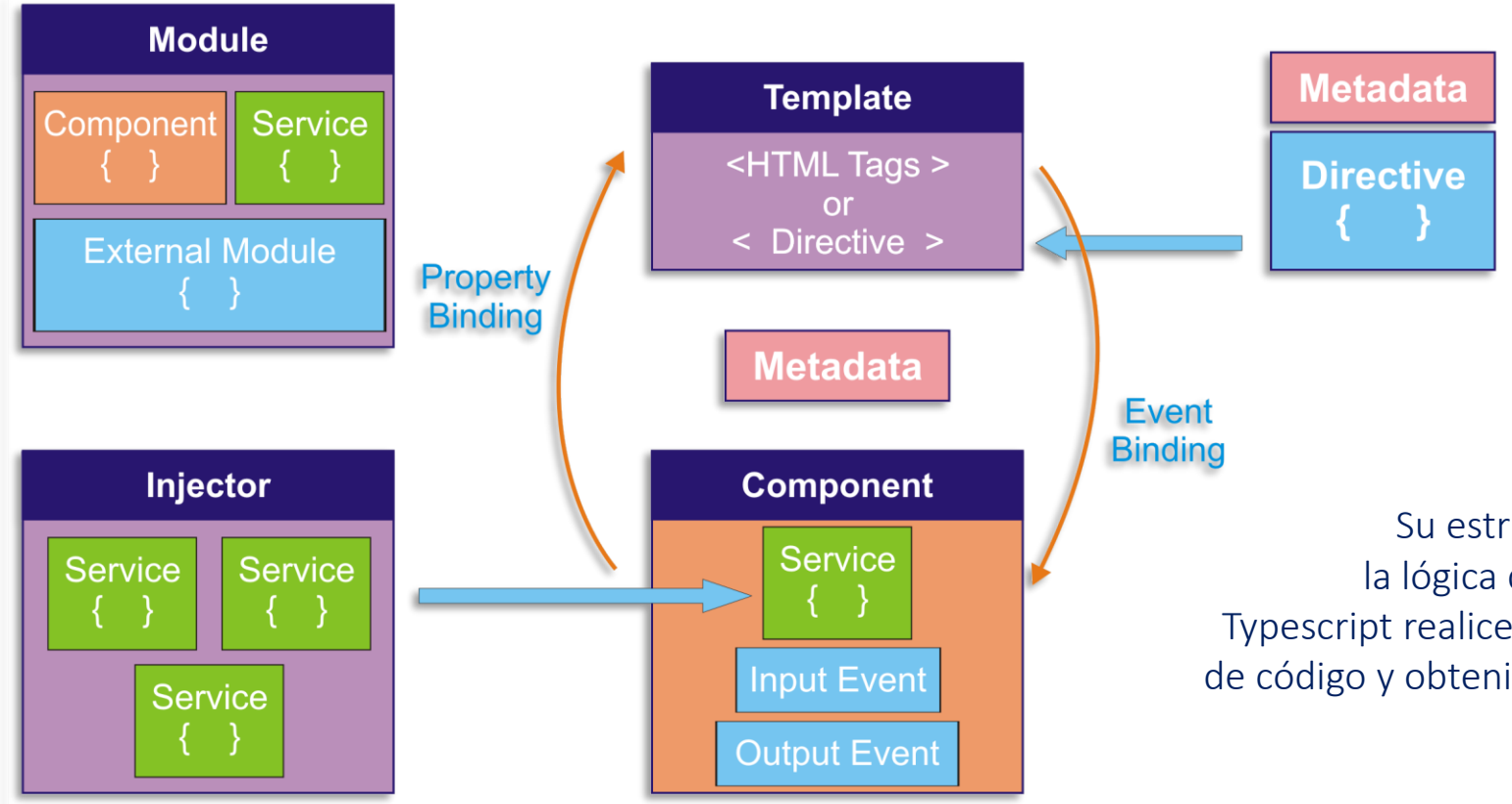
Probando las rutas de la API

Una vez definidos los modelos y desarrollado los controladores de nuestro ORM, definimos las rutas de la API en nuestro fichero 'web.php'

Emplearemos la extensión POSTMAN para realizar pruebas sobre la API que hemos creado para la aplicación, siguiendo las normas de uso estandarizadas y que se muestran en la imagen inferior.

GET	/movies	Get list of movies
GET	/movies/:id	Find a movie by its ID
POST	/movies	Create a new movie
PUT	/movies	Update an existing movie
DELETE	/movies	Delete an existing movie

Angular 7



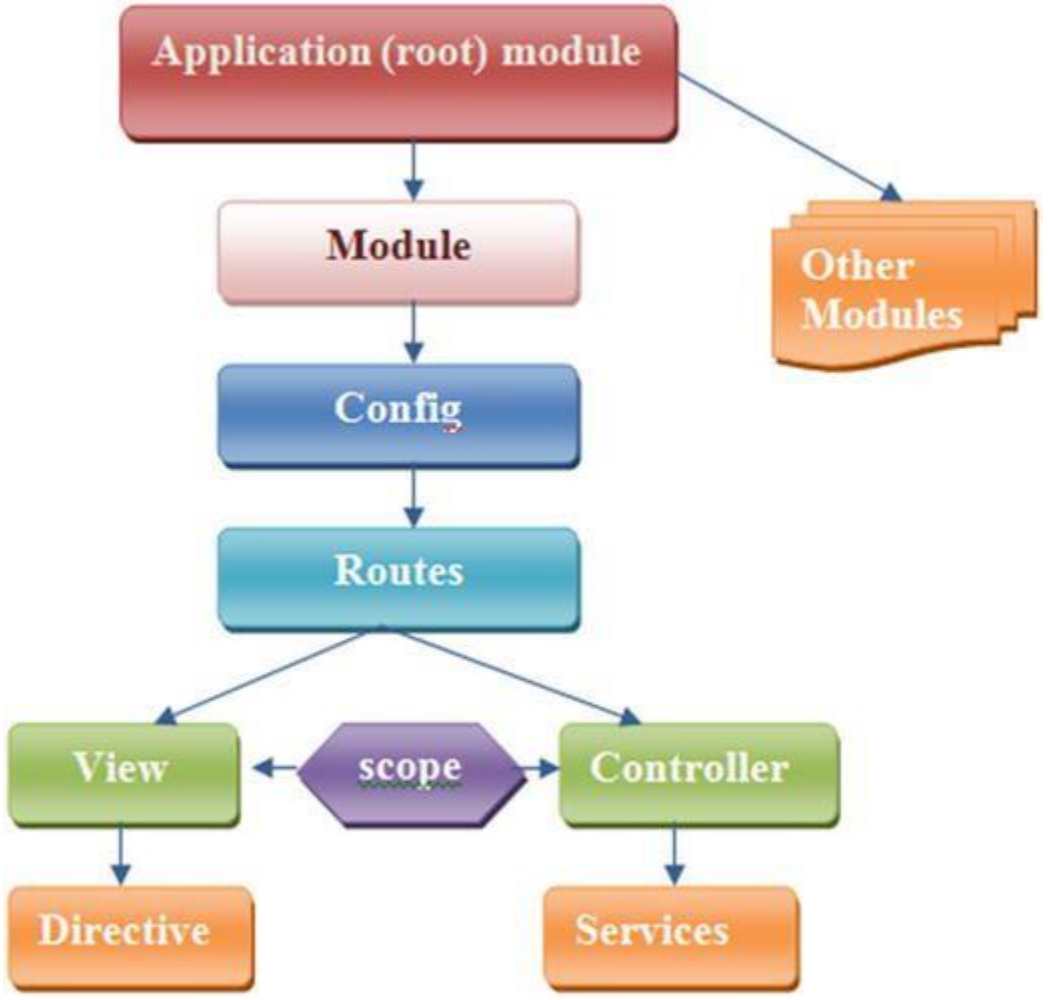
Su estructura diferenciada nos permite categorizar la lógica de la aplicación, haciendo que ciertas clases Typescript realicen tareas concretas, evitando la redundancia de código y obteniendo un código más ordenado y entendible

La estructura de Angular nos permite desarrollar una aplicación dinámica, modular y con amplias características y una gran posibilidad de crecimiento

Creación de Nuevos componentes y enrutado

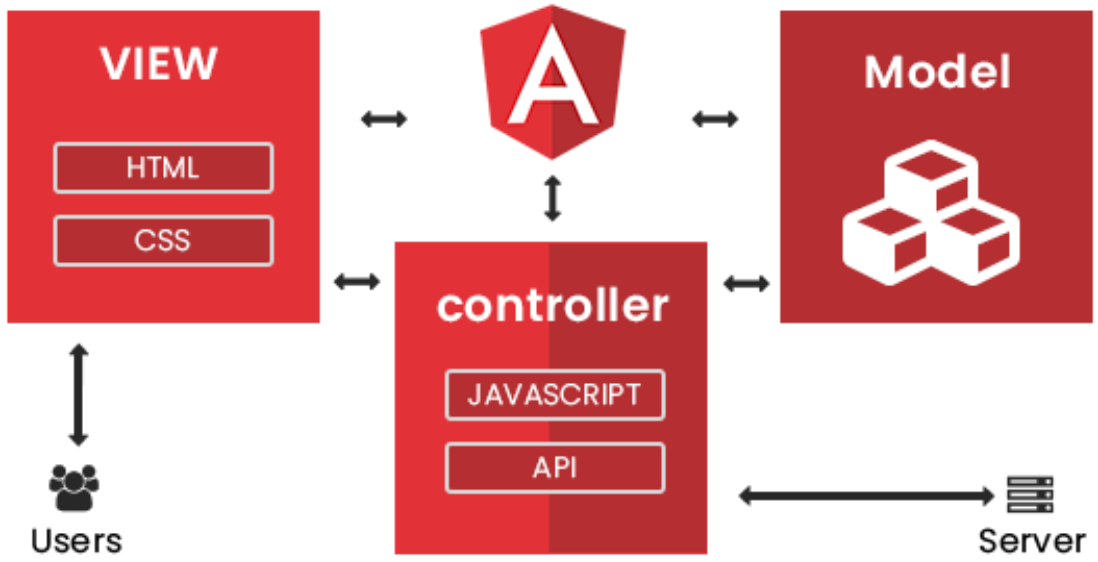
Separaremos la lógica de nuestra aplicación en distintos componentes o servicios. Cada componente realizará una tarea concreta en nuestra aplicación: mostrar un listado de incidencias, añadir una implantación, mostrar un mensaje de confirmación, cerrar la sesión, etc.

Para navegar entre la vista de los distintos componentes, emplearemos los módulos 'Router' y 'ActivateRouter', definiendo rutas (URL) dentro de la propia aplicación



Comunicación Vista – Modelo – Controlador

En nuestra aplicación seguimos el paradigma modelo-vista-controlador, en el cual separamos la lógica del componente de la interfaz, por imposición del propio Framework, así como el modelo de ambos, en el cual contiene los datos y la lógica de negocio de la aplicación.



Hallamos los tres tipos de ficheros muy diferenciados:

- Deploy.ts: Contiene las propiedades del objeto 'deploy' que referencia a la tabla 'deploy' de la base de datos
- add-deploy.component.ts: Contiene la lógica del componente que añade implantaciones mediante un método que realiza la petición POST
- add-deploy.component.html: Contiene la plantilla con los elementos de la interfaz necesarios para interactuar con la lógica y el usuario (formulario, títulos, etc.)

Servicio CRUD Angular

C Create

R Read

U Update

D Delete

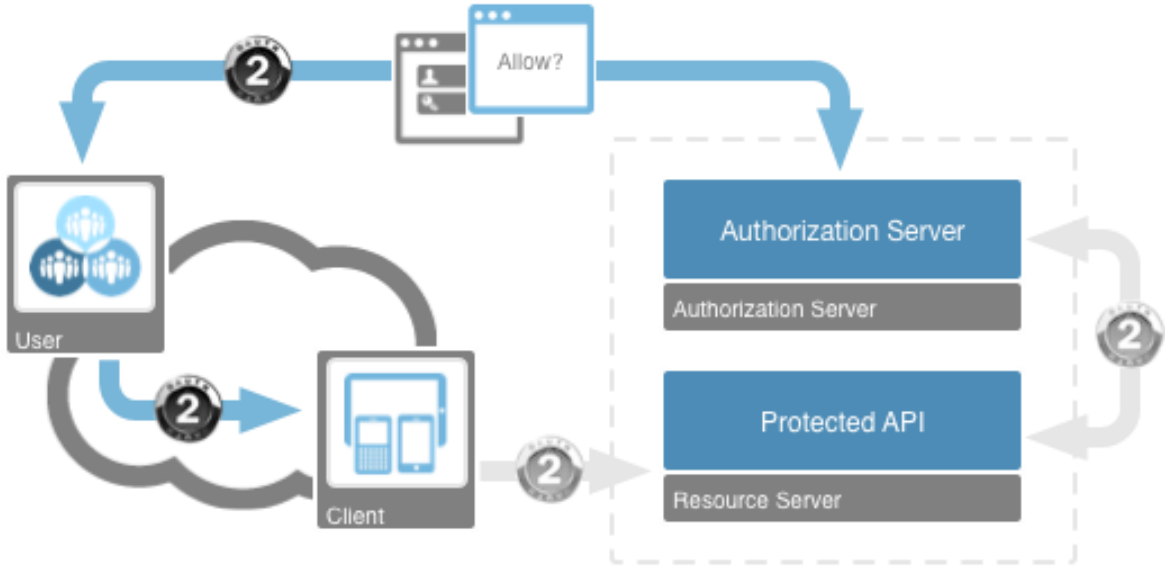
Realizaremos un CRUD de incidencias y un CRUD de implantaciones desde la aplicación, empleando un servicio Angular que posea los métodos HTTPClient, encargados de realizar las peticiones (Promesas) al Back-End

En cada componente encargado de realizar una tarea relacionada con el CRUD (añadir incidencia, listar detalles de implantación, etc.), haremos una llamada al método del servicio Angular mediante un método observable, que devuelva la respuesta de la petición en una función de CallBack

Validación de usuario: Componente y servicio de Autenticación

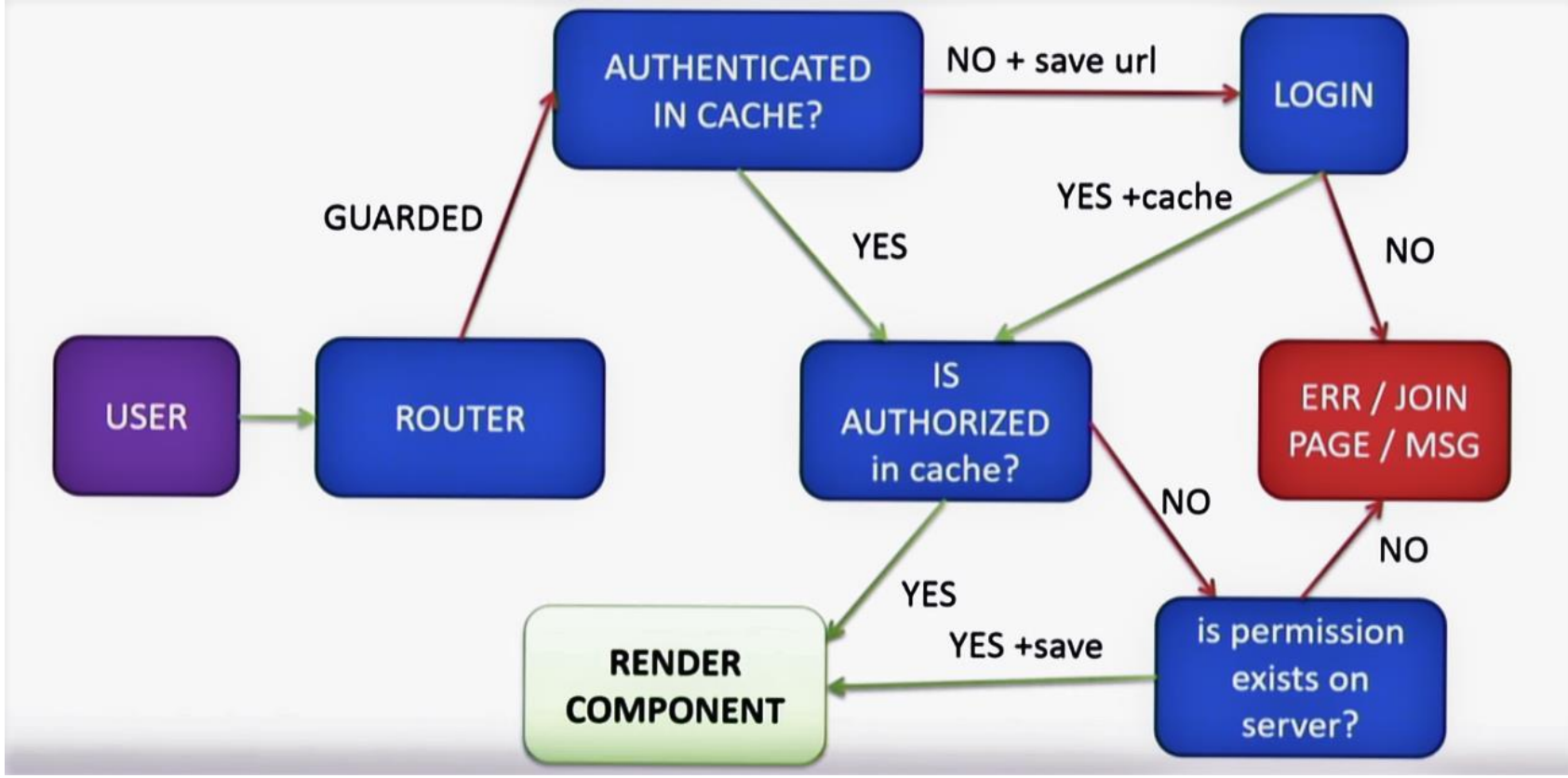
La autenticación se realiza mediante una combinación de técnicas del Front-End y el Back-End

1. Recogemos los datos de la interfaz de la aplicación, empleando directivas 'ngModel', realizando el llamado 'Binding'
2. Preparamos el envío hacia el servicio Angular de Autenticación desde el método observable del componente de Login
3. El servicio Angular realiza la petición POST al Back-End mediante una *promesa*.
4. El Back-End procesa la información. Valida que los datos sean correctos y, en caso afirmativo, genera el token de autenticación mediante la herramienta Oauth2
5. El resultado de la petición regresa al método del componente mediante una función de *Callback.* El token generado es recibido por Angular.

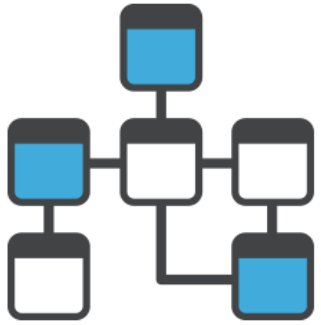


Roles de usuario y validación de la sesión por JWT y servicio Auth-guard

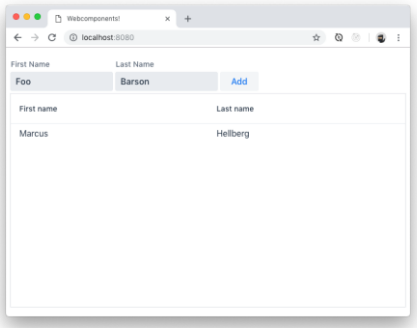
Contamos con un servicio adicional que comprueba que el usuario que ha iniciado sesión tiene permisos (está autorizado) para realizar las acciones que va a realizar dentro de la aplicación



CREATE, READ, UPDATE→ GET, POST, PUT



JSON Table
{“_id … }
{“_id … }
{“_id … }



SELECT

- Base de datos MySQl
- Clase Modelo del ORM

GET

- Clase Controlador del ORM
- Fichero de rutas de la API

LIST(){

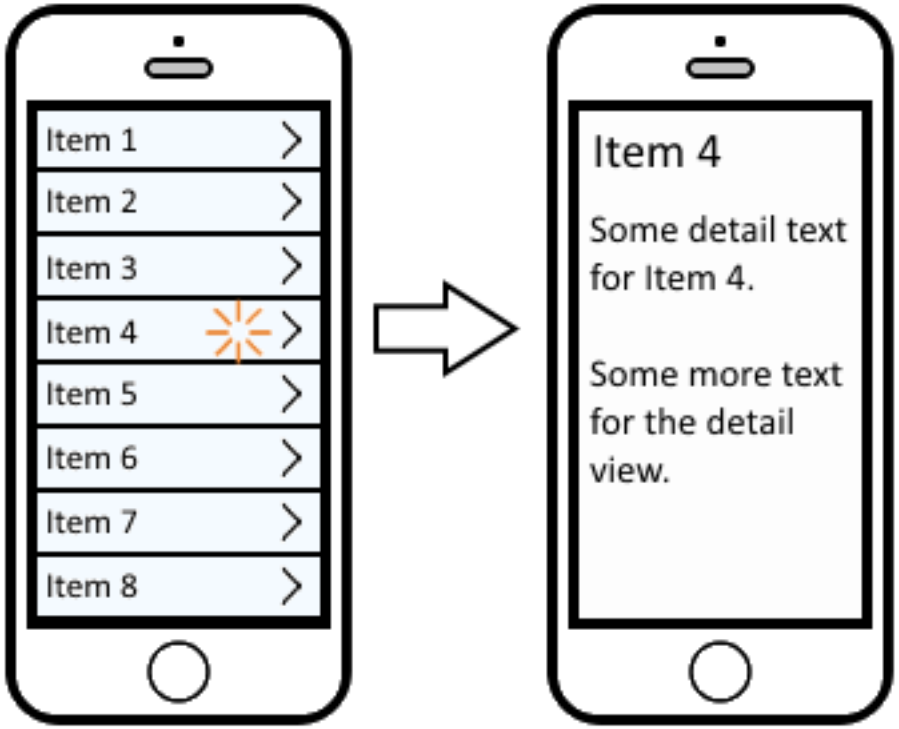
- Servicio Angular de Peticiones
- Componente de Listado

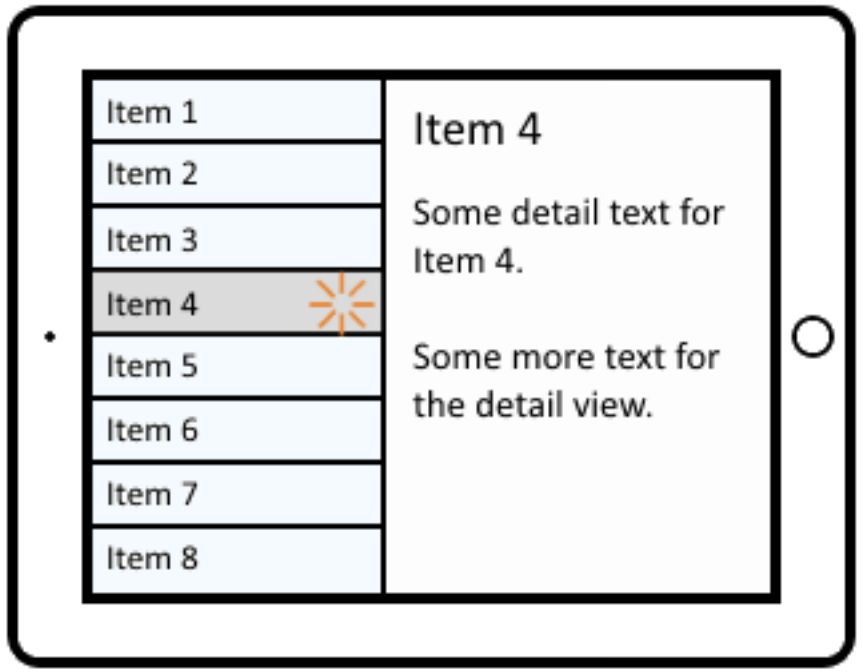
Maestro – Detalle y ‘ActivatedRouter’

Encontramos dos variantes de esta filosofía de diseño:

- ***En el menú lateral de filtrado de incidencias por proyecto***
- ***En la vista de detalles de las incidencias de la aplicación***

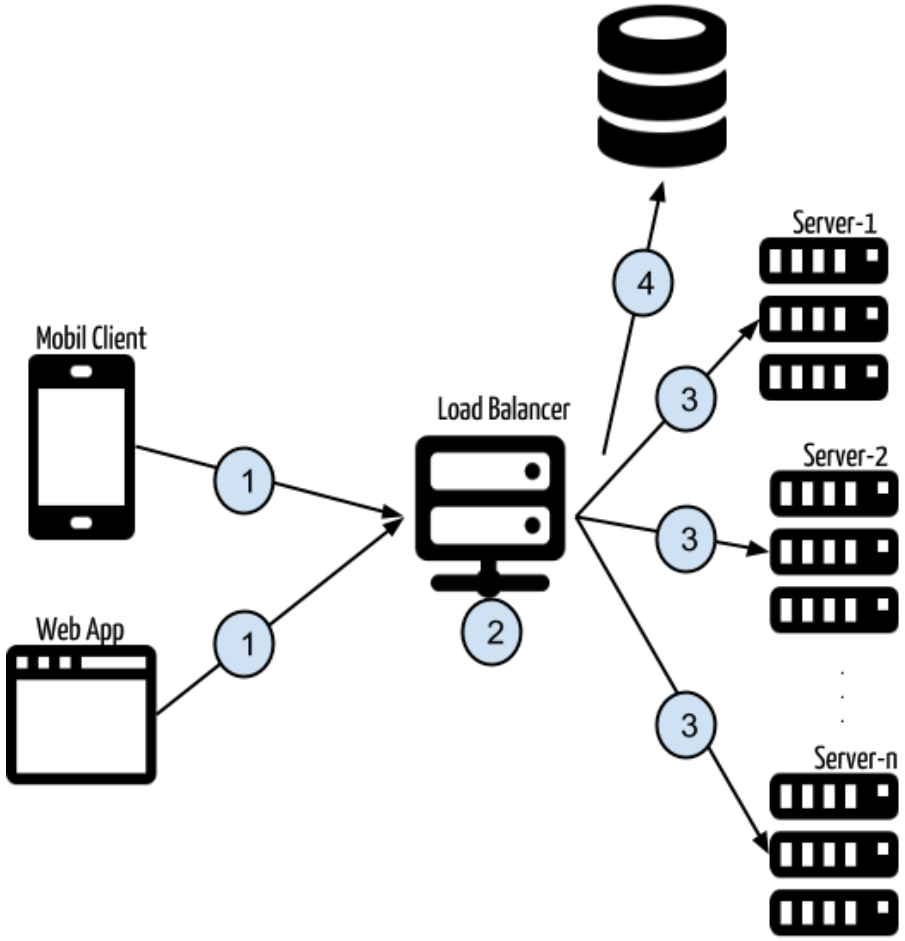
Usamos el módulo ActivateRoute para que la navegación entre los detalles de un objeto, conocido su ID, sea más sencillo.

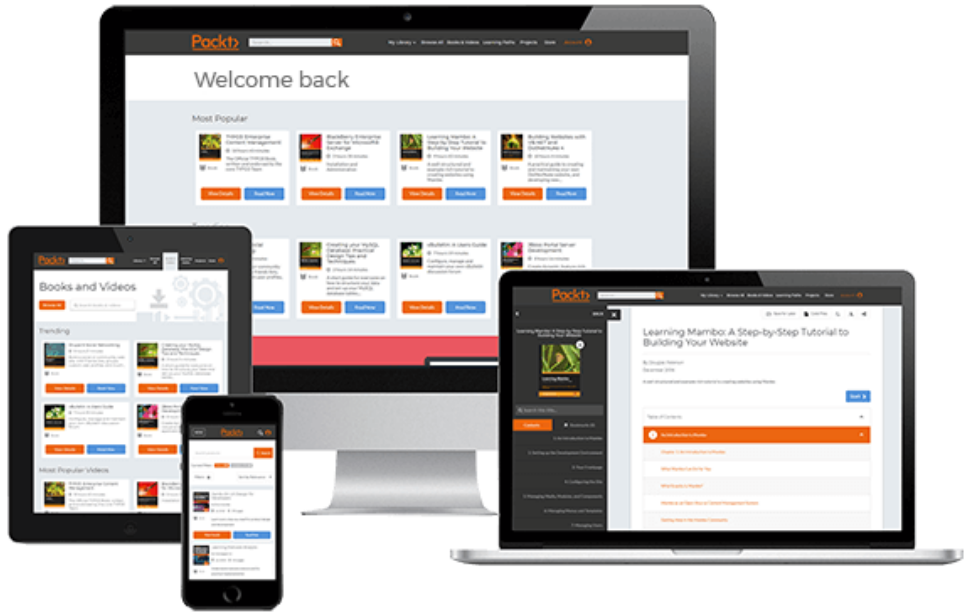




Limitaciones del proyecto

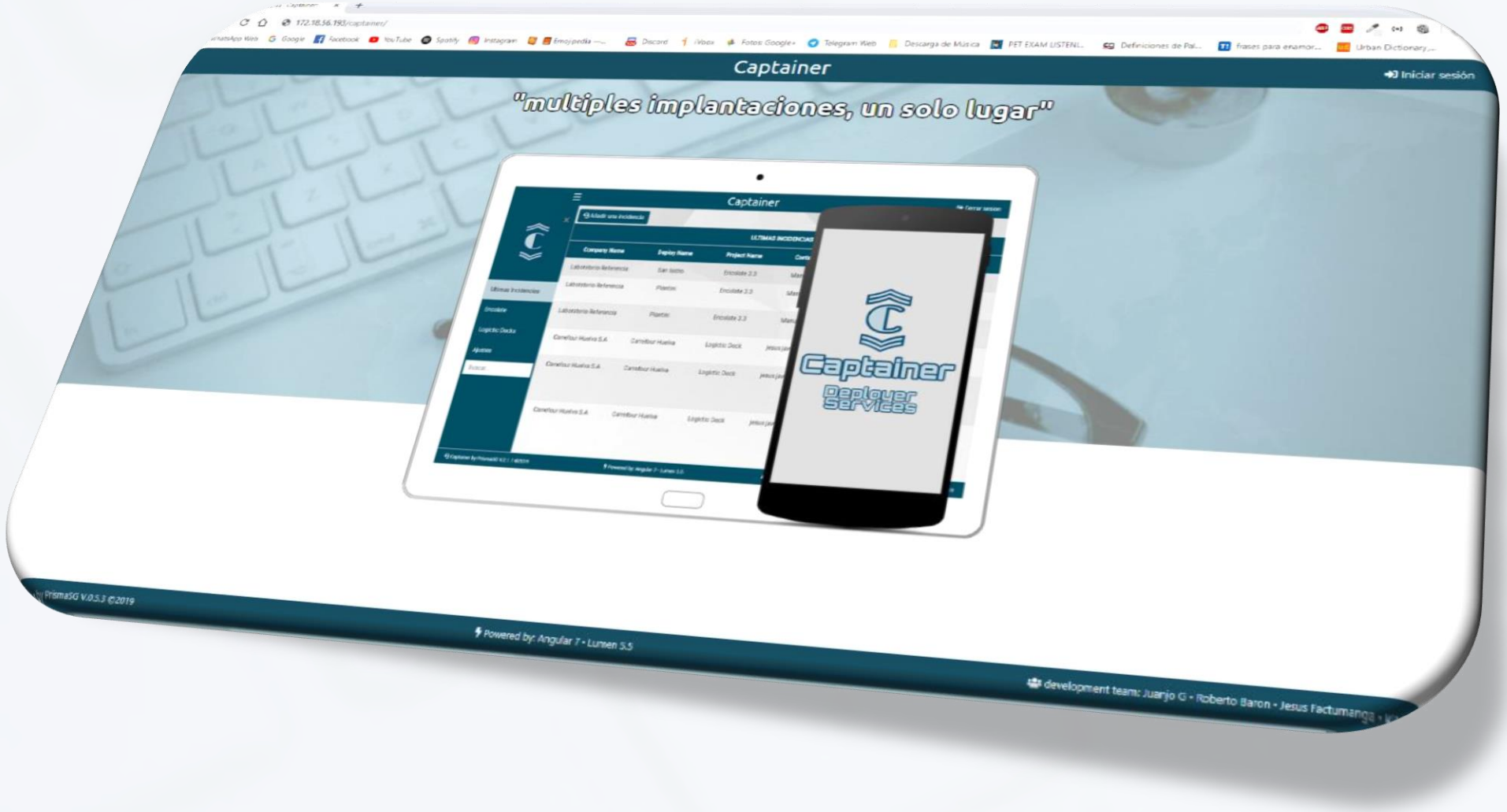
El tiempo de desarrollo total de la aplicación, con todas las correcciones de errores y pruebas de la aplicación que se requieren, excede el tiempo límite de desarrollo del proyecto





Prueba de la aplicación

Probamos la aplicación funcionando en un servidor remoto



¡Gracias!

No olviden formular sus preguntas ☺