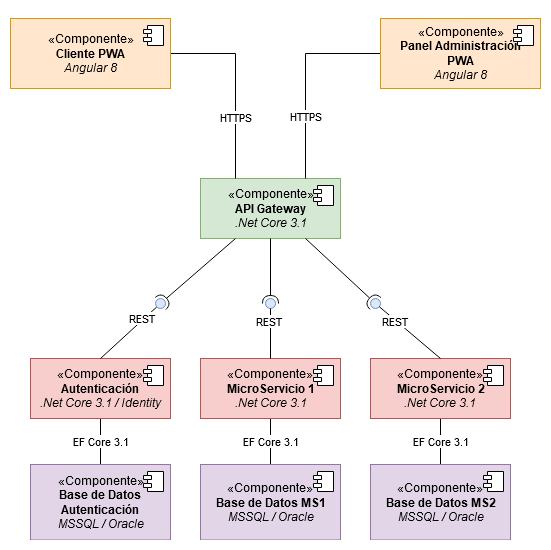
**Arquitectura Microservicios / PWA**

****

*Figura 1. Diagrama de componentes para arquitectura de microservicios*

* **PWA:** *Progressive Web Application* es un tipo de aplicación web que incluye funcionalidades de una aplicación móvil nativa como funcionalidad offline, push notifications y acceso a hardware del dispositivo.
* **HTTPS:** *HyperText Transfer Protocol Secure* es un protocolo seguro de intercambio de texto estructurado usado para la comunicación entre cliente y servidor y entre microservicios.
* **API Gateway:** Permite la integración de microservicios y la comunicación con los clientes a través de un *único* punto de acceso. Tiene conocimiento de todos los microservicios, pero no realiza ninguna operación CRUD, por lo tanto no tiene base de datos asignada.
* **REST:** *Representational State Transfer* es un estilo arquitectónico que define una serie de reglas para la comunicación de microservicios a través de una serie de operaciones *stateless* predefinidas por medio del protocolo *HTTP* y sus acciones.
* **Microservicio:** Componente autónomo e independiente que encapsula una parte del sistema y que debe ser tan pequeño y liviano como sea posible. Se comunica con otros microservicios o el API Gateway a través de REST.
* **EF Core:** ORM para .Net Core que es agnóstico a la base de datos utilizada.
* **Base de Datos:** Cada microservicio puede tener 0 o más bases de datos asociadas. Cada base de datos puede ser una tecnología diferente y debe contener solo los datos correspondientes a ese microservicio.

***Especificación REST***

Las acciones de HTTP usadas en REST son:

* C → Create → POST
* R → Read → GET
* U → Update → PUT
* D → Delete → DELETE

Se debe hacer uso de los códigos de estado definidos en HTTP para responder a las peticiones REST:

* 200 – OK – Operación correcta.
* 201 – OK – Nuevo recurso creado.
* 204 – OK – El recurso fue eliminado correctamente.
* 304 – Not Modified – El cliente puede usar datos cacheados.
* 400 – Bad Request – La petición es inválida o no puede ser procesada.
* 401 – Unauthorized – La petición requiere autenticación del cliente.
* 403 – Forbidden – El cliente no tiene permisos para acceder al recurso.
* 404 – Not found – El recurso no fue encontrado.
* 500 – Internal Server Error – Ocurrió un error en el servidor durante el procesamiento de la petición.

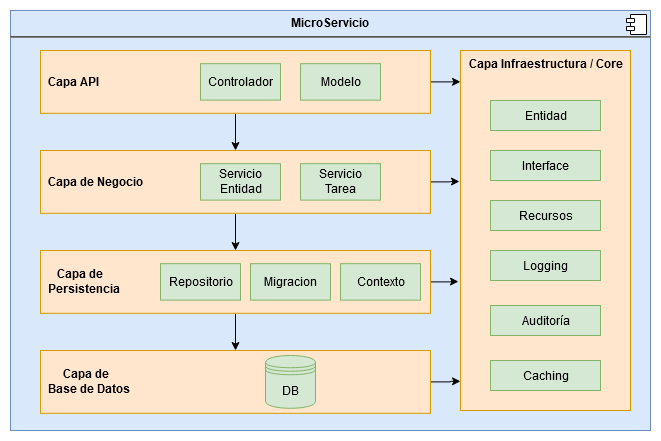
Las URIs (Identificador de Recursos Uniforme) deben seguir el siguiente formato:

**http://***[host]***:***[puerto]***/api/***{nombre del servicio}***/v***{número de versión}***/***{recurso}*

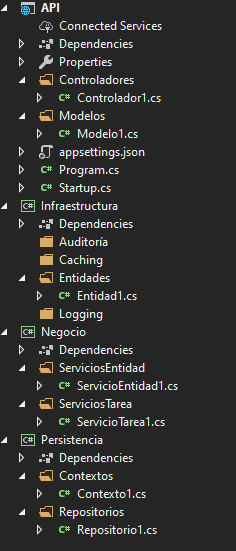
Más información:

*https://wiki.onap.org/display/DW/RESTful+API+Design+Specification*

***BackEnd***

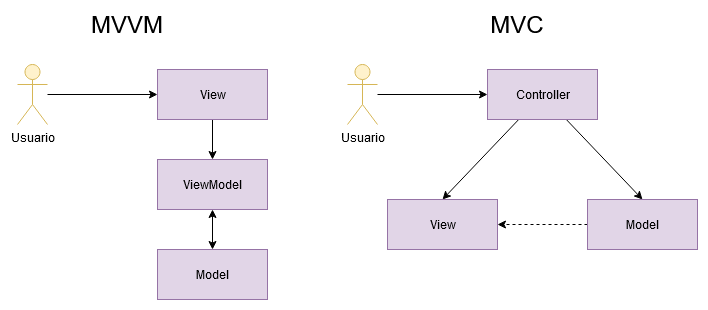
****

*Figura 2. Vista de capas de un microservicio*

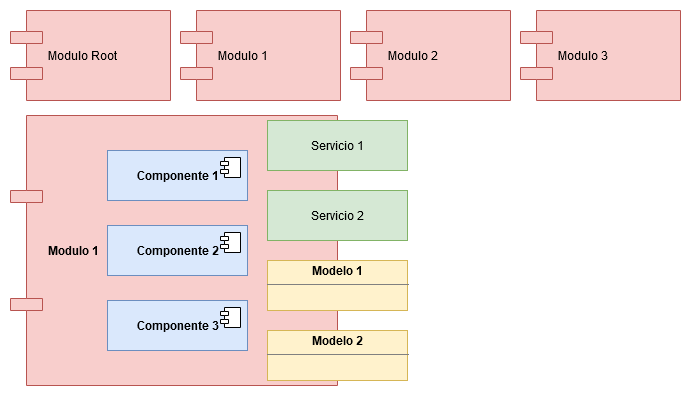


*Figura 3. Estructura de carpetas para un microservicio*

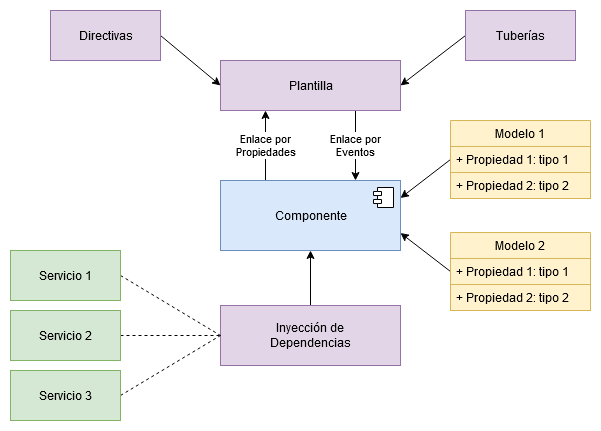
***FrontEnd***

******

*Figura 4. Patrón MVVM usado por Angular*

**

*Figura 5. Estructura de módulos y componentes de Angular*

**

*Figura 6. Interacción de componente con otros elementos de Angular*

* **Módulo:** Contexto de compilación que tiene un conjunto de componentes que pertenecen a un mismo flujo de trabajo o dominio de la aplicación . El módulo puede asociar otros elementos como servicios o modelos a sus componentes.
* **Componente:** Clase que contiene datos y lógica de una funcionalidad y que se asocia con una plantilla.
* **Plantilla:** Archivo HTML que define una vista y que se comunica con el componente para el enlace de datos. La plantilla puede hacer uso de directivas y tuberías para el procesamiento de datos directamente en la plantilla.
* **Tubería:** Permite transformar valores en el momento de visualización.
* **Directiva:** Provee lógica que puede ser usada directamente en la plantilla sin hacer uso del componente.
* **Enlace por propiedades:** Permite a la plantilla acceder a los datos que contiene el componente.
* **Enlace por eventos:** Permite al componente realizar alguna operación cuando una interacción o evento ocurre en la vista.
* **Modelo:** Clase que abstrae la estructura y atributos de un objeto real.
* **Servicio:** Clase que define un conjunto de operaciones de negocio y que pueden ser usados por los componentes o por otros servicios. Deberían encargarse de realizar las peticiones HTTP a el servidor.
* **Inyección de dependencias:** Técnica implementada en Angular que permite a los componentes hacer uso de servicios proporcionados por un módulo sin tener que crear su propia instancia. Permite crear servicios singleton.

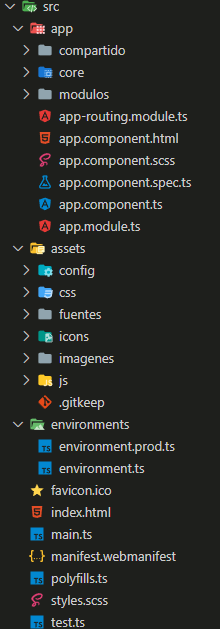
***Estructura de Carpetas***

Ya que Angular no tiene una estructura definida para las carpetas se recomienda el uso de 3 carpetas principales dentro de la carpeta App.

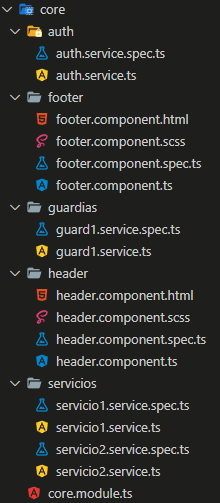
* **Core:** En esta se encontrará el módulo Core. El módulo core solo debe ser instanciado una vez y debe contener los singleton de la aplicación (servicios, guardias, autenticación) y componentes que conforman la base de la aplicación (header, footer, login).
* **Compartido:** En esta carpeta se encontrará el módulo Compartido. Este módulo contiene los elementos que pueden ser reutilizados por otros módulos, componentes comunes, tuberías, directivas y modelos.
* **Módulos:** En esta carpeta se ubicarán todos los otros módulos de la aplicación. Cada módulo debe tener su propio routing y debe ser cargado a través de *lazy-loading*. Los módulos deben contener las carpetas *páginas* y *componentes.* La carpeta páginas contiene componentes que actuarán como vistas y que serán asignados a una URL específica, por ejemplo, Inicio, Contacto, etc. La carpeta componentes contiene componentes que serán usados en las páginas y que no tienen una URL asignada a ellos, por ejemplo, Formulario, Modal, etc.

La carpeta assets tiene los archivos estáticos que serán usados por la aplicación una vez sea compilada.

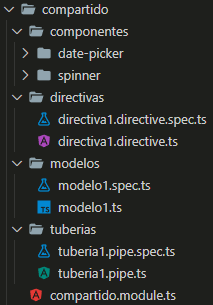
La carpeta environments contiene configuración de la aplicación que depende del ambiente en el que se ejecuta la aplicación (desarrollo, producción).



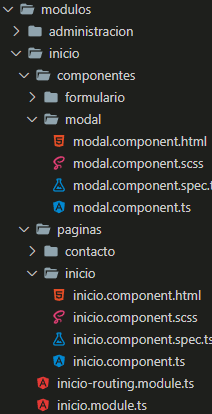
*Figura 6. Estructura general de PWA en Angular*



*Figura 7. Estructura de carpetas del módulo core*



*Figura 8. Estructura de carpetas del módulo compartido*



*Figura 7. Estructura de carpetas para los otros módulos de la aplicación*

***Especificación PWA***

Una PWA debe ser:

* **Progresiva:** Funcionar para cualquier usuario, sin importar el navegador que utilice.
* **Responsive:** Ajustarse a cualquier tipo de dispositivo, escritorio, tablet, celular, televisor, etc.
* **Independiente de conectividad:** Tener funcionalidad offline o en redes de baja velocidad.
* **App-like:** Interacción y navegación similar a un app nativa.
* **Segura:** Servida a través de HTTPS para la protección de datos.
* **Descubrible:** Identificables como aplicaciones a través de manifiestos y *service workers* para que los motores de búsqueda puedan encontrarla.
* **Reenganchable:** Facilitar el retorno del usuario a la app a través de *push notifications.*
* **Instalable:** Permitir al usuario *guardar* la aplicación en su pantalla de inicio sin necesidad del uso de una *app store*.
* **Enlazable:** Fácilmente compartible a través de una URL.

Más información de PWA en Angular : <https://angular.io/guide/service-worker-intro>

***Requerimientos No Funcionales***

|  |  |
| --- | --- |
| **ID** | **REQUERIMIENTO** |
| 1 | El sistema debe ser desarrollado usando la técnica de diseño web Responsive y debe ser una PWA. |
| 2 | Los campos obligatorios deben estar plenamente identificados. |
| 3 | Todos los botones de acción deben tener un mensaje de respuesta positivo o negativo. |
| 4 | Todos los botones de acción deben tener un mensaje de tipo tooltip. Debe quedar en archivo para multiidioma y también ser configurado. |
| 5 | Todas las pantallas que carguen información a grillas deben tener paginación. (Documento de desarrollo Contar ERP Web ver 1.1) |
| 6 | La aplicación debe tener la capacidad de Multiidioma. |
| 7 | Hacer validación de campos obligatorios a la medida en que se van digitando. |
| 8 | Descripción por cada uno de los campos directamente en el formulario. |
| 9 | Autosalvar (No en base de datos, debe ser en cache de cliente) |
| 10 | Notificaciones por correo y por pantalla (no destruir con tiempo) |
| 11 | Autenticación a través de usuario y contraseña. |
| 12 | El sistema debe ser desarrollado orientado a pruebas automatizadas. (ppt de selección de proveedores) |
| 13 | Autorización basada en roles o perfiles. |
| 14 | Auditoria y Log a nivel de usuario. |
| 15 | Auditoria y log a nivel de datos. (Documento de desarrollo Contar ERP Web ver 1.1) |
| 16 | Auditoria y log a nivel de error. |
| 17 | Pantalla de visualización de log de auditoria. |
| 18 | Manejo de Procesos Largos, Procesos Concurrentes y límite de ejecución de procesos. |
| 19 | Que exista en la arquitectura de la aplicación uno o varios servicios de uso general para funciones que aplican en distintas partes de la aplicación. (Singleton, inyeccion de dependencias) |
| 20 | Documentación y estándares. (Uso de swagger o herraminetas similares) |
| 21 | Manejo de transacciones. |
| 22 | Manejo de Cache y optimización de componentes para mejor desempeño. |
| 23 | Manejo de Errores Try Catch. |
| 24 | Manejo de Clases Base. |