UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN INGENIERÍA DEL SOFTWARE PARA LA WEB

Trabajo Fin de Máster

REST API PLATAFORMA COLABORATIVA

Isaac Fernández Alonso

2020

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN

INGENIERÍA DEL SOFTWARE PARA LA WEB

Trabajo Fin de Máster

"REST API PLATAFORMA COLABORATIVA"

Autor: **Isaac Fernández Alonso**

Director: **Jose Javier Martínez Herráiz**

Tribunal:

Presidente: .....

Vocal 1º: ..... Vocal 2º: .....

Calificación: .....

Fecha: ..... de ..... de .....

Líneas de agradecimiento, si se quieren poner.

ÍNDICE RESUMIDO

ÍNDICE DETALLADO

INTRODUCCIÓN

Compartir datos entre dos o más sistemas se ha convertido en un requisito casi fundamental del desarrollo de software, y para ello, han ido surgiendo varios estándares a lo largo del tiempo, tales como CORBA, SOAP, … que habitualmente establecen unas reglas estrictas para los mensajes.

REST no es un estándar, es un estilo de arquitectura de software que se utiliza para describir cualquier interfaz entre diferentes sistemas que utilicen Hipermedia (conjunto de métodos o procedimientos para escribir, diseñar o componer contenidos que integren soportes tales como: texto, imagen, video, …) para comunicarse. Este término proviene de las siglas en inglés *REpresentational State Transfer* (transferencia de estado representacional) y se originó en el año 2000 a partir de una tesis doctoral. Y si bien en principio se refería a ese estilo de arquitectura, en la actualidad ha evolucionado hacia un sentido más amplio, y podemos decir que describe cualquier interfaz que utilice directamente HTTP

De forma muy resumida, lo que podemos de con REST es que entre dos llamadas cualquiera, el servicio no guarda los datos. Por ejemplo, podemos autenticar a un usuario con su email y contraseña en una llamada, pero la siguiente que hagamos ya se habrá olvidado de la anterior petición de autenticación.

El cliente de una API REST puede ser una aplicación Android o iOS o un navegador web (algo que ya esperábamos que utilizase HTTP), pero también puede ser "un Alexa", un "Google Home" o incluso una lavadora, maquinaria industrial (IoT), …

Con el presente trabajo, lo que se pretende es dar una visión global (y pedagógica) de la flexibilidad y capacidad de uso de una API REST .

OBJETIVOS DEL PROYECTO

El objetivo principal del presente trabajo consiste en dar una visión global (y pedagógica) de la flexibilidad y capacidad de uso de una API REST en este caso a través de una plataforma colaborativa de técnicos asociados a una tecnología, con disponibilidad de un horario, y cuyos servicios podrán ser contratados/reservados a través de esta REST API.

Que una empresa ofrezca una API REST va a enriquecer su negocio cuando otras empresas generen productos y servicios que se apoyen en los ofrecidos por la empresa original. Como ejemplo, la existencia de las APIs de Twitter permite a otras empresas crear productos y negocios que se apoyan en Twitter (Hootsuite es un servicio para gestionar las redes sociales que pueda utilizar una empresa y una de las redes sociales que puede gestionar es Twitter haciendo uso de sus APIs).

Tanto ampliar el ámbito de negocio, como llegar más lejos, es algo que una empresa no siempre puede alcanzar sola. Creer que una puede llegar a todos los potenciales clientes a través de las plataformas y dispositivos existentes es una idea bastante utópica, sin embargo si se puede crear una API REST que permita encontrar aliados que sí lo hagan (o al menos expandan esas posibilidades) y que además sea en beneficio mutuo (empresas como Dropbox, eBay, Twitter, Instagram, Facebook lo saben bien y una parte muy importante de sus usuarios no son directos, sino que les llegan a través de otras aplicaciones que usan sus respectivas APIs).

ESTADO DEL ARTE

3.1 ¿Qué es REST API?

El término REST (Representational State Transfer, en español "transferencia de estado representacional") se acuñó en el año 2000 por Roy Fielding (uno de los padres de la especificación del protocolo HTTP) en su tesis doctoral y se refería originalmente a un conjunto de principios de arquitectura. En la actualidad se usa en el sentido más amplio para describir cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML, JSON, etc) sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes, como por ejemplo SOAP. Es posible diseñar sistemas de servicios web de acuerdo con el estilo arquitectural REST de Fielding y también es posible diseñar interfaces XMLHTTP de acuerdo con el estilo de llamada a procedimiento remoto (RPC), pero sin usar SOAP. Estos dos usos diferentes del término REST causan cierta confusión en las discusiones técnicas, aunque RPC no es un ejemplo de REST.

Según Fielding las restricciones que definen a un sistema RESTful serían:

- **Cliente-servidor**: esta restricción mantiene al cliente y al servidor débilmente acoplados. Esto quiere decir que el cliente no necesita conocer los detalles de implementación del servidor y el servidor se "despreocupa" de cómo son usados los datos que envía al cliente. Cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Los recursos (elementos de información) cobran importancia en este sentido ya que pueden ser accedidos utilizando un identificador global (un Identificador Uniforme de Recurso). Para manipular estos recursos, los componentes de la red (clientes y servidores) se comunican a través de una interfaz estándar (HTTP) e intercambian representaciones de estos recursos (los ficheros que se descargan y se envían). La petición puede ser transmitida por cualquier número de conectores (por ejemplo: clientes, servidores, cachés, túneles, etc.) pero cada uno lo hace sin "ver más allá" de su propia petición. Así, una aplicación puede interactuar con un recurso conociendo el identificador del recurso y la acción requerida, no necesitando conocer si existen cachés, proxys, cortafuegos, túneles o cualquier otra cosa entre ella y el servidor que guarda la información.

- **Sin estado**: aquí decimos que cada petición que recibe el servidor debería ser independiente, es decir, no es necesario mantener sesiones. El contenido de los clientes no se almacena en el servidor entre las solicitudes (en todo caso la información sobre el estado de la sesión se queda en el cliente).

- **Cacheable**: debe admitir un sistema de almacenamiento en caché. La infraestructura de red debe soportar una caché de varios niveles. Este almacenamiento evitará repetir varias conexiones entre el servidor y el cliente para recuperar un mismo recurso.

- **Interfaz uniforme**: define una interfaz genérica para administrar cada interacción que se produzca entre el cliente y el servidor de manera uniforme, lo cual simplifica y separa la arquitectura. Esta restricción indica que cada recurso del servicio REST debe tener una única dirección, "URI". Esta limitación es fundamental para el diseño de las API de RESTful e incluye 4 aspectos:

    - Identificación de recursos en las solicitudes: los recursos se identifican en las solicitudes y se separan de las representaciones que se devuelven al cliente.

- Administración de recursos mediante representaciones: los clientes reciben archivos que representan los recursos. Estas representaciones deben tener la información suficiente como para poder ser modificadas o eliminadas.

- Mensajes autodescriptivos: cada mensaje que se devuelve al cliente contiene la información suficiente para describir cómo debe procesar la información.

- Hipermedios es el motor del estado de la aplicación: después de acceder a un recurso, el cliente REST debe ser capaz de descubrir mediante hipervínculos todas las otras acciones que están disponibles actualmente.

- **Sistema de capas**: Las interacciones cliente-servidor pueden estar mediadas por capas adicionales, que pueden ofrecer otras funciones. Esto ayuda a mejorar la escalabilidad, el rendimiento y la seguridad.

En cuanto al concepto de API (Interfaz de Programación de Aplicaciones) surgió en los primeros días de la informática, mucho antes que la computadora personal. En esa época, una API normalmente se usaba como biblioteca para los sistemas operativos. Casi siempre estaban habilitadas localmente en los sistemas en los que operaban, aunque a veces pasaban mensajes entre las computadoras centrales. Después de casi 30 años, las API se expandieron más allá de los entornos locales. A principios del año 2000, ya eran una tecnología importante para la integración remota de datos.

Hoy en día API representa el conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones y permiten que productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados. Esto simplifica el desarrollo de las aplicaciones y permite ahorrar tiempo y dinero. Las API le otorgan flexibilidad; simplifican el diseño, la administración y el uso de las aplicaciones, y proporcionan oportunidades de innovación, lo cual es ideal al momento de diseñar herramientas y productos nuevos (o de gestionar los actuales).

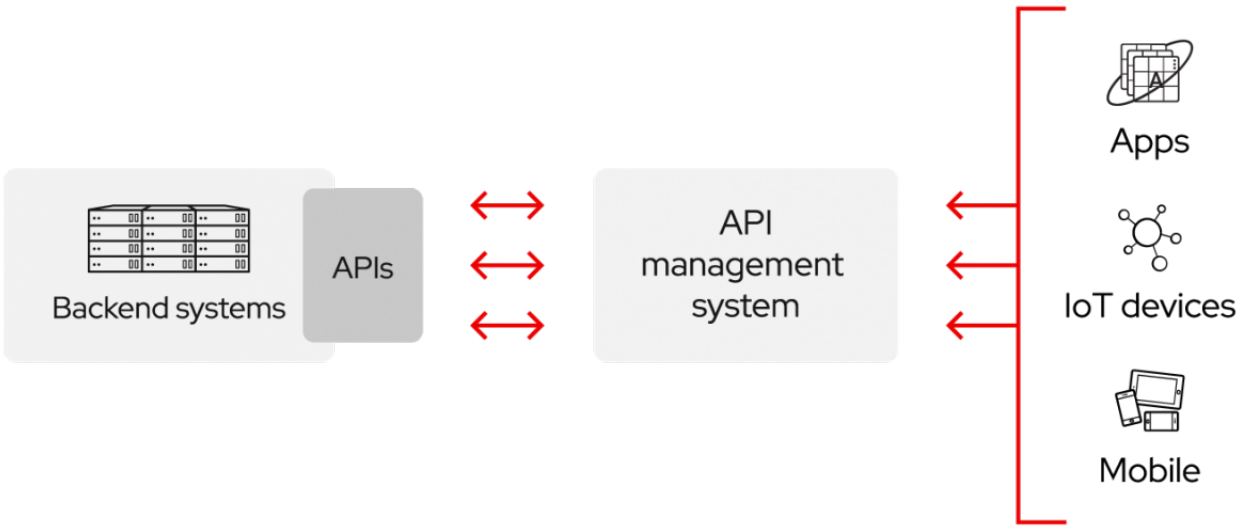


Fig. 1: Descripción de API

3.2 WebService (SOAP) vs REST API

A medida que se han difundido las API, se desarrolló una especificación de protocolo para permitir la estandarización del intercambio de información; se llama Protocolo de Acceso a Objetos Simples, más conocido como SOAP. Las API diseñadas con SOAP usan XML para el formato de sus mensajes y reciben solicitudes a través de HTTP o SMTP. Con SOAP, es más fácil que las aplicaciones que funcionan en entornos distintos o están escritas en diferentes lenguajes compartan información.

Las API web que funcionan con las limitaciones de arquitectura REST se llaman API de RESTful. La diferencia entre REST y SOAP es básica: SOAP es un protocolo, mientras que REST es un estilo de arquitectura. Esto significa que no hay ningún estándar oficial para las API web de REST (tal como se define en la tesis de Roy Fielding), las API son RESTful siempre que cumplan con las 6 limitaciones principales de un sistema REST (arquitectura cliente-servidor, sin estado, capacidad de caché, sistema en capas, código de demanda (opcional), e interfaz uniforme).

Estas limitaciones pueden parecer demasiadas, pero son mucho más sencillas que un protocolo definido previamente. Por eso, las API de REST son cada vez más frecuentes que las de SOAP.

En los últimos años, la especificación de OpenAPI se ha convertido en un estándar común para definir las API de REST. OpenAPI establece una forma independiente del lenguaje para que los desarrolladores diseñen interfaces API de REST, que permite a los usuarios entenderlas con el mínimo esfuerzo.

Las arquitecturas de microservicios se parecen a los patrones SOA en que los servicios son especializados y no tienen conexión directa. Pero además, descomponen las arquitecturas tradicionales en partes más pequeñas. Los servicios de la arquitectura de microservicios usan un marco de mensajería común, como las API de REST. Utilizan API de REST para comunicarse entre sí, sin necesidad de operaciones complejas de conversión de datos ni capas de integración adicionales. Usar las API de REST permite e incluso fomenta una distribución más rápida de nuevas funciones y actualizaciones. Cada servicio es independiente. Un servicio se puede reemplazar, mejorar o abandonar, sin afectar los demás servicios de la arquitectura. Esta arquitectura liviana optimiza los recursos distribuidos o en la nube y admite la escalabilidad dinámica de los servicios individuales.

3.3 Ventajas de REST API

Hoy en día la mayoría de las empresas utilizan API REST para crear servicios. Esto se debe a que es un estándar lógico y eficiente para la creación de servicios web. Por poner algún ejemplo tenemos los sistemas de identificación de Facebook, la autenticación en los servicios de Google (hojas de cálculo, Google Analytics, …).

Debido a que simplifican la forma en que los desarrolladores integran los elementos de las aplicaciones nuevas en una arquitectura actual, las API permiten la colaboración entre el equipo comercial y el de IT. Las necesidades comerciales suelen cambiar rápidamente en respuesta a los mercados digitales en constante cambio, donde la competencia puede modificar un sector entero con una aplicación nueva. Para seguir siendo competitivos, es importante admitir la implementación y el desarrollo rápidos de servicios innovadores. El desarrollo de aplicaciones nativas de la nube es una forma identificable de aumentar la velocidad de desarrollo y se basa en la conexión de una arquitectura de aplicaciones de microservicios a través de las API.

Las API son un medio simplificado para conectar su propia infraestructura a través del desarrollo de aplicaciones nativas de la nube, pero también le permiten compartir sus datos con clientes y otros usuarios externos. Las API públicas representan un valor comercial único porque simplifican y amplían la forma en que se conecta con sus partners y, además, pueden rentabilizar sus datos (un ejemplo conocido es la API de Google Maps).

Existen tres enfoques respecto a las políticas de las versiones de las API:

- Privado: Las API solo se pueden usar internamente lo que le da a las empresas un mayor control sobre sus API.

- De Partners: Las API se comparten con partners empresariales específicos.

- Público: Todos tienen acceso a las API, así que otras empresas pueden desarrollar API que interactúen con las de usted y así convertirse en una fuente de innovaciones. Esto permite que terceros desarrollen aplicaciones que interactúan con su API.

En resumen, las API permiten habilitar el acceso a recursos y, al mismo tiempo, mantener la seguridad y el control.

ESTANDARIZACIÓN Y USO DE REST API

4.1 Reglas de una arquitectura REST

En los últimos años, el término REST API ha levantado una gran expectación en la comunidad de desarrolladores de todo el mundo.

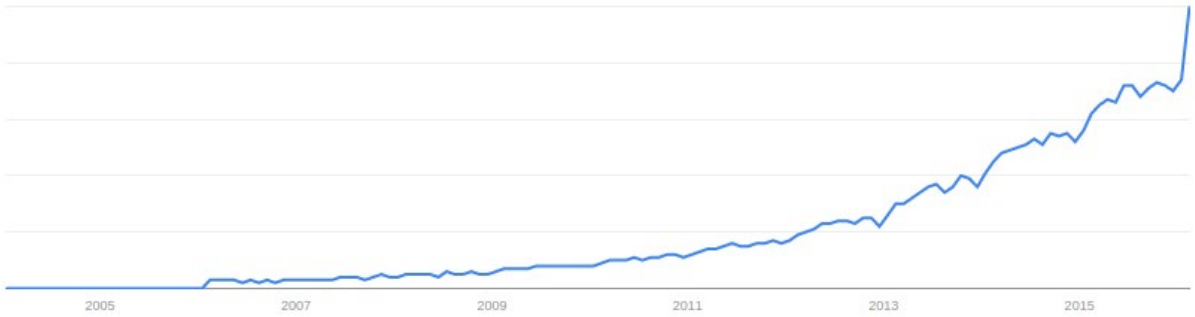


Fig 2: Evolución en el uso de REST API

Pero como desarrolladores, ¿estamos utilizando las mejores herramientas y buenas prácticas para abordar desarrollos de APIs REST?

En el desarrollo de aplicaciones, con una capa frontal que acceden a servicios back, es fundamental aislar el desarrollo del API del desarrollo de los clientes (móvil, web...etc) que lo utilizan.

Existen lenguajes de especificación y modelado de APIs, como RAML, API Blueprint o Swagger, que nos permiten definir conjuntamente un contrato de comunicación entre el consumidor y el API y generar un prototipo para que, desde ese momento, los equipos que producen el servicio y los que lo consumen puedan trabajar en paralelo y de forma independiente.

REST nos permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es increíblemente más simple y convencional que otras alternativas que se han usado en los últimos diez años como SOAP y XML-RPC.

Por todo esto, REST es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a Internet, e implementa una serie de restricciones o "reglas" que deberán seguirse:

- **Interfaz Uniforme:** La interfaz de basa en recursos (por ejemplo el recurso Empleado (Id, Nombre, Apellido, Puesto, …). El servidor mandará los datos en el formato que le solicite el cliente (json, xml, …) pero los datos y el tipo de los mismo que dicho servidor almacene (en base de datos) o maneje son totalmente transparentes al cliente. La representación del recurso que le llega al cliente, será suficiente para poder cambiar/borrar dicho recurso (siempre que tenga permisos). Para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT, …) 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:** Algunas aplicaciones HTTP incorporan memoria caché. Se configura lo que se conoce como protocolo *cliente-caché-servidor* sin estado: 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. Las respuestas se deben marcar de forma implícita o explícita como cacheables o no (en futuras peticiones, el cliente sabrá si puede reutilizar o no los datos que ya ha obtenido). Si ahorramos peticiones, mejoraremos la escalabilidad de la aplicación y el rendimiento en cliente (evitamos principalmente la latencia).

- **Cliente / Servidor:** La unión entre cliente y servidor se hace mediante la interfaz uniforme, con lo que mientras esta no se modifique podremos cambiar el cliente o el servidor de forma totalmente transparente.

- **Sistema de Capas**: El cliente puede estar conectado mediante la interfaz al servidor o a un intermediario, para el es irrelevante y desconocido. Al cliente solo le preocupa que la API REST funcione como debe (no importa el CÓMO sino el QUÉ). El uso de capas o servidores intermedios puede servir para aumentar la escalabilidad (sistemas de balanceo de carga, cachés), para implementar políticas de seguridad, …

4.2 Autenticación y Autorización

La protección y el control de la información en los servicios REST es un requisito obligatorio a la hora de publicar estos mediante una API (o cualquier otro sistema). La seguridad no sólo se determina mediante un "usuario y contraseña", sino que se debe verificar qué funciones y permisos están establecidos en cada usuario para que el acceso al sistema sea legítimo.

La distinción entre *autontication* y *autorización* es importante para entender cómo funcionan las API RESTful y por qué ciertos intentos de conexión o acceso a recursos son aceptadas o denegadas.

- **Autenticación:** Es la verificación de las credenciales de intento de conexión, lo que se traduce en enviar las credenciales desde cliente remoto al servidor (puedes in encriptadas o en texto plano) utilizando un protocolo de autenticación.

- **Autorización:** Es la verificación de que el intento de conexión (o acceso a ciertos recursos) está permitida (la autorización ocurre tras una autenticación previa).

La autentificación se basa en proporcionar un conjunto de credenciales para que el servidor o aplicación pueda verificar que eres quien dices ser y que la información que se proporciona es válida.

La autorización parte del proceso de autentificación y este garantiza que una vez que se determine que las credenciales de usuario son correctas, se aplicará el control de acceso para determinar a qué recursos se puede acceder en el sistema.

Fig 3: Autorización y Auntenticación

En otras palabras: *autenticación* es probar que eres quién dices ser (probar una correcta identidad) mientras que *autorización* es pedir acceso a ciertos recursos o acciones.

4.2.1 Métodos de autenticación API REST

Hay muchos métodos de seguridad y muchas formas de autentificarse, que pueden depender desde el tipo de dispositivo, el tipo de uso, la confidencialidad de la información, entre otros. No hay una sola manera de asegurar tu API. Es por ello que trataremos aquí los métodos principales y más utilizados.

Recordemos, no obstante, que en una API REST, enviar las credenciales una vez para iniciar sesión no es suficiente, ya que las API REST son asíncronas y por tanto no pueden recordar las credenciales ya que no existe ninguna sesión activa HTTP. Debemos indicar "quién somos" en cada petición que hagamos.

4.2.1.1 Autenticación Básica

Esta es la forma más sencilla de asegurar tu API. Se basa principalmente en un nombre de usuario y una contraseña para identificarte. Para comunicar estas credenciales desde el cliente hasta el servidor, se debe realizar mediante el encabezado HTTP Autorización (Authorization), según la especificación del protocolo HTTP (codificado en base64).

Aunque el nombre de usuario o contraseña estén codificados, cualquiera que intercepte la trasmisión de datos puede decodificar fácilmente esta información (esto se denomina ataque Man-In-The-Middle MiTM). Para proteger tu API mediante la autentificación básica debes configurar que las conexiones entre los clientes y tu servicio API funcionen únicamente mediante una conexión TLS/SSL, nunca sobre HTTP. Pero, incluso forzando en uso de SSL, el resultado es una degradación del tiempo de respuesta.

El único uso en el que se puede encajar la autenticación básica en nuestros días es dentro de una red interna donde (especialmente en el IoT) la velocidad puede ser un hándicap y HTTP Basic Authentication es un buen balance entre coste de implementación y velocidad en las peticiones.

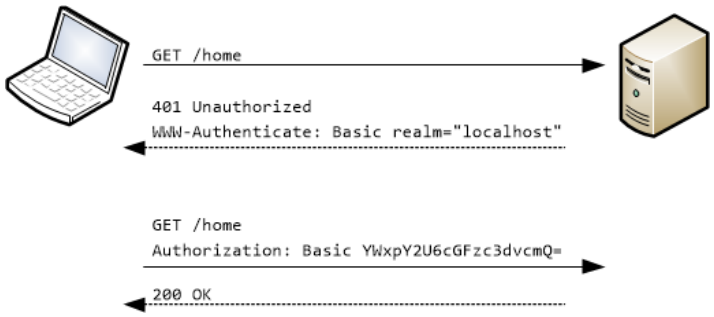


Fig 4: Autenticación básica

4.2.1.2 Autenticación basada en token

El hecho de usar un token como mecanismo involucrado en la autenticación no produce efectos visibles para los usuarios, ya que estos no tienen que lidiar con él directamente. Dicho de otra manera, todas las diferencias en el flujo son transparentes para el usuario. Por lo tanto, desde el punto de vista del usuario, para la autenticación se deberá seguir aportando el par usuario/clave habitual de cualquier otro sistema.

En la autenticación por token, cuando el cliente se ha podido validar como un usuario de la aplicación, recibe una cadena encriptada como respuesta. Esa cadena es el token y sirve para que, en los siguientes accesos, el usuario pueda informar al servidor que ya ha pasado por el proceso de autenticación.

En el siguiente diagrama, podemos ver cómo se realiza la recepción del token por parte del usuario. Primero, se envía la correspondiente información de login y, en caso de que se haya validado correctamente en el servidor, se devuelve el token al cliente.

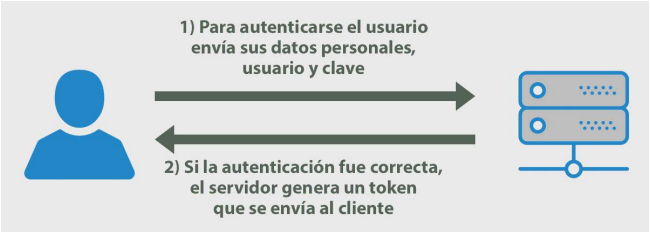


Fig 5.1 Recepción del token

Cada vez que el cliente deba realizar nuevas solicitudes al servidor, certificando que es un usuario correctamente autenticado, tendrá que enviar el token de nuevo al servidor. Por lo tanto, una vez recibido el token, deberá ser almacenado del lado del cliente para enviarlo con las posteriores solicitudes. En estos casos, el token suele viajar en las cabeceras del HTTP, de modo que llegue al servidor.

Fig 5.2 Envío del token en cada solicitud

El servidor que recibe el token tiene la capacidad de desencriptarlo, de modo que pueda comprobar qué usuario es el que está realizando esta solicitud. Durante el proceso de decodificación del token, el servidor puede comprobar si este es válido y si resulta serlo, puede recuperar toda la información encriptada en el mismo, que suele ser al menos la referencia inequívoca del usuario involucrado. Por supuesto, si en cualquier momento se detecta que el token no es correcto, se obligará al usuario a autenticarse nuevamente.

Para que la aplicación genere las cadenas de token y las pueda verificar, se utiliza programación del lado del servidor. Para la realización de todas estas operativas, generalmente, las aplicaciones se apoyan en librerías, que dependen de la tecnología de backend que se esté usando en el lado del servidor. Una muy común y sencilla de usar es JWT (JSON Web Token), que tiene implementaciones en diversos lenguajes.

Estas librerías, por lo general, liberan al desarrollador de la carga de trabajo más compleja de este proceso, ofreciendo una interfaz sencilla para la programación. Por ejemplo, ayudará a comprobar la validez de los tokens, asegurándose de que tengan un tiempo de caducidad configurable.

4.2.1.3 Clave API

A diferencia de los métodos anteriores, en este caso, primero se debe configurar el acceso a los recursos de la API, la cual debe generar una clave (*key*) y un *secret key* para cada cliente que requiera acceso a los servicios. Cada vez que una aplicación necesite consumir los datos de la API, se deberá enviar tanto la *key* como la *secret key*.

La generación de clave API fue creada en cierta manera para arreglar los problemas de seguridad que surgieron en los métodos anteriores, y aunque este sistema es más seguro que los métodos anteriores, la generación de credenciales debe ser manual y esto dificulta la escalabilidad de la API.

En este método se genera un valor único que se asigna a cada usuario que trate de acceder a la API "por primera vez", lo que significa que desde ese momento el usuario será conocido. Cuando dicho usuario intente acceder de nuevo a la API su identificador único se usa para probar que es el mismo usuario que ya ha accedido anteriormente.

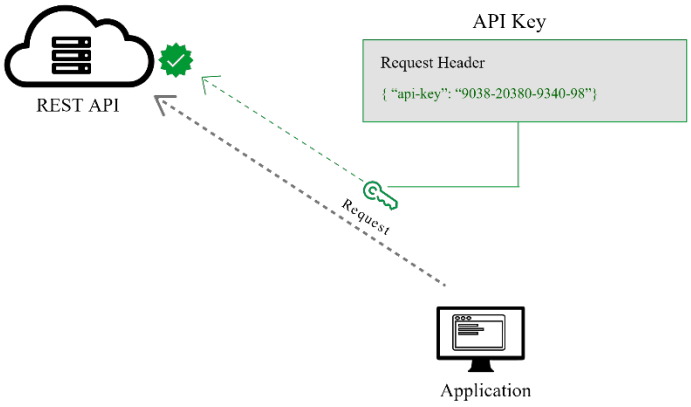


Fig 6: Autenticación por clave API

La misma recomendación que se hizo anteriormente debemos de recordarla ahora y si cabe de forma más encarecida y es que en la petición de consulta de la URL numerosas claves API son enviadas, lo que las hace fácilmente accesibles para quien no debiera, por lo que la mejor opción (y recomendada) es utilizar de nuevo el encabezado HTTP Autorización.

Otros problemas con la autentificación basada en clave API es la administración de claves. Con tareas tan relevantes como:

a. Genera la key y el secret key.

b. Enviar las credenciales a los desarrolladores.

c. Guardar de forma segura la key y el secret key.

Puede ser complicado poder almacenar y administrar estas credenciales. Es por ello que es imprescindible contar con una API Gateway.

4.2.1.4 OAuth

En diciembre de 2007, OAuth 1.0 abordó el problema de la autenticación con un marco basado en firmas digitales. Era fuerte y robusto, y por tanto poco a poco fue adoptado por las principales compañías que ofrecía servicios REST (Google comenzó a admitir OAuth 1.0 en 2008. En 2010, Twitter obligó a todas las aplicaciones de terceros a usar su implementación OAuth 1.0, …).

Sin embargo, OAuth 1.0 requería implementación criptográfica e interoperabilidad criptográfica, que si bien era seguro, era tedioso de implementar y fue un desafío para muchos desarrolladores.

En Octubre de 2012 surgió OAuth 2.0, fruto de la unión e influencia de estas grandes empresas al organismo estándar OAuth, el cual tuvo como resultado una implementación mucho más sencilla que su predecesor. El principal cambio en esta versión fue que ya no era necesario incluir la *firma* con la clave en cada llamada, en lugar de eso la implementación común usaba uno de estos tipos de tokens:

- **Token de acceso:** Enviado de forma similar a la clave API, permite a la aplicación acceder a los datos del usuario (de forma opcional estos tokens pueden expirar).

- **Token de actualización:** Estos token recuperan un nuevo token de acceso si este ha expirado. Se combina Autenticación y Autorización para permitir un alcance más sofisticado y un control de validez.

OAuth 2.0 es la mejor opción para identificar cuentas de usuario personales y otorgar los permisos adecuados. En este método, el usuario inicia sesión en un sistema. Ese sistema luego solicitará autenticación, generalmente en forma de token. El usuario reenviará esta solicitud a un servidor de autenticación, que rechazará o permitirá esta autenticación. Desde aquí, el token se proporciona al usuario y luego al solicitante. Dicho token puede ser verificado en cualquier momento independientemente del usuario por el solicitante para la validación y puede usarse a lo largo del tiempo con un alcance y una edad de validez estrictamente limitados.

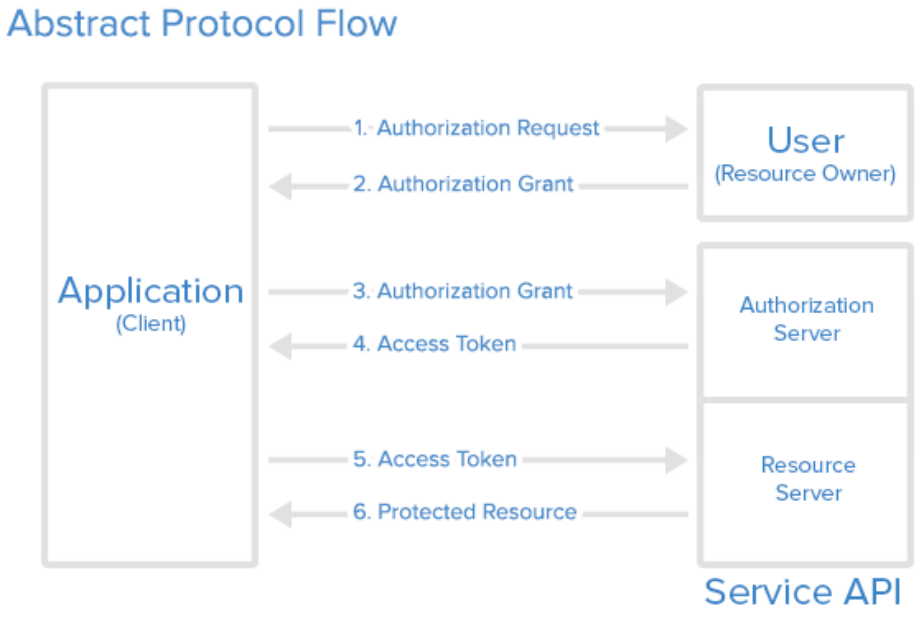


Fig 7: Flujo de OAuth 2.0

Los flujos (también llamados tipos de concesión) son escenarios que realiza un cliente API para obtener un token de acceso del servidor de autorizaciones. OAuth 2.0 proporciona varios flujos populares adecuados para diferentes tipos de clientes API:

- **Código de autorización:** utilizado principalmente para aplicaciones web móviles y del lado del servidor. Este flujo es similar a cómo los usuarios se registran en una aplicación web utilizando su cuenta de Facebook o Google.

- **Implícito:** Este flujo requiere que el cliente recupere un token de acceso directamente. Es útil en los casos en que las credenciales del usuario no se pueden almacenar en el código del cliente porque un tercero puede acceder fácilmente a ellas. Es adecuado para aplicaciones web, de escritorio y móviles que no incluyen ningún componente del servidor.

- **Contraseña de recurso:** requiere iniciar sesión con un nombre de usuario y contraseña. Como en ese caso, las credenciales serán parte de la solicitud, este flujo es adecuado solo para clientes de confianza (por ejemplo, aplicaciones oficiales lanzadas por el proveedor de API).

- **Credenciales del Cliente:** destinado a la autenticación de servidor a servidor, este flujo describe un enfoque cuando la aplicación del cliente actúa en su propio nombre y no en nombre de cualquier usuario individual. En la mayoría de los escenarios, este flujo proporciona los medios para permitir a los usuarios especificar sus credenciales en la aplicación del cliente, para que pueda acceder a los recursos bajo el control del cliente.

4.3 Protocolo Cliente / Servidor

La arquitectura cliente-servidor es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios (servidores), y los demandantes, (clientes). Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.

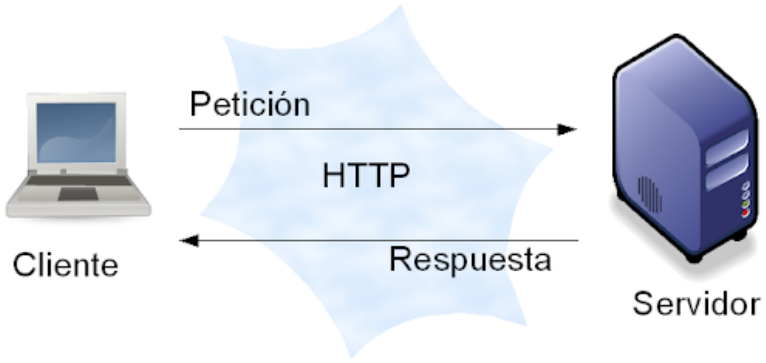


Fig 8: Modelo Cliente / Servidor

En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema. El cliente suele ser estaciones de trabajo que solicitan varios servicios al servidor, mientras que un servidor es una máquina que actúa como depósito de datos, como procesador de la información y se encarga de dar la respuesta demandada por el cliente.

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un solo programa. Los tipos específicos de servidores incluyen los servidores web, los servidores de archivo, los servidores del correo, etc. Mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma.

Una disposición muy común son los sistemas multicapa en los que el servidor se descompone en diferentes programas que pueden ser ejecutados por diferentes computadoras aumentando así el grado de distribución del sistema. El más claro ejemplo de uso de una arquitectura cliente servidor es la red de Internet donde existen ordenadores de diferentes personas conectadas alrededor del mundo, las cuales se conectan a través de los servidores de su proveedor de Internet por ISP donde son redirigidos a los servidores de las páginas que desean visualizar y de esta manera la información de los servicios requeridos viaja a través de Internet dando respuesta a la solicitud demandada.

La red cliente-servidor es una red de comunicaciones en la cual los clientes están conectados a un servidor, en el que se centralizan los diversos recursos y aplicaciones con que se cuenta; y que los pone a disposición de los clientes cada vez que estos son solicitados. Esto significa que todas las gestiones que se

realizan se concentran en el servidor, de manera que en él se disponen los requerimientos provenientes de los clientes que tienen prioridad, los archivos que son de uso público y los que son de uso restringido, los archivos que son de sólo lectura y los que, por el contrario, pueden ser modificados, etc. Este tipo de red puede utilizarse conjuntamente en caso de que se esté utilizando en una red mixta.

Un sistema Cliente/Servidor está basado en las siguientes características:

- **Servicio:** Unidad básica de diseño. El servidor los proporciona y el cliente los utiliza.
- **Recursos compartidos:** Muchos clientes utilizan los mismos servidores y, a través de ellos, comparten tanto recursos lógicos como físicos.
- **Protocolos asimétricos:** Los clientes inician "conversaciones". Los servidores esperan su establecimiento pasivamente.
- **Transparencia de localización física de los servidores y clientes:** El cliente no tiene por qué saber dónde se encuentra situado el recurso que desea utilizar.
- **Independencia de la plataforma**: Las funciones de Cliente y Servidor pueden estar en plataformas separadas, o en la misma plataforma. El HW y SW que se emplee es irrelevante.
- **Sistemas débilmente acoplados:** Interacción basada en envío de mensajes.
- **Encapsulamiento de servicios:** Los detalles de la implementación de un servicio son transparentes al cliente. La interrelación entre el hardware y el software están basados en una infraestructura poderosa, de tal forma que el acceso a los recursos de la red no muestra la complejidad de los diferentes tipos de formatos de datos y de los protocolos.
- **Escalabilidad horizontal** (añadir clientes) **y vertical** (ampliar potencia de los servidores). Cada plataforma puede ser escalable independientemente. Los cambios realizados en las plataformas de los Clientes o de los Servidores, ya sean por actualización o por reemplazo tecnológico, se realizan de una manera transparente para el usuario final.
- **Integridad:** Datos y programas centralizados en servidores facilitan su integridad y mantenimiento.

Lo normal es que los servicios de un mismo servidor puedan ser utilizados por múltiples clientes distintos. Tanto los programas cliente como los servidores son con frecuencia parte de un programa o aplicación mayores.

4.3.1 Elementos principales

En la arquitectura Cliente/Servidor el remitente de una solicitud es conocido como **cliente**, y es todo proceso que reclama servicios de otro. Una definición un poco más elaborada podría ser que **cliente** es el proceso que permite al usuario formular los requerimientos y pasarlos al servidor (también se le conoce con el término front-end). Sus características son:

- Es quien inicia solicitudes o peticiones, tienen por tanto un papel activo en la comunicación (dispositivo maestro o amo).
- Espera y recibe las respuestas del servidor.
- Por lo general, puede conectarse a varios servidores a la vez.
- Normalmente interactúa directamente con los usuarios finales mediante una interfaz gráfica de usuario.
- Se encarga de procesar la lógica de la aplicación y hacer validaciones locales.
- Recibir resultados del servidor y formatearlos.

De este modo **el cliente se puede clasificar** en:

- **Cliente basado en aplicación de usuario**. Si los datos son de baja interacción y están fuertemente relacionados con la actividad de los usuarios de esos clientes.
- **Cliente basado en lógica de negocio**. Toma datos suministrados por el usuario y/o la base de datos y efectúa los cálculos necesarios según los requerimientos del usuario.

Un **servidor** es todo proceso que proporciona un servicio a otros. Es el proceso encargado de atender a múltiples clientes que hacen peticiones de algún recurso administrado por él (también se le conoce con el término back-end). El servidor normalmente maneja todas las funciones relacionadas con la mayoría de las reglas del negocio y los recursos de datos. Sus características son:

- Al iniciarse esperan a que lleguen las solicitudes de los clientes, desempeñan entonces un papel pasivo en la comunicación (dispositivo esclavo).
- Tras la recepción de una solicitud, la procesan y luego envían la respuesta al cliente.
- Por lo general, acepta las conexiones de un gran número de clientes (aunque en ocasiones el número máximo de las mismas puede estar limitado).

4.4 ¿Por qué utilizar REST API en un negocio?

Para las empresas, las API permiten crear soluciones que satisfagan las necesidades de sus clientes y brinden mejores experiencias sin incrementar considerablemente el costo, así como agilizar sus procesos internos. Sus usos van desde la creación de una app móvil para sus clientes hasta el intercambio de información entre departamentos de la empresa o con terceros.

Las personas utilizamos cada vez más los ordenadores, tablets, dispositivos móviles, … pero es el llamado "Internet de las cosas" (IoT) el que va a suponer una revolución en los próximos años. El lenguaje que utilizan las máquinas para comunicarse es HTTP a través de API REST. Tu reloj deportivo almacena los datos recopilados en un servidor a través de una API. Diversos dispositivos de salud pueden monitorizar la salud de un paciente y notificarla a un médico o generar una alerta a través de una API. Un coche puede notificar, sin intervención humana, una situación de accidente a través de una API. Las posibilidades son inmensas. Aquellas empresas que estén preparadas para el internet de las cosas tendrán una ventaja frente a sus competidores.

Algunos de los beneficios concretos de utilizar API son:

- **Automatización:** Gracias a la automatización e integración de procesos se ahorran costes, tiempo y esfuerzo. Gran parte de las estrategias de automatización se basan en la integración de aplicaciones, para lo cual se vuelve indispensable el uso de API.

- **Innovación:** Las API conforman una de las claves para la transformación digital y creación de modelos de negocio innovadores. Son la base de la economía de las aplicaciones, las cuales pueden desarrollarse más rápido, a menor costo y con características que superen las expectativas de los clientes.

- **Alianzas:** Cuando una empresa reutiliza herramientas creadas por personas y empresas talentosas para diseñar nuevos productos o servicios, se da una excelente oportunidad de crear alianzas estratégicas donde ambas partes se beneficien.

Las API son una herramienta clave en el crecimiento de un negocio. Reestructuran sistemas internos que favorecen la innovación, agilizan procesos, reducen costos por mantenimiento y brindan nuevas oportunidades de construir alianzas con otras empresas, llegar a los clientes y ofrecer una mejor experiencia de usuario.

Las mismas ventajas indicadas anteriormente se pueden aplicar a los desarrollos informáticos internos de la empresa (APIs Internas). Detrás de una API REST pueden existir tecnologías y sistemas muy heterogéneos y por tanto una API permite acceder a esos sistemas de una forma conocida, uniforme e independiente. Imaginemos que diversas aplicaciones de una empresa deben acceder a la base de datos de clientes. Crear una API que sirva de interfaz entre la base de datos de clientes y las aplicaciones tiene múltiples ventajas:

- Todas las aplicaciones se integrarán con la base de datos de clientes de la misma manera, sin importar en qué tecnologías se desarrollen las aplicaciones.

- Si cambia la estructura de la base de datos sólo hay que adaptar la API. Si las aplicaciones accediesen directamente quizás habría que cambiarlas todas.

- Crear una API significa añadir una barrera de seguridad. Las aplicaciones no tienen acceso directo a la base de datos, sólo van a poder realizar lo que les permita la API.

- La base de datos tendrá una vida más larga que las aplicaciones que puedan utilizarla. Quizás en un momento dado se necesite desarrollar una aplicación con una plataforma tecnológica que no ofrezca la posibilidad de conectarse directamente a la base de datos, pero es casi seguro que esa plataforma permitirá las comunicaciones por HTTP. Utilizar una API REST nos da garantías de futuro.

- Abrazar la tecnología API REST internamente facilita su uso externamente. Quizás el acceso a la base de datos de clientes (o productos, facturas, pedidos, etc.) desde un dispositivo móvil sea una necesidad importante para los comerciales de la empresa cuando están de viaje. El paso de un uso interno a otro externo es más fácil si ya se está utilizando la tecnología necesaria.

4.4.1 B2B a través de las REST API

Pensemos en las API REST como un canal más para ofrecer servicios y hacer negocios a través de la integración con otras empresas. Estas son algunas de las posibilidades:

- **Ofrecer un mejor servicio y posibilitar la expansión del negocio:** un ejemplo claro es la empresa Fastway, dedicada al transporte de paquetes, y su API REST . La API tiene dos funciones: permite calcular el coste de envío de un paquete y permite conocer su estado de envío y entrega. Las empresas que utilizan los servicios de Fastway seguramente tendrán sus sitios web y/o aplicaciones móviles. La API de Fastway les permite dar un mejor servicio al incorporar en sus propios servicios online la posibilidad de informar a sus clientes el coste de los envíos y conocer en qué situación se encuentran estos. Si cuidas a tus clientes estás cuidando tu negocio. La API REST de Fastway permite dar servicio las 24 horas del día y sobre todo permite la integración con los sistemas de sus propios clientes. Los clientes de los clientes de Fastway no necesitan acceder al sitio web de Fastway porque obtienen lo que necesitan en los propios sistemas de los clientes de Fastway, debido a que Fastway ofrece una API REST.

- **Favorecer al núcleo del negocio:** en el ejemplo anterior informar a los clientes sobre tarifas y entregas es una ventaja, pero no es el núcleo del negocio. Existen otras situaciones en las que una API REST se engrana en la propia cadena comercial de la empresa. Un ejemplo de esto se da cuando el sistema ERP de una empresa puede ser accedido por los clientes, en las condiciones que se definan, a través de una API REST, permitiendo a sus clientes que sus propios sistemas informáticos se integren con los de la empresa que ofrece la API REST para realizar pedidos, consultar stocks, etc. En el año 2016 el 50% de la colaboración B2B entre empresas se realizó a través de Web APIs.

- **REST API como un producto:** Existen empresas con modelos de negocio basados en servicios de información que puede ser ofrecidos a través de APIs aprovechando todas las ventajas descritas anteriormente. En estos casos el propio negocio es la comercialización del uso de la API para ofrecer un servicio. También están en esta categoría todas las empresas que venden publicidad que puede ser insertada en sitios web y aplicaciones móviles a través de sus APIs. Una API como producto puede ser vendida directamente u ofrecerse a otras empresas para que hagan negocio con ella utilizándola como una marca blanca.

IMPLEMENTACIÓN DE UNA REST API PARA PLATAFORMA COLABORATIVA

5.1 Introducción

Una vez comentado lo que es una REST API, así como las ventajas que aporta, reglas que se le aplican, elementos principales y conceptos a tener en cuenta, en este punto analizaremos el desarrollo y la implementación de un ejemplo real de REST API.

El proyecto está hecho en Visual Studio 2017 y C#, y como simulación de la parte cliente se ha utilizado Postman. Para cumplir con el propósito de TFM del Máster en Ingeniería Web (y la parte pedagógica asociada) en el proyecto se ha tratado de incluir el máximo número de conceptos asociados a dicho master. Así por ejemplo, se puede llamar al REST Service pasando la información de maneras y formatos diferentes, desde datos simples como parámetros a través de la URL, hasta estructuras complejas que se pasan en formato JSON. Igualmente, en las respuestas hay diversidad de estructuras: desde simples (un boolean) hasta estructuras complejas en formato JSON o XML (todo esto se ha visto en el master).

Durante la implementación, se puede ver claramente que el proyecto sigue el modelo MVC (también estudiado en el master) en la medida de lo posible (ya que es un REST Service sin interface de usuario), y es que se compone de una parte en la que se definen las distintas clases con los modelos de datos, otra con los controladores de todas las clases (encargados de realizar las operaciones pertinentes) y por último las clases "Persistence" (que son las que se encargar de realizar el acceso a datos).

Como motor de base de datos se ha utilizado SQL Server 2016, donde nos apoyamos en una base de datos de estructura sencilla, que también carga con una pequeña parte de responsabilidad no sólo para con los datos, si no por ejemplo, para evitar duplicidades en las tablas relacionales (cosa que se podría haber implementado en el controlador, pero es mucho mejor práctica dejar esa responsabilidad a la BBDD).

Añadir también que el proyecto consta de una parte de UNIT Test (que también se ha visto en el master) que asegura que cualquier cambio o nueva funcionalidad no afecta a lo que ya ha sido implementado.

5.2 Descripción de la necesidad a cubrir y propuesta

El proyecto constituye la parte back-end de un proyecto total en el que se trata de simular una plataforma colaborativa de contratación de técnicos de IT asociados a una tecnología con disponibilidad de un horario.

En este caso se ofrece una REST API que permite realizar la contratación/reserva de dichos técnicos, así como las labores propias de administración de todas las clases asociadas (horarios, tecnologías, …).

Como propuesta, además, se ha utilizado en la medida de lo posible el modelo MVC (Model View Controller) en el que los modelos son objetos o clases con propiedades "get / set" que se utilizan para recuperar y almacenar el estado del modelo, el controlador administra la interacción de los usuarios y qué datos enviar como respuesta, y en nuestro caso la vista sería el cliente que hace la petición HTTP que se encargaría de recibir los datos enviados como respuesta y mostrar la UI con dichos datos al usuario.

Por último, se implementa en esta REST API una clase Persistence que es la que se encarga del acceso a datos, así como del formateo de los mismos para dar respuesta a la petición del controlador, de esta forma tanto la modularidad como el encapsulamiento están asegurados.

Modelos:

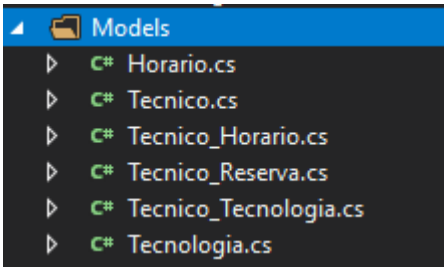


Fig 9: Modelos

Controladores:

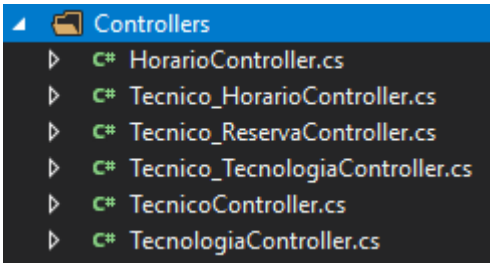


Fig 10: Controladores

Persistence:

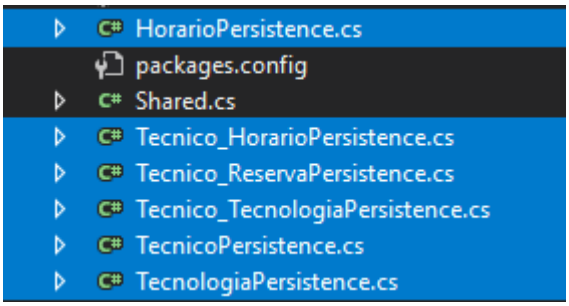


Fig 11: Persistence

5.3 Tipo de datos utilizados

En lo que a la base de datos se refiere, ya se ha comentado que se utiliza SQL Server 2016, y en sus tablas no se han almacenado tipos de datos complejos. De esta forma, para los identificadores y claves primarias se utiliza *bigint*. Para los textos se utiliza *nchar* con diferentes longitudes dependiendo del tipo de información que vaya a guardar. Y para los numéricos se utiliza el tipo de datos *int*.

En cuanto a las peticiones cliente a la REST API, se utilizan tipos de datos simples como enteros pasados en formato de URL, como FormRequest, … y tipos de datos complejos (estructuras de tipo *modelo/clase*) en formato JSON (raw) como por ejemplo a hora de hacer una petición POST.

En cuanto a las respuestas, se pueden recibir desde tipos de datos *boolean* para las peticiones DELETE, *string* (cadenas de caracteres) para las peticiones POST, estructuras del tipo *modelo/clase* para las peticiones GET que pueden ser devueltas en formato JSON o en formato XML (el formato de respuestas en el que se espera la información debe especificarse en la cabecera de la petición, si no se hace por defecto se entiende que se espera en formato JSON).

5.4 Usuarios

Podemos distinguir tres tipos principales de usuarios en la aplicación:

- **Técnicos:** Estos son los que una vez dados de alta en el sistema (tarea que pueden hacer ellos mismos o el usuario Administrador) recibirán las peticiones de contratación de sus servicios para una tecnología concreta en un horario concreto

- **Contratistas:** Estos usuarios son los que harán las peticiones de contratación de técnicos para una determinada tecnología. Al ser una plataforma colaborativa, la negociación se realiza directamente entre el técnico y el contratista, por lo que estos usuarios no tienen que estar registrados en el sistema.

- **Administradores:** Se encargan de la gestión tanto de los técnicos, como de los horarios, tecnologías, etc… dentro de la plataforma. Son los que tradicionalmente realizarán las tareas de alta-baja-modificación.

5.5 Casos de uso

5.5.1 Técnico

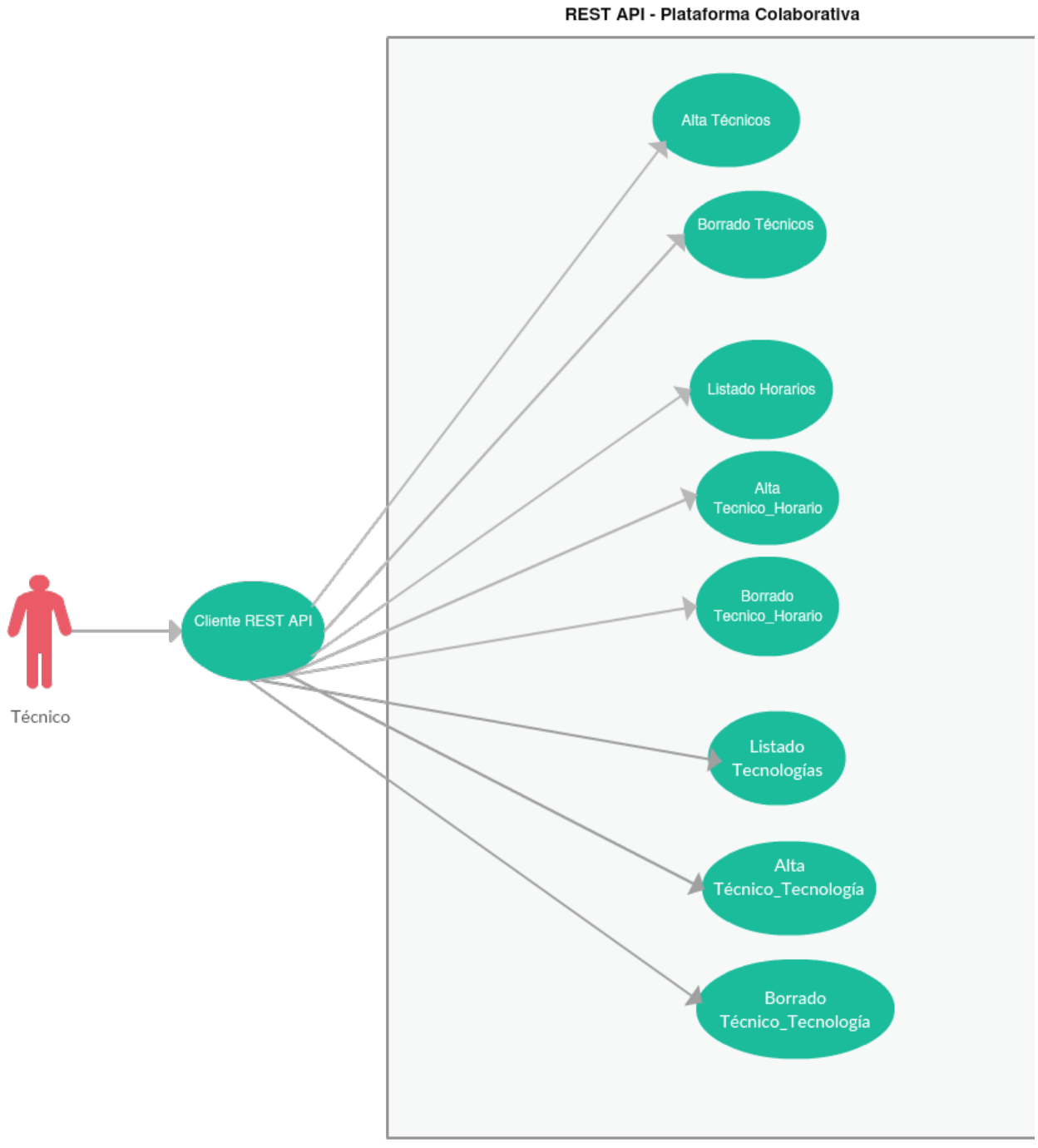


Fig 12: Caso de uso usuario Técnico

5.5.2 Contratista

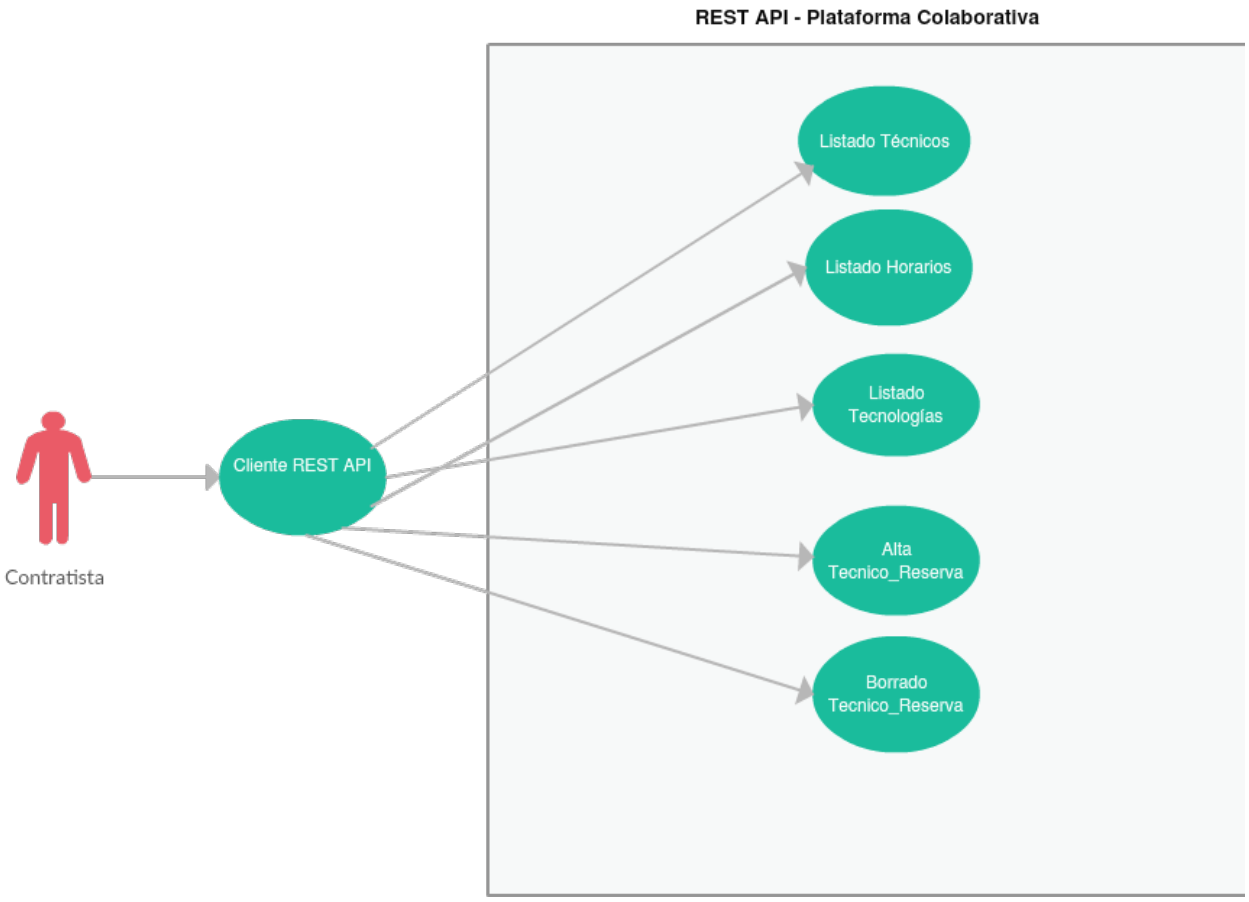


Fig 13: Caso de uso usuario Contratista

5.5.3 Administrador

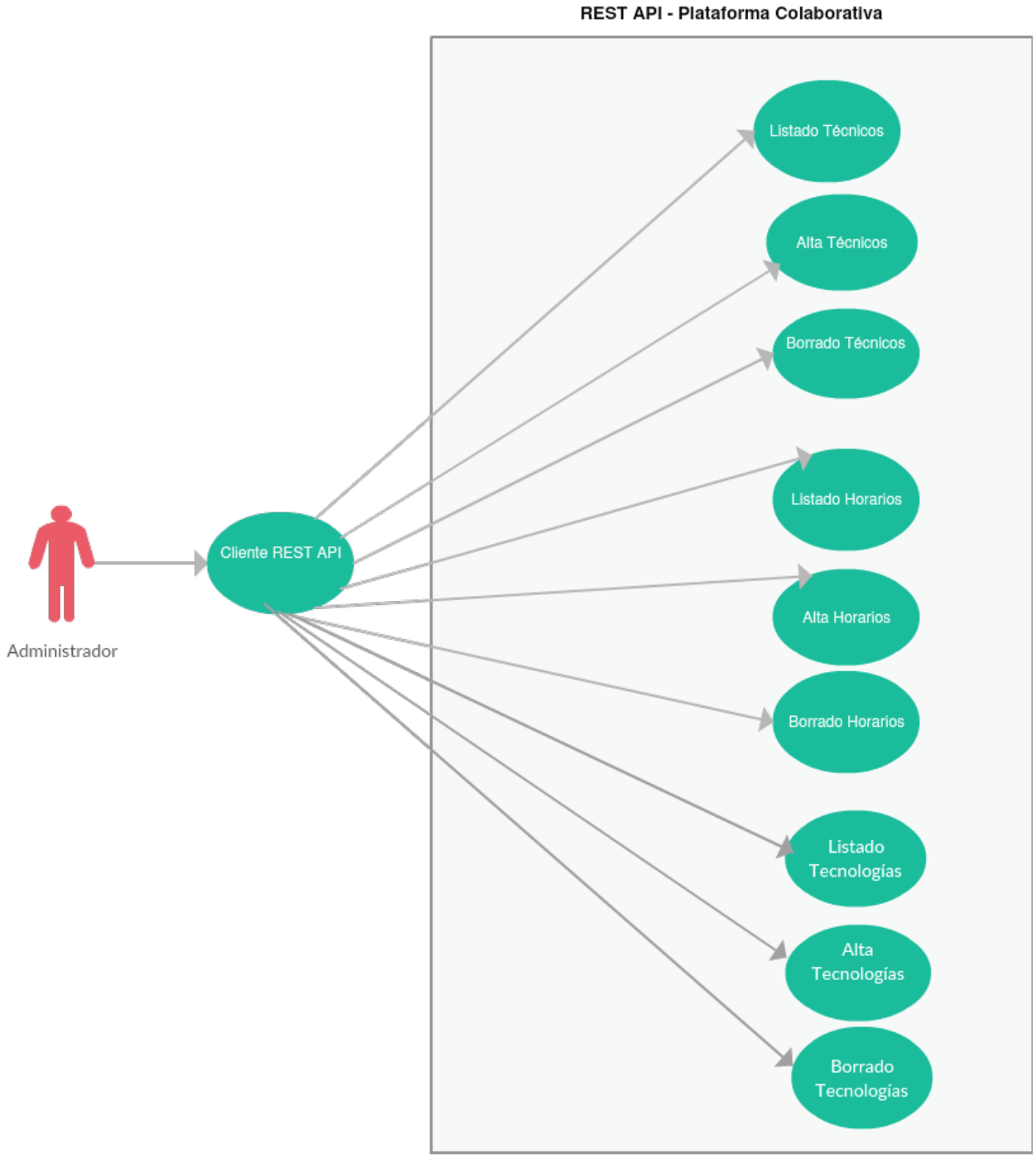


Fig 14:Caso de uso usuario Administrador

5.6 Diseño

5.6.1 BBDD

Comenzaremos con el diseño de la BBDD que soporta el almacenamiento de los datos básicos que maneja la API REST

Horario: Es la tabla que almacena la información referida al horario

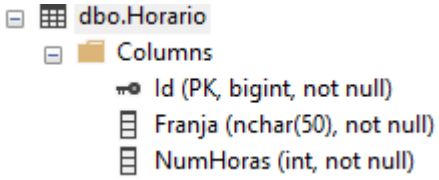


Fig 15.1: Tabla "Horario"

```
USE [TFM_Plat_Colaborativa]
GO

/****** Object:  Table [dbo].[Horario]    Script Date: 17/02/2020 19:13:46 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Horario](
    [Id] [bigint] IDENTITY(1,1) NOT NULL,
    [Franja] [nchar](50) NOT NULL,
    [NumHoras] [int] NOT NULL,
 CONSTRAINT [PK_Horario] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Fig 15.2: Script Tabla "Horario"

Técnicos: Es la tabla que almacena la información referida a los técnicos

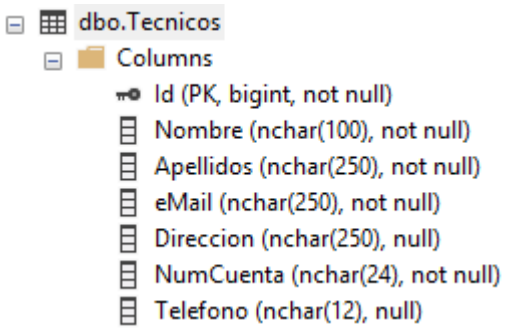


Fig 16.1: Tabla "Tecnicos"

```
USE [TFM_Plat_Colaborativa]
GO

/****** Object:  Table [dbo].[Tecnicos]    Script Date: 17/02/2020 19:16:52 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Tecnicos](
    [Id] [bigint] IDENTITY(1,1) NOT NULL,
    [Nombre] [nchar](100) NOT NULL,
    [Apellidos] [nchar](250) NOT NULL,
    [eMail] [nchar](250) NOT NULL,
    [Direccion] [nchar](250) NULL,
    [NumCuenta] [nchar](24) NOT NULL,
    [Telefono] [nchar](12) NULL,
 CONSTRAINT [PK_Tecnicos] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Fig 16.2: Script Tabla "Tecnicos"

Tecnología: Es la tabla que almacena la información referida a la tecnología

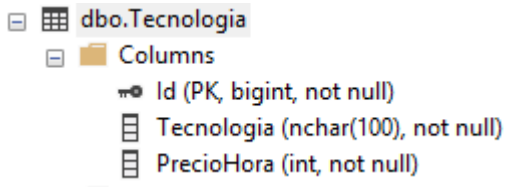


Fig 17.1: Tabla "Tecnología"

```
USE [TFM_Plat_Colaborativa]
GO

/****** Object:  Table [dbo].[Tecnologia]    Script Date: 17/02/2020 19:20:47 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Tecnologia](
    [Id] [bigint] IDENTITY(1,1) NOT NULL,
    [Tecnologia] [nchar](100) NOT NULL,
    [PrecioHora] [int] NOT NULL,
 CONSTRAINT [PK_Tecnologia] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Fig 17.2: Script Tabla "Tecnologia"

Comentaremos a continuación el diseño de las tablas relacionales, tales como:

Técnico_Horario: Esta tabla establece la relación entre los técnicos y los horarios en los que están disponibles.

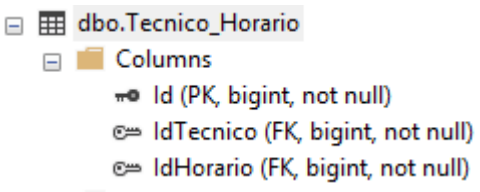


Fig 18.1: Tabla "Tecnico_Horario"

```
USE [TFM_Plat_Colaborativa]
GO

/****** Object:  Table [dbo].[Tecnico_Horario]    Script Date: 17/02/2020 19:23:19 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Tecnico_Horario](
    [Id] [bigint] IDENTITY(1,1) NOT NULL,
    [IdTecnico] [bigint] NOT NULL,
    [IdHorario] [bigint] NOT NULL,
 CONSTRAINT [PK_Tecnico_Horario] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY],
 CONSTRAINT [noDuplicates] UNIQUE NONCLUSTERED
(
    [IdTecnico] ASC,
    [IdHorario] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY],
 CONSTRAINT [noDuplicates_T_H] UNIQUE NONCLUSTERED
(
    [IdTecnico] ASC,
    [IdHorario] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Tecnico_Horario]  WITH CHECK ADD  CONSTRAINT [FK_Tecnico_Horario_Horario] FOREIGN KEY([IdHorario])
REFERENCES [dbo].[Horario] ([Id])
GO

ALTER TABLE [dbo].[Tecnico_Horario] CHECK CONSTRAINT [FK_Tecnico_Horario_Horario]
GO

ALTER TABLE [dbo].[Tecnico_Horario]  WITH CHECK ADD  CONSTRAINT [FK_Tecnico_Horario_Tecnicos] FOREIGN KEY([IdTecnico])
REFERENCES [dbo].[Tecnicos] ([Id])
GO

ALTER TABLE [dbo].[Tecnico_Horario] CHECK CONSTRAINT [FK_Tecnico_Horario_Tecnicos]
GO
```

Fig 18.2: Script Tabla "Tecnico_Horario"

Técnico_Reserva: Establece la relación entre las reservas actuales, los técnicos reservados y en qué horario están reservados.

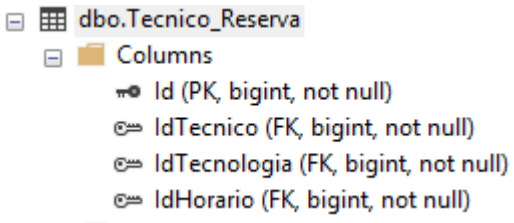


Fig 19.1: Tabla "Tecnico_Reserva"

```
USE [TFM_Plat_Colaborativa]
GO

/****** Object:  Table [dbo].[Tecnico_Reserva]    Script Date: 17/02/2020 19:26:02 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Tecnico_Reserva](
    [Id] [bigint] IDENTITY(1,1) NOT NULL,
    [IdTecnico] [bigint] NOT NULL,
    [IdTecnologia] [bigint] NOT NULL,
    [IdHorario] [bigint] NOT NULL,
 CONSTRAINT [PK_Tecnico_Reserva] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY],
 CONSTRAINT [noDuplicates_T_R] UNIQUE NONCLUSTERED
(
    [IdTecnico] ASC,
    [IdTecnologia] ASC,
    [IdHorario] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Tecnico_Reserva]  WITH CHECK ADD  CONSTRAINT [FK_Tecnico_Reserva_Horario] FOREIGN KEY([IdHorario])
REFERENCES [dbo].[Horario] ([Id])
GO

ALTER TABLE [dbo].[Tecnico_Reserva] CHECK CONSTRAINT [FK_Tecnico_Reserva_Horario]
GO

ALTER TABLE [dbo].[Tecnico_Reserva]  WITH CHECK ADD  CONSTRAINT [FK_Tecnico_Reserva_Tecnicos] FOREIGN KEY([IdTecnico])
REFERENCES [dbo].[Tecnicos] ([Id])
GO

ALTER TABLE [dbo].[Tecnico_Reserva] CHECK CONSTRAINT [FK_Tecnico_Reserva_Tecnicos]
GO

ALTER TABLE [dbo].[Tecnico_Reserva]  WITH CHECK ADD  CONSTRAINT [FK_Tecnico_Reserva_Tecnologia] FOREIGN KEY([IdTecnologia])
REFERENCES [dbo].[Tecnologia] ([Id])
GO

ALTER TABLE [dbo].[Tecnico_Reserva] CHECK CONSTRAINT [FK_Tecnico_Reserva_Tecnologia]
GO
```

Fig 19.2: Script Tabla "Tecnico_Reserva"

Técnico_Tecnologia: Esta tabla es la que establece la relación entre los diferentes técnicos y las diferentes tecnologías.

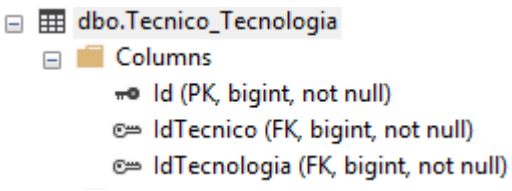


Fig 20.1: Tabla "Tecnico_Tecnologia"

```
USE [TFM_Plat_Colaborativa]
GO

/****** Object:  Table [dbo].[Tecnico_Reserva]    Script Date: 17/02/2020 19:26:02 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Tecnico_Reserva](
    [Id] [bigint] IDENTITY(1,1) NOT NULL,
    [IdTecnico] [bigint] NOT NULL,
    [IdTecnologia] [bigint] NOT NULL,
    [IdHorario] [bigint] NOT NULL,
 CONSTRAINT [PK_Tecnico_Reserva] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY],
 CONSTRAINT [noDuplicates_T_R] UNIQUE NONCLUSTERED
(
    [IdTecnico] ASC,
    [IdTecnologia] ASC,
    [IdHorario] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Tecnico_Reserva]  WITH CHECK ADD  CONSTRAINT [FK_Tecnico_Reserva_Horario] FOREIGN KEY([IdHorario])
REFERENCES [dbo].[Horario] ([Id])
GO

ALTER TABLE [dbo].[Tecnico_Reserva] CHECK CONSTRAINT [FK_Tecnico_Reserva_Horario]
GO

ALTER TABLE [dbo].[Tecnico_Reserva]  WITH CHECK ADD  CONSTRAINT [FK_Tecnico_Reserva_Tecnicos] FOREIGN KEY([IdTecnico])
REFERENCES [dbo].[Tecnicos] ([Id])
GO

ALTER TABLE [dbo].[Tecnico_Reserva] CHECK CONSTRAINT [FK_Tecnico_Reserva_Tecnicos]
GO

ALTER TABLE [dbo].[Tecnico_Reserva]  WITH CHECK ADD  CONSTRAINT [FK_Tecnico_Reserva_Tecnologia] FOREIGN KEY([IdTecnologia])
REFERENCES [dbo].[Tecnologia] ([Id])
GO

ALTER TABLE [dbo].[Tecnico_Reserva] CHECK CONSTRAINT [FK_Tecnico_Reserva_Tecnologia]
GO
```

Fig 20.2: Script Tabla "Tecnico_Tecnologia"

Además del diseño de estas tablas, como ya se comentó anteriormente, se ha dotado a la base de datos de la responsabilidad de evitar duplicados en las tablas relacionales, el diseño de esas restricciones se puede ver en el siguiente script:

```
ALTER TABLE [dbo].[Tecnico_Tecnologia]  WITH CHECK ADD  CONSTRAINT [FK_Tecnico_Tecnologia_Tecnicos] FOREIGN KEY([IdTecnico])
REFERENCES [dbo].[Tecnicos] ([Id])
GO

ALTER TABLE [dbo].[Tecnico_Tecnologia] CHECK CONSTRAINT [FK_Tecnico_Tecnologia_Tecnicos]
GO

ALTER TABLE [dbo].[Tecnico_Tecnologia]  WITH CHECK ADD  CONSTRAINT [FK_Tecnico_Tecnologia_Tecnologia] FOREIGN KEY([IdTecnologia])
REFERENCES [dbo].[Tecnologia] ([Id])
GO

ALTER TABLE [dbo].[Tecnico_Tecnologia] CHECK CONSTRAINT [FK_Tecnico_Tecnologia_Tecnologia]
GO
```

Fig 21: Script para evitar duplicidades en las tablas relacionales

5.6.2 REST API

Como comentamos en la introducción, hemos tratado de encapsular y modularizar al máximo nuestro servicio, de esta forma, se ha seguido en la medida de lo posible el modelo MVC para el diseño y desarrollo del mismo. Por tanto, abordaremos de forma independiente cada uno de los "grupos" que componen el proyecto final.

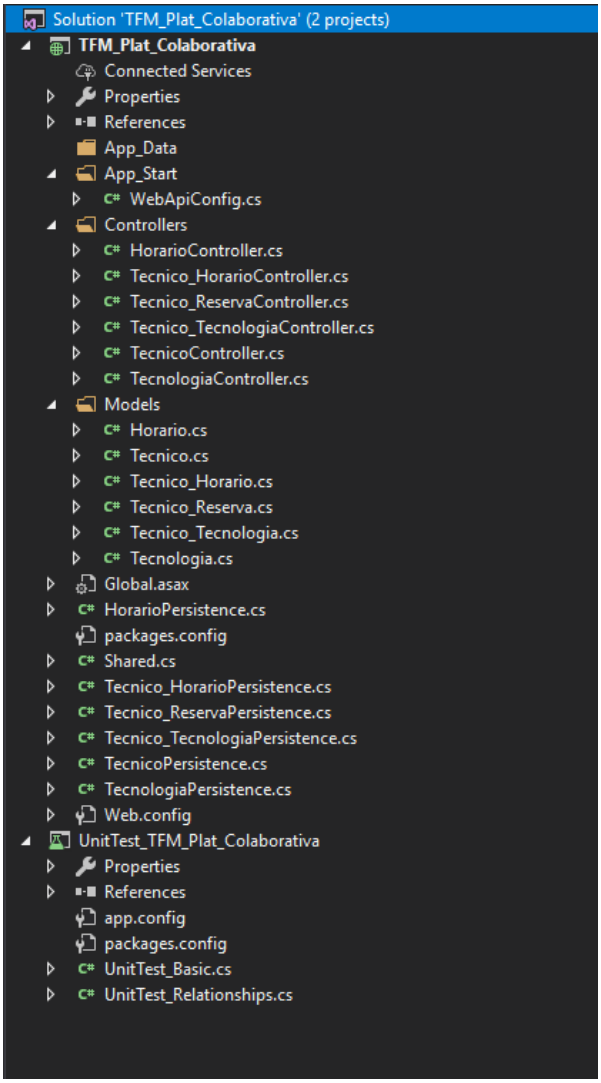


Fig 22: Solución REST API

5.6.2.1 Modelos

Son las clases utilizadas para recuperar y almacenar el estado de las diferentes propiedades (campos) que componen dichos modelos, como se podrá observar, tienen una correspondencia bastante directa con los tipos de datos utilizados a nivel de base de datos (cosa que es lógica) ya que el modelo es la representación del dato a nivel API mientras que la tabla en BBDD es la representación del almacenamiento de dicho dato.

Horario: Modelo que gestiona las propiedades de la entidad Horario.

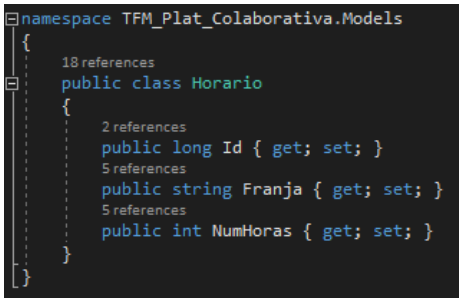


Fig 23: Modelo Horario

Técnico: Modelo que gestiona las propiedades de la entidad Técnico.

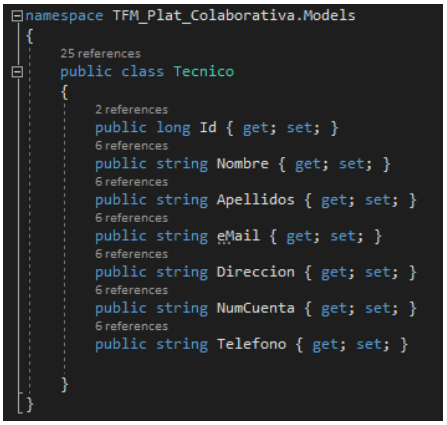


Fig 24: Modelo Técnico

Tecnología: Modelo que gestiona las propiedades de la entidad Tecnología

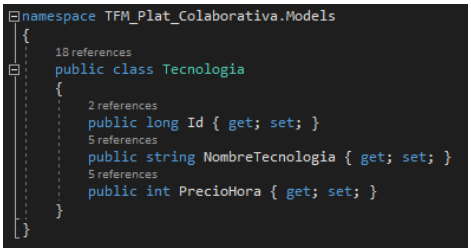


Fig 25: Modelo Tecnología

Técnico_Horario: Modelo que gestiona la entidad de relación entre los técnicos y los horarios

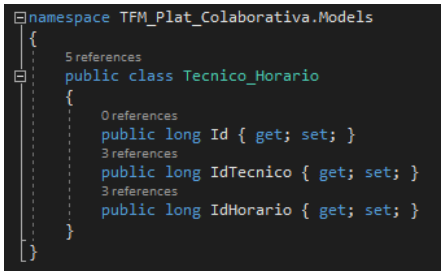


Fig 26: Modelo Tecnico_Horario

Técnico_Reserva: Modelo que gestiona la entidad de relación entre técnicos y sus reservas (un técnico se reserva para una tecnología en un horario determinado).

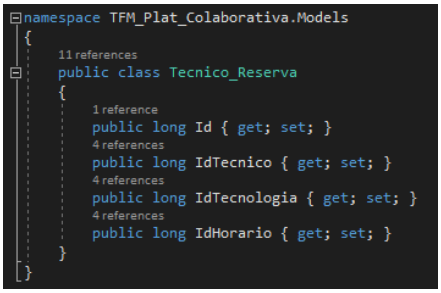


Fig 27: Modelo Técnico_Reserva

Técnico_Tecnología: Modelo que gestiona la entidad de relación entre técnicos y tecnologías.

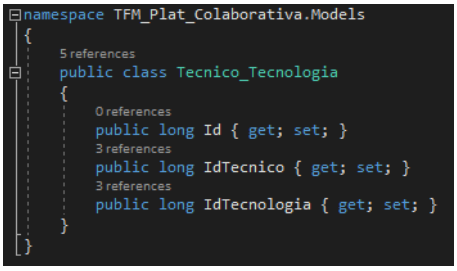


Fig 28: Modelo Técnico_Tecnología

5.6.2.2 Controladores

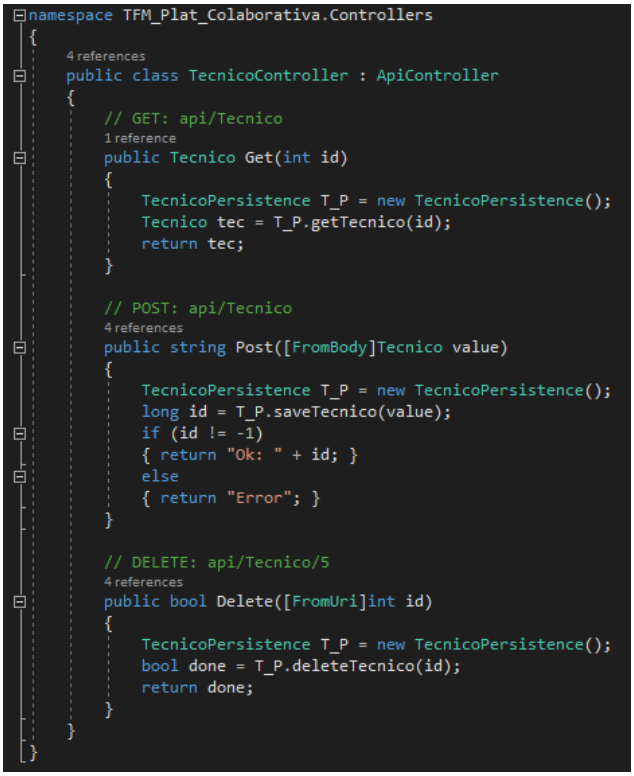
Son los encargados de recibir la petición del cliente de nuestra API REST, hacer comprobaciones de la robustez de la petición (por ejemplo si se trata de reservar un técnico con un Id que no existe en el sistema) y llamar a la clase Persistence correspondiente para interactuar con la información a nivel de BBDD.

HorarioController: Es el controlador de la clase *Horario* e implementa los métodos GET (devuelve un tipo de dato *Horario*), POST (recibe un tipo de dato *Horario* en formato JSON), y DELETE (recibe el Id del horario en la URL de llamada).

```
namespace TFM_Plat_Colaborativa.Controllers
{
    3 references
    public class HorarioController : ApiController
    {
        // GET: api/Horario/5
        1 reference
        public Horario Get(int id)
        {
            HorarioPersistence H_P = new HorarioPersistence();
            Horario hor = H_P.getHorario(id);
            return hor;
        }

        // POST: api/Horario
        3 references
        public string Post([FromBody]Horario value)
        {
            HorarioPersistence H_P = new HorarioPersistence();
            long id = H_P.saveHorario(value);
            if (id != -1)
            { return "Ok: " + id; }
            else
            { return "Error"; }
        }

        // DELETE: api/Horario/5
        3 references
        public bool Delete([FromUri]int id)
        {
            HorarioPersistence H_P = new HorarioPersistence();
            bool done = H_P.deleteHorario(id);
            return done;
        }
    }
}
```

Fig 29: HorarioController

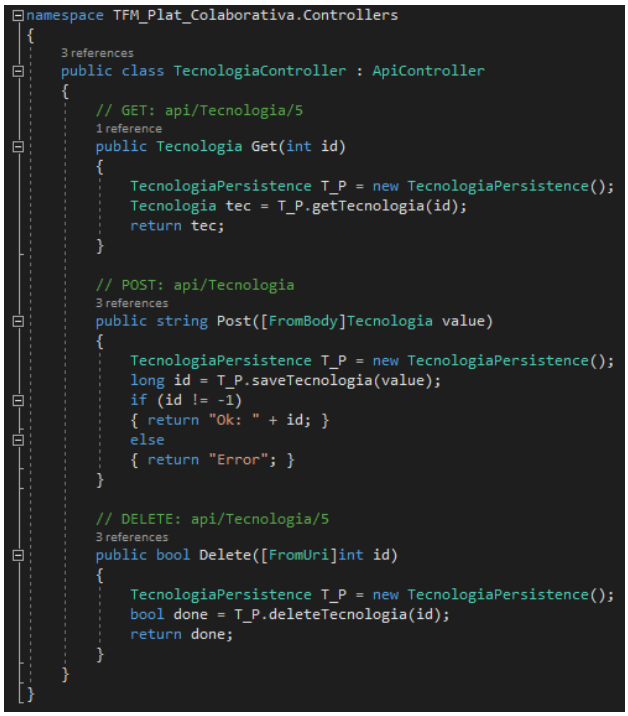
TecnicoController: Es el controlador de la clase *Técnico* e implementa los métodos GET (devuelve un tipo de dato *Técnico*), POST (recibe un tipo de dato *Técnico* en formato JSON), y DELETE (recibe el Id del técnico en la URL de llamada).



```
namespace TFM_Plat_Colaborativa.Controllers
{
    4 references
    public class TecnicoController : ApiController
    {
        // GET: api/Tecnico
        1 reference
        public Tecnico Get(int id)
        {
            TecnicoPersistence T_P = new TecnicoPersistence();
            Tecnico tec = T_P.getTecnico(id);
            return tec;
        }

        // POST: api/Tecnico
        4 references
        public string Post([FromBody]Tecnico value)
        {
            TecnicoPersistence T_P = new TecnicoPersistence();
            long id = T_P.saveTecnico(value);
            if (id != -1)
            { return "Ok: " + id; }
            else
            { return "Error"; }
        }

        // DELETE: api/Tecnico/5
        4 references
        public bool Delete([FromUri]int id)
        {
            TecnicoPersistence T_P = new TecnicoPersistence();
            bool done = T_P.deleteTecnico(id);
            return done;
        }
    }
}
```

Fig 30: TecnicoController

TecnologiaController: Es el controlador de la clase *Tecnología* e implementa los métodos GET (devuelve un tipo de dato *Tecnología*), POST (recibe un tipo de dato *Tecnología* en formato JSON), y DELETE (recibe el Id de la tecnología en la URL de llamada).



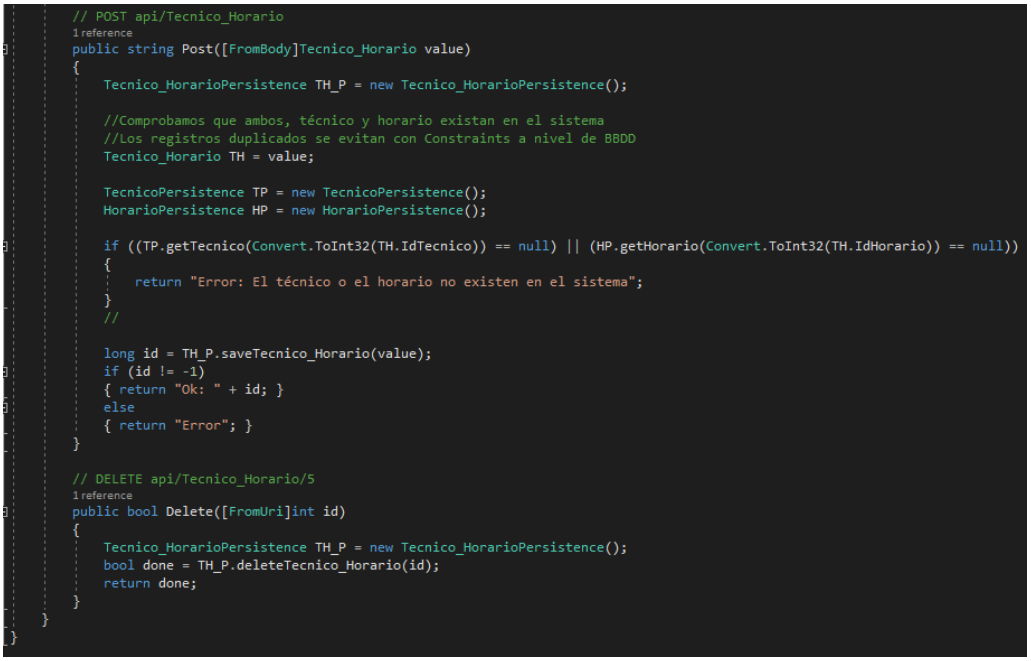
```
namespace TFM_Plat_Colaborativa.Controllers
{
    3 references
    public class TecnologiaController : ApiController
    {
        // GET: api/Tecnologia/5
        1 reference
        public Tecnologia Get(int id)
        {
            TecnologiaPersistence T_P = new TecnologiaPersistence();
            Tecnologia tec = T_P.getTecnologia(id);
            return tec;
        }

        // POST: api/Tecnologia
        3 references
        public string Post([FromBody]Tecnologia value)
        {
            TecnologiaPersistence T_P = new TecnologiaPersistence();
            long id = T_P.saveTecnologia(value);
            if (id != -1)
            { return "Ok: " + id; }
            else
            { return "Error"; }
        }

        // DELETE: api/Tecnologia/5
        3 references
        public bool Delete([FromUri]int id)
        {
            TecnologiaPersistence T_P = new TecnologiaPersistence();
            bool done = T_P.deleteTecnologia(id);
            return done;
        }
    }
}
```

Fig 31: TecnologiaController

Tecnico_HorarioController: Es el controlador de la clase *Tecnico_Horario* e implementa los métodos GET (devuelve un listado de tipo *string* en el que se incluyen todo los técnicos y sus horarios asociados en el siguiente formato "Id|IdTecnico|IdHorario"), (GET)GetTecnicosForHorario (recibe el identificador de horario y devuelve un array de datos tipo *Técnico* con todos los técnicos para ese horario), (GET)GetHorariosForTecnico (recibe el identificador de un técnico y devuelve un array de datos tipo *Horario* con todos los horarios para ese técnico), POST (recibe un tipo de dato *Tecnico_Horario* comprueba que ambos identificadores existan en el sistema y devuelve un tipo de dato *string* con el resultado de la operación), y DELETE (recibe el Id de Tecnico_Horario en la URL de llamada).

```
namespace TFM_Plat_Colaborativa.Controllers
{
    [RoutePrefix("api/Tecnico_Horario")]
    1 reference
    public class Tecnico_HorarioController : ApiController
    {
        // GET api/Tecnico_Horario
        1 reference
        public IEnumerable<string> Get()
        {
            Tecnico_HorarioPersistence TH_P = new Tecnico_HorarioPersistence();
            string[] result = TH_P.getAllTecnico_Horario();
            if (result != null) return result;
            else return null;
        }

        // GET: api/Tecnico_Horario/GetTecnicosForHorario?IdHorario=5
        [Route("GetTecnicosForHorario")]
        0 references
        public Tecnico[] GetTecnicosForHorario(int IdHorario)
        {
            Tecnico_HorarioPersistence TH_P = new Tecnico_HorarioPersistence();
            Tecnico[] result = TH_P.getTecnicosForHorario(IdHorario);
            if (result != null) return result;
            else return null;
        }

        // GET: api/Tecnico_Horario/GetHorariosForTecnico?Id=5
        [Route("GetHorariosForTecnico")]
        0 references
        public Horario[] GetHorariosForTecnico(int idTecnico)
        {
            Tecnico_HorarioPersistence TH_P = new Tecnico_HorarioPersistence();
            Horario[] result = TH_P.getHorariosForTecnico(idTecnico);
            if (result != null) return result;
            else return null;
        }
```

Fig 32.1: Tecnico_HorarioController parte 1.



```
        // POST api/Tecnico_Horario
        1 reference
        public string Post([FromBody]Tecnico_Horario value)
        {
            Tecnico_HorarioPersistence TH_P = new Tecnico_HorarioPersistence();

            //Comprobamos que ambos, técnico y horario existan en el sistema
            //Los registros duplicados se evitan con Constraints a nivel de BBDD
            Tecnico_Horario TH = value;

            TecnicoPersistence TP = new TecnicoPersistence();
            HorarioPersistence HP = new HorarioPersistence();

            if ((TP.getTecnico(Convert.ToInt32(TH.IdTecnico)) == null) || (HP.getHorario(Convert.ToInt32(TH.IdHorario)) == null))
            {
                return "Error: El técnico o el horario no existen en el sistema";
            }
            //

            long id = TH_P.saveTecnico_Horario(value);
            if (id != -1)
            { return "Ok: " + id; }
            else
            { return "Error"; }
        }

        // DELETE api/Tecnico_Horario/5
        1 reference
        public bool Delete([FromUri]int id)
        {
            Tecnico_HorarioPersistence TH_P = new Tecnico_HorarioPersistence();
            bool done = TH_P.deleteTecnico_Horario(id);
            return done;
        }
    }
}
```

Fig 32.2: Tecnico_HorarioController parte 2

Técnico_TecnologíaController: Es el controlador de la clase *Tecnico_Tecnología* e implementa los métodos GET (devuelve un listado de tipo *string* en el que se incluyen todo los técnicos y sus tecnologías asociadas en el siguiente formato "Id|IdTecnico|IdTecnologia"), (GET)GetTecnicosForTecnologia (recibe el identificador de tecnología y devuelve un array de datos tipo *Técnico* con todos los técnicos para esa tecnología), (GET)GetTecnologiasForTecnico (recibe el identificador de un técnico y devuelve un array de datos tipo *Tecnología* con todas las tecnologías para ese técnico), POST (recibe un tipo de dato *Tecnico_Tecnologia* comprueba que ambos identificadores existan en el sistema y devuelve un tipo de dato *string* con el resultado de la operación), y DELETE (recibe el Id de Tecnico_Tecnologia en la URL de llamada).

```
namespace TFM_Plat_Colaborativa.Controllers
{
    [RoutePrefix("api/Tecnico_Tecnologia")]
    1 reference
    public class Tecnico_TecnologiaController : ApiController
    {
        // GET: api/Tecnico_Tecnologia
        1 reference
        public IEnumerable<string> Get()
        {
            Tecnico_TecnologiaPersistence TT_P = new Tecnico_TecnologiaPersistence();
            string[] result = TT_P.getAllTecnicos_Tecnologia();
            if (result != null) return result;
            else return null;
        }

        // GET: api/Tecnico_Tecnologia/GetTecnicosForTecnologia?idTecnologia=5
        [Route("GetTecnicosForTecnologia")]
        0 references
        public Tecnico[] GetTecnicosForTecnologia(int idTecnologia)
        {
            Tecnico_TecnologiaPersistence TT_P = new Tecnico_TecnologiaPersistence();
            Tecnico[] result = TT_P.getTecnicosForTecnologia(idTecnologia);
            if (result != null) return result;
            else return null;
        }

        // GET: api/Tecnico_Tecnologia/GetTecnologiasForTecnico?idTecnico=5
        [Route("GetTecnologiasForTecnico")]
        0 references
        public Tecnologia[] GetTecnologiasForTecnico(int idTecnico)
        {
            Tecnico_TecnologiaPersistence TT_P = new Tecnico_TecnologiaPersistence();
            Tecnologia[] result = TT_P.getTecnologiasForTecnico(idTecnico);
            if (result != null) return result;
            else return null;
        }
```

Fig 33.1: Tecnico_TecnologiaController parte 1

```
        // POST: api/Tecnico_Tecnologia
        1 reference
        public string Post([FromBody]Tecnico_Tecnologia value)
        {
            Tecnico_TecnologiaPersistence TT_P = new Tecnico_TecnologiaPersistence();
            //Comprobamos que ambos, técnico y técnología existan en el sistema
            Tecnico_Tecnologia TT = value;

            TecnicoPersistence TP = new TecnicoPersistence();
            TecnologiaPersistence T_P = new TecnologiaPersistence();

            if ((TP.getTecnico(Convert.ToInt32(TT.IdTecnico)) == null) || (T_P.getTecnologia(Convert.ToInt32(TT.IdTecnologia)) == null))
            {
                return "Error: El técnico o el horario no existen en el sistema";
            }

            long id = TT_P.saveTecnico_Tecnologia(value);
            if (id != -1)
            { return "Ok: " + id; }
            else
            { return "Error"; }
        }

        // DELETE: api/Tecnico_Tecnologia/5
        1 reference
        public bool Delete([FromUri]int id)
        {
            Tecnico_TecnologiaPersistence TT_P = new Tecnico_TecnologiaPersistence();
            bool done = TT_P.deleteTecnico_Tecnologia(id);
            return done;
        }
    }
}
```

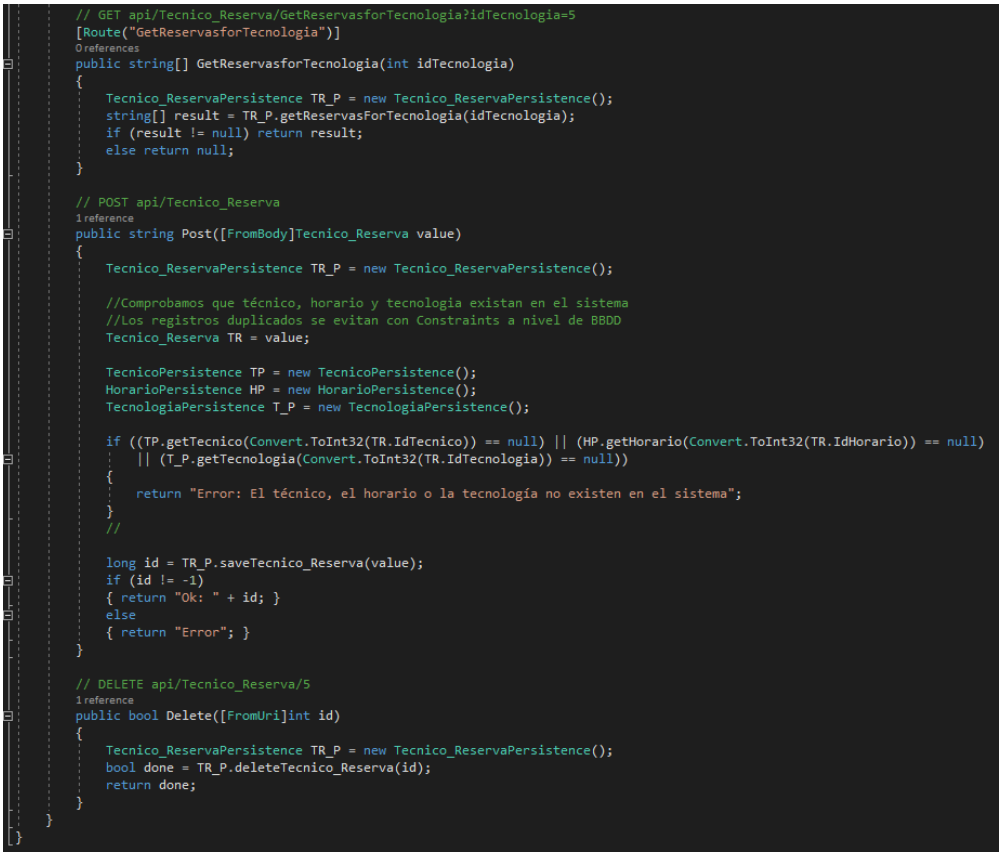
Fig 33.2: Tecnico_TecnologiaController parte 2

Técnico_ReservaController: Es el controlador de la clase *Tecnico_Reserva* e implementa los métodos GET (devuelve un listado de tipo *string* en el que se incluyen todo los técnicos reservados, en qué tecnologías y para qué horarios en el siguiente formato "Id|IdTecnico|IdTecnologia|IdHorario"), (GET)GetTecnicoReservaXML (devuelve un array del tipo *Tecnico_Reserva* con la peculiaridad de que está

pensado para recibir una llamada en cuya cabecera se indique que la respuesta debe ser XML (no JSON), en dicho array se incluyen todos los técnicos que están reservados en una tecnología concreta para un horario concreto), (GET)GetReservasforTecnologia (recibe el identificador de tecnología y devuelve un array de datos tipo *Técnico_Reserva* con todos los técnicos reservados para esa tecnología), (GET)GetReservasforTecnico (recibe el identificador de un técnico y devuelve un array de datos tipo *Técnico_Reserva* con todas las reservas para ese técnico), (GET)GetReservasforHorario (recibe el identificador de horario y devuelve un array de datos tipo *Técnico_Reserva* con todas las reservas para ese horario), POST (recibe un tipo de dato *Tecnico_Reserva* comprueba que todos los identificadores existan en el sistema y devuelve un tipo de dato *string* con el resultado de la operación), y DELETE (recibe el Id de Tecnico_Reserva en la URL de llamada).

```
namespace TFM_Plat_Colaborativa.Controllers
{
    [RoutePrefix("api/Tecnico_Reserva")]
    1 reference
    public class Tecnico_ReservaController : ApiController
    {
        // GET api/Tecnico_Reserva
        1 reference
        public IEnumerable<string> Get()
        {
            Tecnico_ReservaPersistence TR_P = new Tecnico_ReservaPersistence();
            string[] result = TR_P.getAllTecnico_Reserva_string();
            if (result != null) return result;
            else return null;
        }

        // GET api/Tecnico_Reserva/GetTecnicoReservaXML
        [Route("GetTecnicoReservaXML")]
        0 references
        public Tecnico_Reserva[] GetTecnicoReservaXML()
        {
            Tecnico_ReservaPersistence TR_P = new Tecnico_ReservaPersistence();
            Tecnico_Reserva[] TR = TR_P.getAllTecnico_Reserva_TR();
            if (TR != null) return TR;
            else return null;
        }

        // GET api/Tecnico_Reserva/GetReservasforTecnico?idTecnico=5
        [Route("GetReservasforTecnico")]
        0 references
        public string[] GetReservasforTecnico(int idTecnico)
        {
            Tecnico_ReservaPersistence TR_P = new Tecnico_ReservaPersistence();
            string[] result = TR_P.getReservasForTecnico(idTecnico);
            if (result != null) return result;
            else return null;
        }

        // GET api/Tecnico_Reserva/GetReservasforHorario?idHorario=5
        [Route("GetReservasforHorario")]
        0 references
        public string[] GetReservasforHorario(int idHorario)
        {
            Tecnico_ReservaPersistence TR_P = new Tecnico_ReservaPersistence();
            string[] result = TR_P.getReservasForHorario(idHorario);
            if (result != null) return result;
            else return null;
        }
```

Fig 34.1: Tecnico_ReservaController parte 1



```
        // GET api/Tecnico_Reserva/GetReservasforTecnologia?idTecnologia=5
        [Route("GetReservasforTecnologia")]
        0 references
        public string[] GetReservasforTecnologia(int idTecnologia)
        {
            Tecnico_ReservaPersistence TR_P = new Tecnico_ReservaPersistence();
            string[] result = TR_P.getReservasForTecnologia(idTecnologia);
            if (result != null) return result;
            else return null;
        }

        // POST api/Tecnico_Reserva
        1 reference
        public string Post([FromBody]Tecnico_Reserva value)
        {
            Tecnico_ReservaPersistence TR_P = new Tecnico_ReservaPersistence();

            //Comprobamos que técnico, horario y tecnologia existan en el sistema
            //Los registros duplicados se evitan con Constraints a nivel de BBDD
            Tecnico_Reserva TR = value;

            TecnicoPersistence TP = new TecnicoPersistence();
            HorarioPersistence HP = new HorarioPersistence();
            TecnologiaPersistence T_P = new TecnologiaPersistence();

            if ((TP.getTecnico(Convert.ToInt32(TR.IdTecnico)) == null) || (HP.getHorario(Convert.ToInt32(TR.IdHorario)) == null)
                || (T_P.getTecnologia(Convert.ToInt32(TR.IdTecnologia)) == null))
            {
                return "Error: El técnico, el horario o la tecnología no existen en el sistema";
            }
            //

            long id = TR_P.saveTecnico_Reserva(value);
            if (id != -1)
            { return "Ok: " + id; }
            else
            { return "Error"; }
        }

        // DELETE api/Tecnico_Reserva/5
        1 reference
        public bool Delete([FromUri]int id)
        {
            Tecnico_ReservaPersistence TR_P = new Tecnico_ReservaPersistence();
            bool done = TR_P.deleteTecnico_Reserva(id);
            return done;
        }
    }
}
```

Fig 34.2: Tecnico_ReservaController parte 2

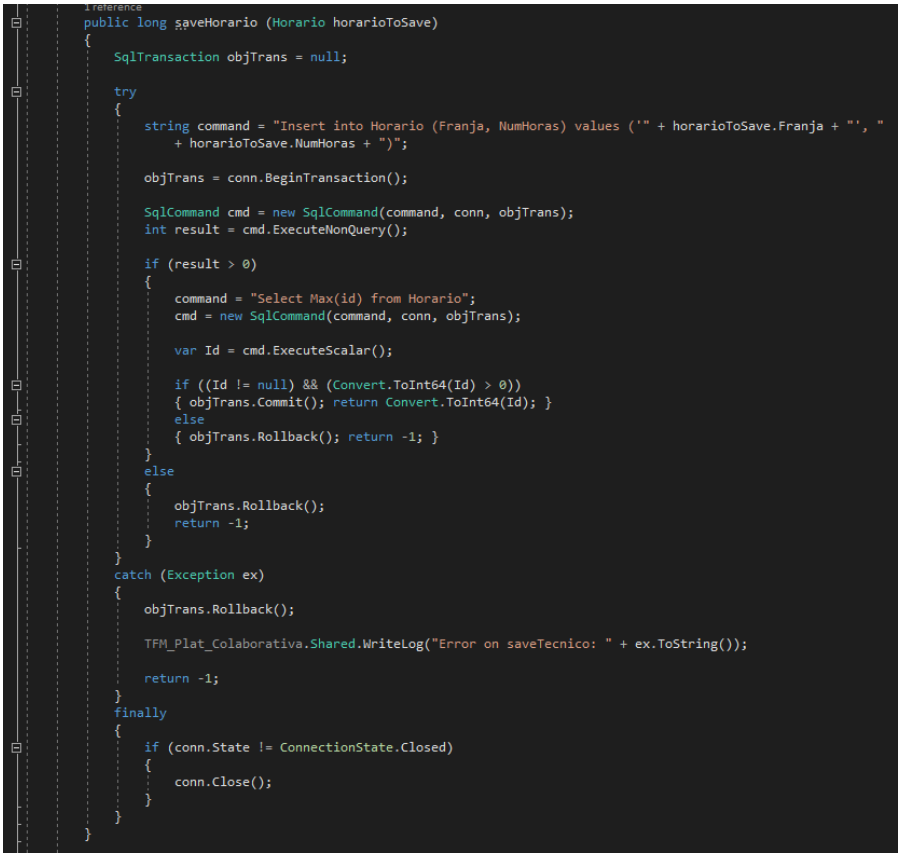
5.6.2.3 Persistence

Todas las clases que se agrupan dentro de *Persistence* son las encargadas de realizar las tareas de conexión y acceso al almacén de datos, en nuestro caso una BBDD SQL Server 2016. Lo bueno del uso de estas clases es que abstraemos al resto de la API de preocuparse de la gestión de dicho almacén de datos, y la independizamos del mismo. De esta manera, si cambiáramos el almacén, por ejemplo otra BBDD en Cloud, otro motor que no fuera SQL Server, etc… con cambiar la cadena de conexión en nuestro archivo de configuración bastaría y dicho cambio sería transparente y pasaría desapercibido para el resto de la API. Algo en común para todas las clases es que en su constructor instauran la conexión con la base de datos.

HorarioPersistence: Es la clase encargada de realizar el acceso a datos del modelo *Horario*. Implementa un constructor que es el que configura y abre la conexión con la base de datos, métodos saveHorario (para guardar un *Horario* en la tabla correspondiente, recibe un tipo de dato complejo *Horario*), getHorario (recibe un entero con el identificador del *Horario* del cual queremos devolver su información en una estructura compleja de tipo *Horario*), y deleteHorario (recibe el identificador entero del *Horario* que queremos eliminar y devuelve true/false según el resultado de la operación).

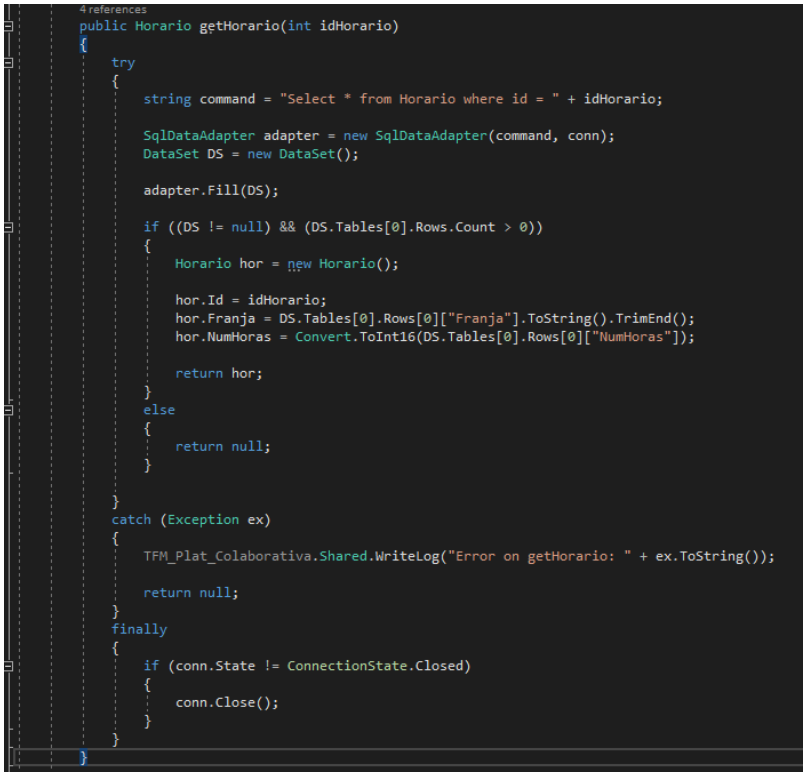
```
namespace TFM_Plat_Colaborativa
{
    13 references
    public class HorarioPersistence
    {
        private SqlConnection conn;

        6 references
        public HorarioPersistence()
        {
            string myConnectionString = null;

            try
            {
                //string myConnectionString = "Server = SAG-2J9W9H2; Database=TFM_Plat_Colaborativa; User Id = sa; Password = saIsaac;";
                if (ConfigurationManager.AppSettings["database"] != null)
                {
                    myConnectionString = ConfigurationManager.AppSettings["database"].ToString();
                }
                else
                {
                    myConnectionString = "Server = SAG-2J9W9H2; Database=TFM_Plat_Colaborativa; User Id = sa; Password = saIsaac;";
                }

                conn = new SqlConnection();
                conn.ConnectionString = myConnectionString;
                conn.Open();
            }
            catch (Exception ex)
            {
                TFM_Plat_Colaborativa.Shared.WriteLog("Error on HorarioPersistence: " + ex.ToString());
            }
        }
```

Fig 35.1: HorarioPersistence parte 1



```
1 reference
public long saveHorario (Horario horarioToSave)
{
    SqlTransaction objTrans = null;

    try
    {
        string command = "Insert into Horario (Franja, NumHoras) values ('" + horarioToSave.Franja + "', "
            + horarioToSave.NumHoras + ")";

        objTrans = conn.BeginTransaction();

        SqlCommand cmd = new SqlCommand(command, conn, objTrans);
        int result = cmd.ExecuteNonQuery();

        if (result > 0)
        {
            command = "Select Max(id) from Horario";
            cmd = new SqlCommand(command, conn, objTrans);

            var Id = cmd.ExecuteScalar();

            if ((Id != null) && (Convert.ToInt64(Id) > 0))
            { objTrans.Commit(); return Convert.ToInt64(Id); }
            else
            { objTrans.Rollback(); return -1; }
        }
        else
        {
            objTrans.Rollback();
            return -1;
        }
    }
    catch (Exception ex)
    {
        objTrans.Rollback();

        TFM_Plat_Colaborativa.Shared.WriteLog("Error on saveTecnico: " + ex.ToString());

        return -1;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

Fig 35.2: HorarioPersistence parte 2



```
4 references
public Horario getHorario(int idHorario)
{
    try
    {
        string command = "Select * from Horario where id = " + idHorario;

        SqlDataAdapter adapter = new SqlDataAdapter(command, conn);
        DataSet DS = new DataSet();

        adapter.Fill(DS);

        if ((DS != null) && (DS.Tables[0].Rows.Count > 0))
        {
            Horario hor = new Horario();

            hor.Id = idHorario;
            hor.Franja = DS.Tables[0].Rows[0]["Franja"].ToString().TrimEnd();
            hor.NumHoras = Convert.ToInt16(DS.Tables[0].Rows[0]["NumHoras"]);

            return hor;
        }
        else
        {
            return null;
        }

    }
    catch (Exception ex)
    {
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on getHorario: " + ex.ToString());

        return null;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

Fig 35.3: HorarioPersistence parte 3

```
        public bool deleteHorario(int idHorario)
        {
            SqlTransaction objTrans = null;

            try
            {
                string command = "Delete from Horario where id = " + idHorario;

                objTrans = conn.BeginTransaction();

                SqlCommand cmd = new SqlCommand(command, conn, objTrans);
                int result = cmd.ExecuteNonQuery();

                if (result > 0)
                {
                    objTrans.Commit();
                    return true;
                }
                else
                {
                    objTrans.Rollback();
                    return false;
                }
            }
            catch (Exception ex)
            {
                objTrans.Rollback();

                TFM_Plat_Colaborativa.Shared.WriteLog("Error on deleteHorario: " + ex.ToString());

                return false;
            }
            finally
            {
                if (conn.State != ConnectionState.Closed)
                {
                    conn.Close();
                }
            }
        }

    }
}
```

Fig 35.4: HorarioPersistence parte 4

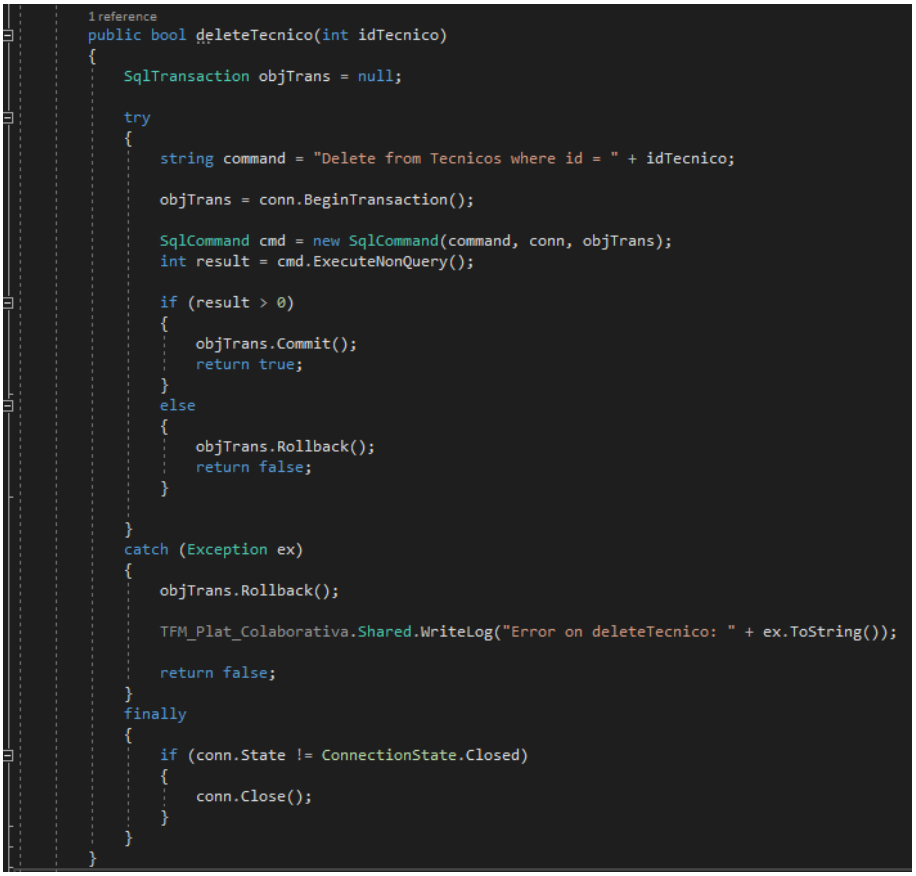
TecnicoPersistence: Es la clase encargada de realizar el acceso a datos del modelo *Técnico*. Implementa un constructor que es el que configura y abre la conexión con la base de datos, métodos saveTecnico (para guardar un *Técnico* en la tabla correspondiente, recibe un tipo de dato complejo *Técnico*), getTecnico (recibe un entero con el identificador del *Técnico* del cual queremos devolver su información en una estructura compleja de tipo *Técnico*), y deleteTecnico (recibe el identificador entero del *Técnico* que queremos eliminar y devuelve true/false según el resultado de la operación).

```
namespace TFM_Plat_Colaborativa
{
    17 references
    public class TecnicoPersistence
    {
        private SqlConnection conn;

        8 references
        public TecnicoPersistence()
        {
            string myConnectionString = null;

            try
            {
                //string myConnectionString = "Server = SAG-2J9W9H2; Database=TFM_Plat_Colaborativa; User Id = sa; Password = saIsaac;";
                if (ConfigurationManager.AppSettings["database"] != null)
                {
                    myConnectionString = ConfigurationManager.AppSettings["database"].ToString();
                }
                else
                {
                    myConnectionString = "Server = SAG-2J9W9H2; Database=TFM_Plat_Colaborativa; User Id = sa; Password = saIsaac;";
                }

                conn = new SqlConnection();
                conn.ConnectionString = myConnectionString;
                conn.Open();
            }
            catch (Exception ex)
            {
                TFM_Plat_Colaborativa.Shared.WriteLog("Error on TecnicoPersistence: " + ex.ToString());
            }

        }
```

Fig 36.1: TecnicoPersistence parte 1

```
        1 reference
        public long saveTecnico(Tecnico tecnicoToSave)
        {
            SqlTransaction objTrans = null;

            try
            {
                string command = "Insert into Tecnicos (Nombre, Apellidos, eMail, Direccion, NumCuenta, Telefono) values ('" + tecnicoToSave.Nombre + "', '"
                    + tecnicoToSave.Apellidos + "', '" + tecnicoToSave.eMail + "', '" + tecnicoToSave.Direccion + "', '" + tecnicoToSave.NumCuenta + "', '"
                    + tecnicoToSave.Telefono + "')";

                objTrans = conn.BeginTransaction();

                SqlCommand cmd = new SqlCommand(command, conn, objTrans);
                int result = cmd.ExecuteNonQuery();

                if (result > 0)
                {
                    command = "Select Max(id) from Tecnicos";
                    cmd = new SqlCommand(command, conn, objTrans);

                    var Id = cmd.ExecuteScalar();

                    if ((Id != null) && (Convert.ToInt64(Id) > 0))
                    { objTrans.Commit(); return Convert.ToInt64(Id); }
                    else
                    { objTrans.Rollback(); return -1; }
                }
                else
                {
                    objTrans.Rollback();
                    return -1;
                }
            }
            catch (Exception ex)
            {
                objTrans.Rollback();

                TFM_Plat_Colaborativa.Shared.WriteLog("Error on saveTecnico: " + ex.ToString());

                return -1;
            }
            finally
            {
                if (conn.State != ConnectionState.Closed)
                {
                    conn.Close();
                }
            }
        }
```

Fig 36.2: TecnicoPersistence parte 2



```
1 reference
public bool deleteTecnico(int idTecnico)
{
    SqlTransaction objTrans = null;

    try
    {
        string command = "Delete from Tecnicos where id = " + idTecnico;

        objTrans = conn.BeginTransaction();

        SqlCommand cmd = new SqlCommand(command, conn, objTrans);
        int result = cmd.ExecuteNonQuery();

        if (result > 0)
        {
            objTrans.Commit();
            return true;
        }
        else
        {
            objTrans.Rollback();
            return false;
        }

    }
    catch (Exception ex)
    {
        objTrans.Rollback();

        TFM_Plat_Colaborativa.Shared.WriteLog("Error on deleteTecnico: " + ex.ToString());

        return false;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

Fig 36.3: TecnicoPersistence parte 3

```
6 references
public Tecnico getTecnico(int idTecnico)
{
    try
    {
        string command = "Select * from Tecnicos where id = " + idTecnico;

        SqlDataAdapter adapter = new SqlDataAdapter(command, conn);
        DataSet DS = new DataSet();

        adapter.Fill(DS);

        if ((DS != null) && (DS.Tables[0].Rows.Count > 0))
        {
            Tecnico tec = new Tecnico();

            tec.Id = idTecnico;
            tec.Nombre = DS.Tables[0].Rows[0]["Nombre"].ToString().TrimEnd();
            tec.Apellidos = DS.Tables[0].Rows[0]["Apellidos"].ToString().TrimEnd();
            tec.eMail = DS.Tables[0].Rows[0]["eMail"].ToString().TrimEnd();
            tec.Direccion = DS.Tables[0].Rows[0]["Direccion"].ToString().TrimEnd();
            tec.NumCuenta = DS.Tables[0].Rows[0]["NumCuenta"].ToString().TrimEnd();
            tec.Telefono = DS.Tables[0].Rows[0]["Telefono"].ToString().TrimEnd();

            return tec;
        }
        else
        {
            return null;
        }

    }
    catch (Exception ex)
    {
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on getTecnico: " + ex.ToString());
        return null;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
}
}
```

Fig 36.4: TecnicoPersistence parte 4

TecnologiaPersistence: Es la clase encargada de realizar el acceso a datos del modelo *Tecnología*. Implementa un constructor que es el que configura y abre la conexión con la base de datos, métodos saveTecnologia (para guardar una *Tecnología* en la tabla correspondiente, recibe un tipo de dato complejo *Tecnología*), getTecnología (recibe un entero con el identificador de la *Tecnología* de la cual queremos devolver su información en una estructura compleja de tipo *Tecnología*), y deleteTecnología (recibe el identificador entero de la *Tecnología* que queremos eliminar y devuelve true/false según el resultado de la operación).

```
namespace TFM_Plat_Colaborativa
{
    13 references
    public class TecnologiaPersistence
    {
        private SqlConnection conn;

        6 references
        public TecnologiaPersistence()
        {
            string myConnectionString = null;

            try
            {
                //string myConnectionString = "Server = SAG-2J9W9H2; Database=TFM_Plat_Colaborativa; User Id = sa; Password = saIsaac;";
                if (ConfigurationManager.AppSettings["database"] != null)
                {
                    myConnectionString = ConfigurationManager.AppSettings["database"].ToString();
                }
                else
                {
                    myConnectionString = "Server = SAG-2J9W9H2; Database=TFM_Plat_Colaborativa; User Id = sa; Password = saIsaac;";
                }

                conn = new SqlConnection();
                conn.ConnectionString = myConnectionString;
                conn.Open();
            }
            catch (Exception ex)
            {
                TFM_Plat_Colaborativa.Shared.WriteLog("Error on TecnologiaPersistence: " + ex.ToString());
            }
        }
```

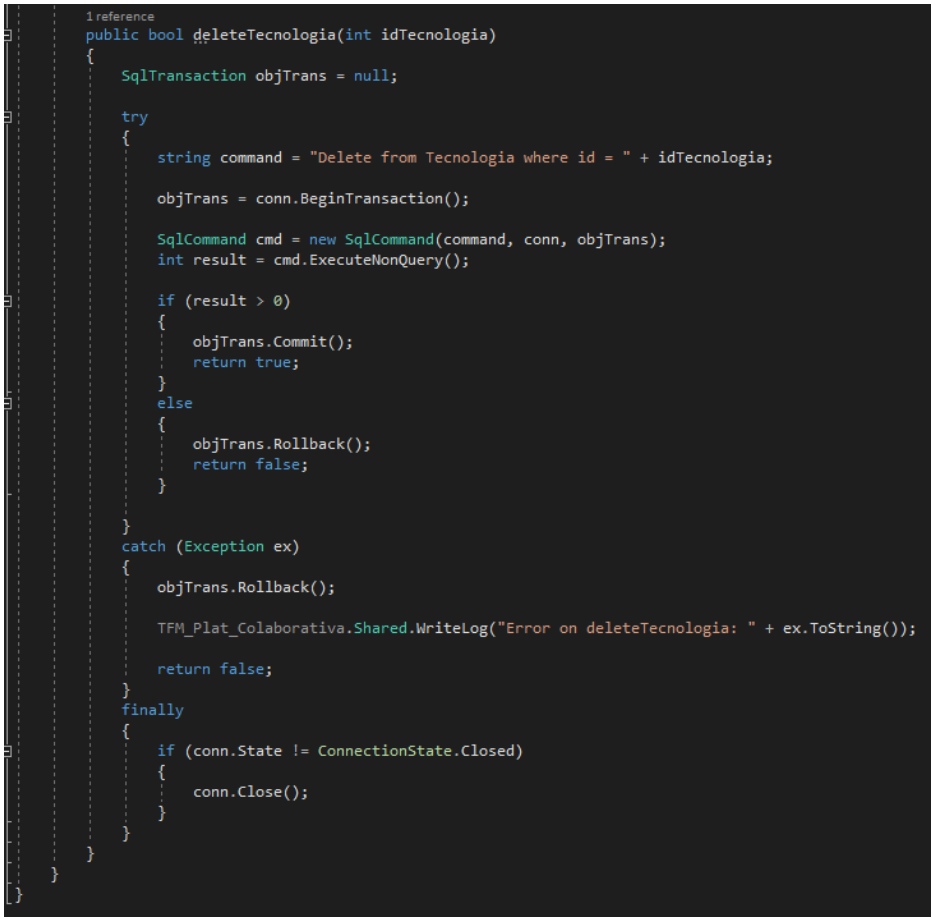
Fig 37.1: TecnologiaPersistence parte 1

```
        1 reference
        public long saveTecnologia(Tecnologia tecnologiaToSave)
        {
            SqlTransaction objTrans = null;

            try
            {

                string command = "Insert into Tecnologia (Tecnologia, PrecioHora) values ('" + tecnologiaToSave.NombreTecnologia + "', "
                    + tecnologiaToSave.PrecioHora + ")";

                objTrans = conn.BeginTransaction();

                SqlCommand cmd = new SqlCommand(command, conn, objTrans);
                int result = cmd.ExecuteNonQuery();

                if (result > 0)
                {
                    command = "Select Max(id) from Tecnologia";
                    cmd = new SqlCommand(command, conn, objTrans);

                    var Id = cmd.ExecuteScalar();

                    if ((Id != null) && (Convert.ToInt64(Id) > 0))
                    { objTrans.Commit(); return Convert.ToInt64(Id); }
                    else
                    { objTrans.Rollback(); return -1; }
                }
                else
                {
                    objTrans.Rollback();
                    return -1;
                }


            }
            catch (Exception ex)
            {
                objTrans.Rollback();

                TFM_Plat_Colaborativa.Shared.WriteLog("Error on saveTecnologia: " + ex.ToString());

                return -1;
            }
            finally
            {
                if (conn.State != ConnectionState.Closed)
                {
                    conn.Close();
                }
            }
        }
```

Fig 37.2: TecnologiaPersistence parte 2

```
4 references
public Tecnologia getTecnologia(int idTecnologia)
{
    try
    {
        string command = "Select * from Tecnologia where id = " + idTecnologia;

        SqlDataAdapter adapter = new SqlDataAdapter(command, conn);
        DataSet DS = new DataSet();

        adapter.Fill(DS);

        if ((DS != null) && (DS.Tables[0].Rows.Count > 0))
        {
            Tecnologia tec = new Tecnologia();

            tec.Id = idTecnologia;
            tec.NombreTecnologia = DS.Tables[0].Rows[0]["Tecnologia"].ToString().TrimEnd();
            tec.PrecioHora = Convert.ToInt16(DS.Tables[0].Rows[0]["PrecioHora"]);

            return tec;
        }
        else
        {
            return null;
        }

    }
    catch (Exception ex)
    {
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on getTecnologia: " + ex.ToString());

        return null;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

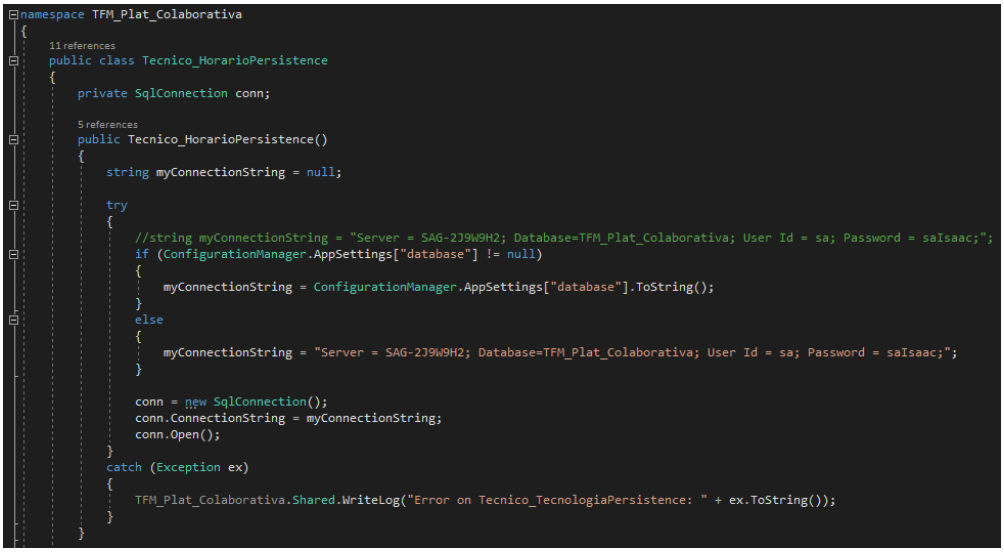
Fig 37.3: TecnologiaPersistence parte 3



```
1 reference
public bool deleteTecnologia(int idTecnologia)
{
    SqlTransaction objTrans = null;

    try
    {
        string command = "Delete from Tecnologia where id = " + idTecnologia;

        objTrans = conn.BeginTransaction();

        SqlCommand cmd = new SqlCommand(command, conn, objTrans);
        int result = cmd.ExecuteNonQuery();

        if (result > 0)
        {
            objTrans.Commit();
            return true;
        }
        else
        {
            objTrans.Rollback();
            return false;
        }

    }
    catch (Exception ex)
    {
        objTrans.Rollback();

        TFM_Plat_Colaborativa.Shared.WriteLog("Error on deleteTecnologia: " + ex.ToString());

        return false;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
}
}
```

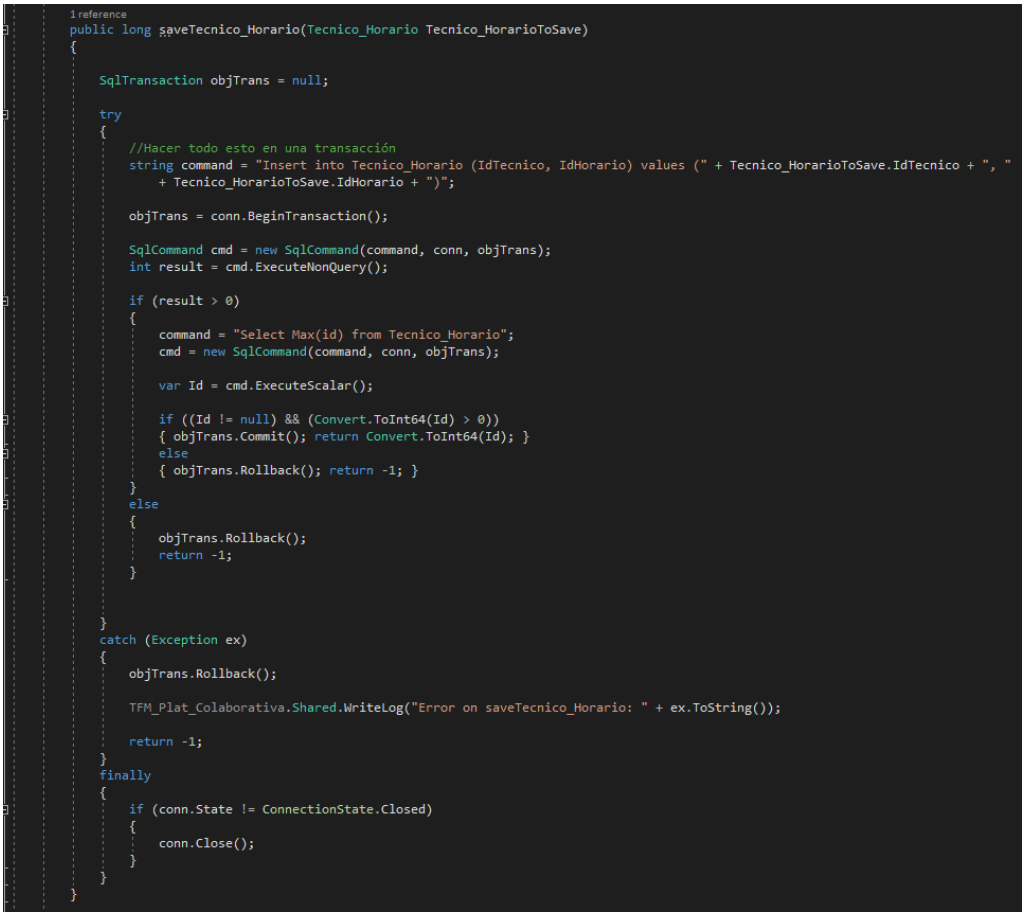
Fig 37.4: TecnologiaPersistence parte 4

Tecnico_HorarioPersistence: Es la clase encargada de realizar el acceso a datos del modelo *Tecnico_Horario*. Como el resto de clases *Persistence* implementa un constructor que se encarga de instaurar y abrir la conexión con la base de datos. Entre sus métodos encontramos saveTecnico_Horario (recibe un tipo de dato complejo *Tecnico_Horario*, lo guarda en la tabla correspondiente, y devuelve el identificador del mismo en caso de éxito), getAllTecnico_Horario (devuelve un *array* con los identificadores de todas las relaciones entre *Técnico* y *Horario)*, getTecnicosForHorario (recibe el identificador de *Horario* y devuelve un array del tipo *Técnico* con todos los técnicos asociados a ese horario), getHorariosForTecnico (recibe el identificador de *Técnico* y devuelve un array de tipo *Horario* con todos los horarios asociados a ese técnico), por último deleteTecnico_Horario (recibe el identificador del *Tecnico_Horario* que queremos eliminar y devuelve un *boolean* con el resultado de la operación).



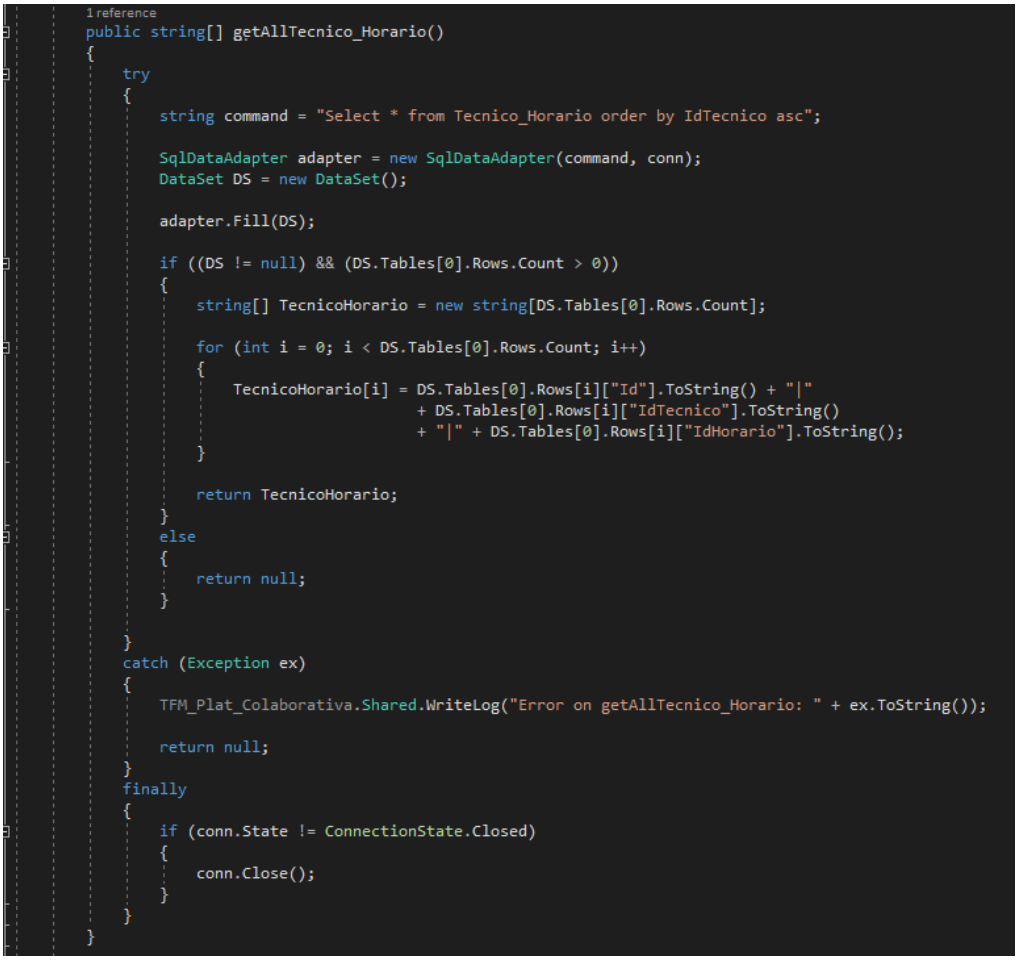
```
namespace TFM_Plat_Colaborativa
{
    11 references
    public class Tecnico_HorarioPersistence
    {
        private SqlConnection conn;

        5 references
        public Tecnico_HorarioPersistence()
        {
            string myConnectionString = null;

            try
            {
                //string myConnectionString = "Server = SAG-2J9W9H2; Database=TFM_Plat_Colaborativa; User Id = sa; Password = saIsaac;";
                if (ConfigurationManager.AppSettings["database"] != null)
                {
                    myConnectionString = ConfigurationManager.AppSettings["database"].ToString();
                }
                else
                {
                    myConnectionString = "Server = SAG-2J9W9H2; Database=TFM_Plat_Colaborativa; User Id = sa; Password = saIsaac;";
                }

                conn = new SqlConnection();
                conn.ConnectionString = myConnectionString;
                conn.Open();
            }
            catch (Exception ex)
            {
                TFM_Plat_Colaborativa.Shared.WriteLog("Error on Tecnico_TecnologiaPersistence: " + ex.ToString());
            }
        }
```

Fig 38.1: Tecnico_HorarioPersistence parte 1



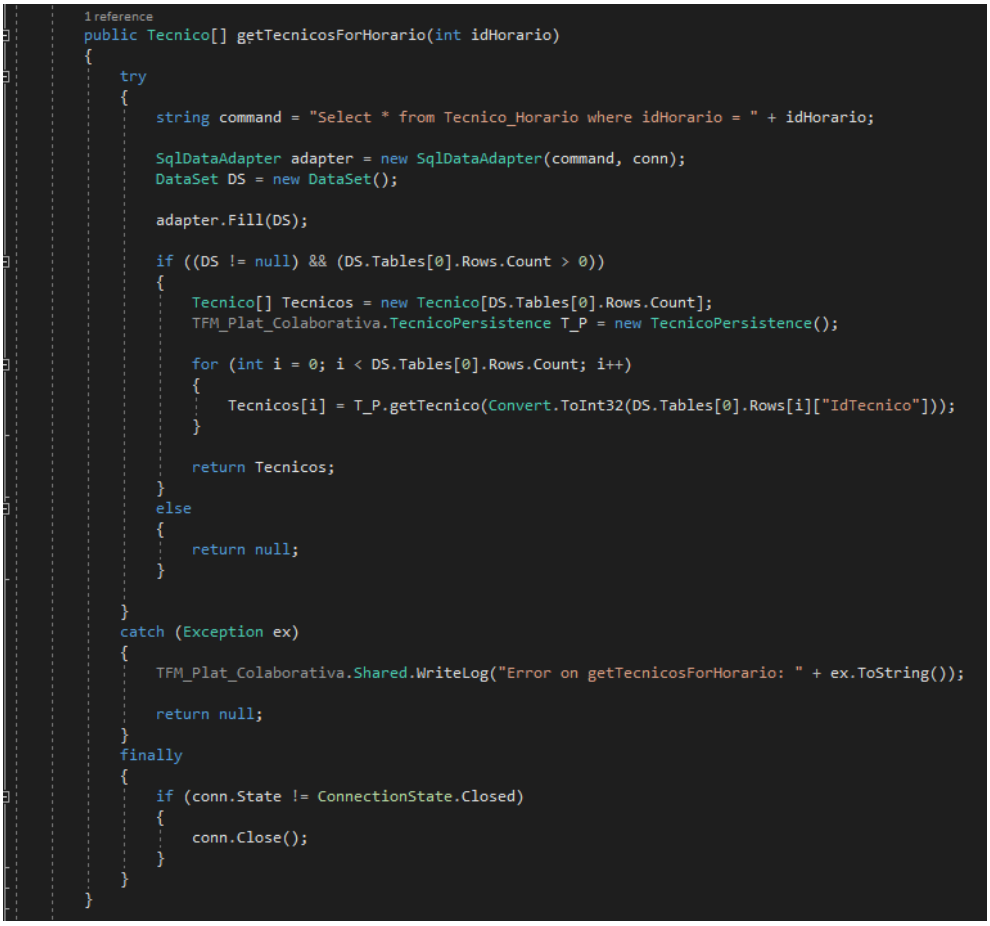
```
        1 reference
        public long saveTecnico_Horario(Tecnico_Horario Tecnico_HorarioToSave)
        {

            SqlTransaction objTrans = null;

            try
            {
                //Hacer todo esto en una transacción
                string command = "Insert into Tecnico_Horario (IdTecnico, IdHorario) values (" + Tecnico_HorarioToSave.IdTecnico + ", "
                    + Tecnico_HorarioToSave.IdHorario + ")";

                objTrans = conn.BeginTransaction();

                SqlCommand cmd = new SqlCommand(command, conn, objTrans);
                int result = cmd.ExecuteNonQuery();

                if (result > 0)
                {
                    command = "Select Max(id) from Tecnico_Horario";
                    cmd = new SqlCommand(command, conn, objTrans);

                    var Id = cmd.ExecuteScalar();

                    if ((Id != null) && (Convert.ToInt64(Id) > 0))
                    { objTrans.Commit(); return Convert.ToInt64(Id); }
                    else
                    { objTrans.Rollback(); return -1; }
                }
                else
                {
                    objTrans.Rollback();
                    return -1;
                }

            }
            catch (Exception ex)
            {
                objTrans.Rollback();

                TFM_Plat_Colaborativa.Shared.WriteLog("Error on saveTecnico_Horario: " + ex.ToString());

                return -1;
            }
            finally
            {
                if (conn.State != ConnectionState.Closed)
                {
                    conn.Close();
                }
            }
        }
```

Fig 38.2: Tecnico_HorarioPersistence parte 2



```
1 reference
public string[] getAllTecnico_Horario()
{
    try
    {
        string command = "Select * from Tecnico_Horario order by IdTecnico asc";

        SqlDataAdapter adapter = new SqlDataAdapter(command, conn);
        DataSet DS = new DataSet();

        adapter.Fill(DS);

        if ((DS != null) && (DS.Tables[0].Rows.Count > 0))
        {
            string[] TecnicoHorario = new string[DS.Tables[0].Rows.Count];

            for (int i = 0; i < DS.Tables[0].Rows.Count; i++)
            {
                TecnicoHorario[i] = DS.Tables[0].Rows[i]["Id"].ToString() + "|"
                                  + DS.Tables[0].Rows[i]["IdTecnico"].ToString()
                                  + "|" + DS.Tables[0].Rows[i]["IdHorario"].ToString();
            }

            return TecnicoHorario;
        }
        else
        {
            return null;
        }

    }
    catch (Exception ex)
    {
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on getAllTecnico_Horario: " + ex.ToString());

        return null;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

Fig 38.3: Tecnico_HorarioPersistence parte 3



```
1 reference
public Tecnico[] getTecnicosForHorario(int idHorario)
{
    try
    {
        string command = "Select * from Tecnico_Horario where idHorario = " + idHorario;

        SqlDataAdapter adapter = new SqlDataAdapter(command, conn);
        DataSet DS = new DataSet();

        adapter.Fill(DS);

        if ((DS != null) && (DS.Tables[0].Rows.Count > 0))
        {
            Tecnico[] Tecnicos = new Tecnico[DS.Tables[0].Rows.Count];
            TFM_Plat_Colaborativa.TecnicoPersistence T_P = new TecnicoPersistence();

            for (int i = 0; i < DS.Tables[0].Rows.Count; i++)
            {
                Tecnicos[i] = T_P.getTecnico(Convert.ToInt32(DS.Tables[0].Rows[i]["IdTecnico"]));
            }

            return Tecnicos;
        }
        else
        {
            return null;
        }

    }
    catch (Exception ex)
    {
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on getTecnicosForHorario: " + ex.ToString());

        return null;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

Fig 38.4: Tecnico_HorarioPersistence parte 4

```
1 reference
public Horario[] getHorariosForTecnico(int idTecnico)
{
    try
    {
        string command = "Select * from Tecnico_Horario where idTecnico = " + idTecnico;

        SqlDataAdapter adapter = new SqlDataAdapter(command, conn);
        DataSet DS = new DataSet();

        adapter.Fill(DS);

        if ((DS != null) && (DS.Tables[0].Rows.Count > 0))
        {
            Horario[] Horarios = new Horario[DS.Tables[0].Rows.Count];
            HorarioPersistence H_P = new HorarioPersistence();

            for (int i = 0; i < DS.Tables[0].Rows.Count; i++)
            {
                Horarios[i] = H_P.getHorario(Convert.ToInt32(DS.Tables[0].Rows[i]["IdHorario"]));
            }

            return Horarios;
        }
        else
        {
            return null;
        }

    }
    catch (Exception ex)
    {
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on getHorariosForTecnico: " + ex.ToString());

        return null;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

Fig 38.5: Tecnico_HorarioPersistence parte 5

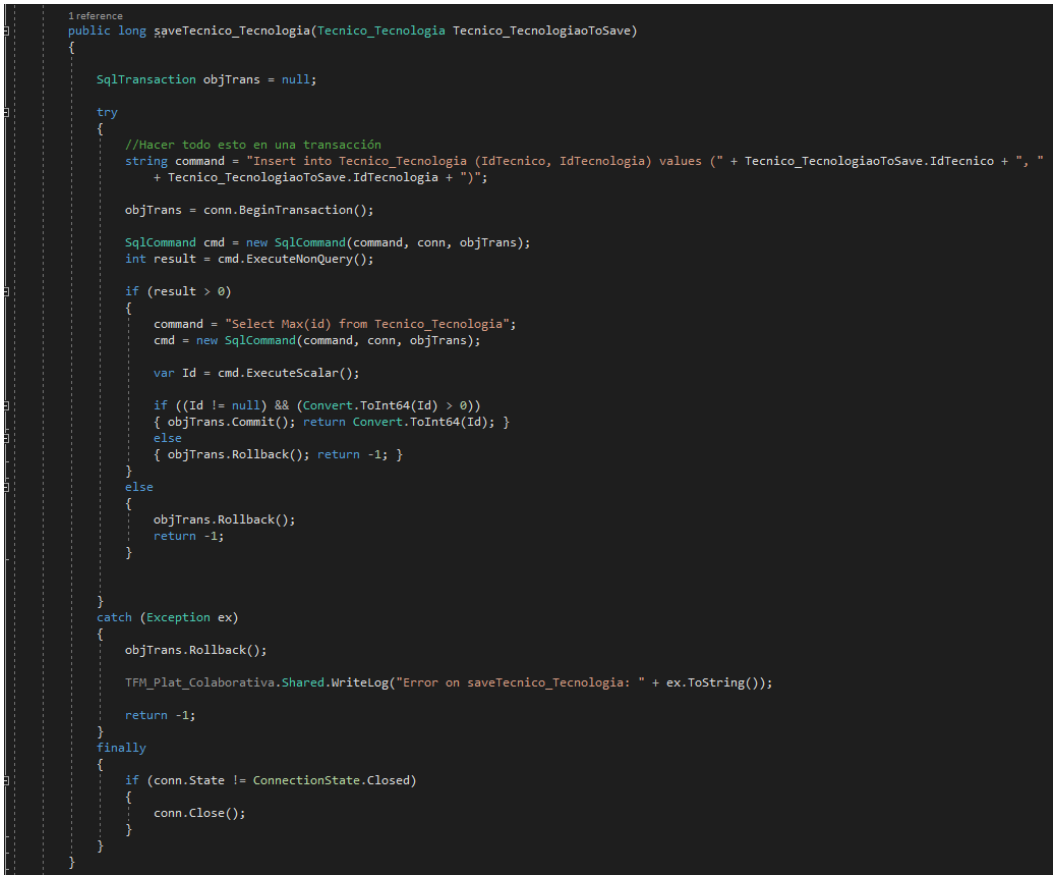
```
1 reference
public bool deleteTecnico_Horario(int idTecnico_Horario)
{
    SqlTransaction objTrans = null;

    try
    {
        string command = "Delete from Tecnico_Horario where id = " + idTecnico_Horario;

        objTrans = conn.BeginTransaction();

        SqlCommand cmd = new SqlCommand(command, conn, objTrans);
        int result = cmd.ExecuteNonQuery();

        if (result > 0)
        {
            objTrans.Commit();
            return true;
        }
        else
        {
            objTrans.Rollback();
            return false;
        }
    }
    catch (Exception ex)
    {
        objTrans.Rollback();
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on deleteTecnico_Tecnologia: " + ex.ToString());
        return false;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
}
```

Fig 38.6: Tecnico_HorarioPersistence parte 6

Tecnico_TecnologiaPersistence: Es la clase encargada de realizar el acceso a datos del modelo *Tecnico_Tecnología*. Como el resto de clases *Persistence* implementa un constructor que se encarga de instaurar y abrir la conexión con la base de datos. Entre sus métodos encontramos saveTecnico_Tecnologia (recibe un tipo de dato complejo *Tecnico_Tecnología*, lo guarda en la tabla correspondiente, y devuelve el identificador del mismo en caso de éxito), getAllTecnicos_Tecnologia (devuelve un *array* con los identificadores de todas las relaciones entre *Técnico* y *Tecnología)*, getTecnicosForTecnologia (recibe el identificador de *Tecnología* y devuelve un array del tipo *Técnico* con todas las tecnologías asociadas a ese técnico), getTecnologiasForTecnico (recibe el identificador de *Técnico* y devuelve un array de tipo *Tecnología* con todas las tecnologías asociadas a ese técnico), por último deleteTecnico_Tecnologia (recibe el identificador del *Tecnico_Tecnología* que queremos eliminar y devuelve un *boolean* con el resultado de la operación).

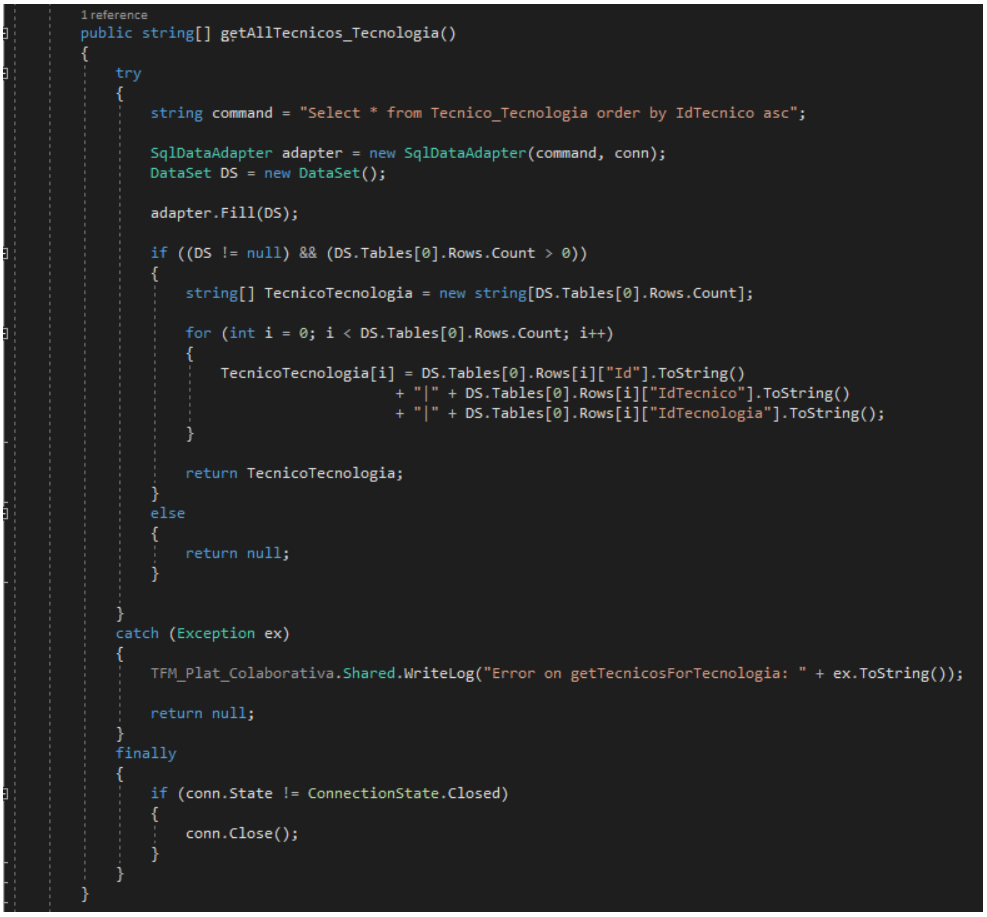
```
namespace TFM_Plat_Colaborativa
{
    11 references
    public class Tecnico_TecnologiaPersistence
    {
        private SqlConnection conn;

        5 references
        public Tecnico_TecnologiaPersistence()
        {
            string myConnectionString = null;

            try
            {
                //string myConnectionString = "Server = SAG-2J9W9H2; Database=TFM_Plat_Colaborativa; User Id = sa; Password = saIsaac;";
                if (ConfigurationManager.AppSettings["database"] != null)
                {
                    myConnectionString = ConfigurationManager.AppSettings["database"].ToString();
                }
                else
                {
                    myConnectionString = "Server = SAG-2J9W9H2; Database=TFM_Plat_Colaborativa; User Id = sa; Password = saIsaac;";
                }

                conn = new SqlConnection();
                conn.ConnectionString = myConnectionString;
                conn.Open();
            }
            catch (Exception ex)
            {
                TFM_Plat_Colaborativa.Shared.WriteLog("Error on Tecnico_TecnologiaPersistence: " + ex.ToString());
            }
        }
```

Fig 39.1: Tecnico_TecnologiaPersistence parte 1



```
1 reference
public long saveTecnico_Tecnologia(Tecnico_Tecnologia Tecnico_TecnologiaoToSave)
{

    SqlTransaction objTrans = null;

    try
    {
        //Hacer todo esto en una transacción
        string command = "Insert into Tecnico_Tecnologia (IdTecnico, IdTecnologia) values (" + Tecnico_TecnologiaoToSave.IdTecnico + ", "
            + Tecnico_TecnologiaoToSave.IdTecnologia + ")";

        objTrans = conn.BeginTransaction();

        SqlCommand cmd = new SqlCommand(command, conn, objTrans);
        int result = cmd.ExecuteNonQuery();

        if (result > 0)
        {
            command = "Select Max(id) from Tecnico_Tecnologia";
            cmd = new SqlCommand(command, conn, objTrans);

            var Id = cmd.ExecuteScalar();

            if ((Id != null) && (Convert.ToInt64(Id) > 0))
            { objTrans.Commit(); return Convert.ToInt64(Id); }
            else
            { objTrans.Rollback(); return -1; }
        }
        else
        {
            objTrans.Rollback();
            return -1;
        }


    }
    catch (Exception ex)
    {
        objTrans.Rollback();

        TFM_Plat_Colaborativa.Shared.WriteLog("Error on saveTecnico_Tecnologia: " + ex.ToString());

        return -1;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

Fig 39.2: Tecnico_TecnoogiaPersistence parte 2



```
1 reference
public string[] getAllTecnicos_Tecnologia()
{
    try
    {
        string command = "Select * from Tecnico_Tecnologia order by IdTecnico asc";

        SqlDataAdapter adapter = new SqlDataAdapter(command, conn);
        DataSet DS = new DataSet();

        adapter.Fill(DS);

        if ((DS != null) && (DS.Tables[0].Rows.Count > 0))
        {
            string[] TecnicoTecnologia = new string[DS.Tables[0].Rows.Count];

            for (int i = 0; i < DS.Tables[0].Rows.Count; i++)
            {
                TecnicoTecnologia[i] = DS.Tables[0].Rows[i]["Id"].ToString()
                                     + "|" + DS.Tables[0].Rows[i]["IdTecnico"].ToString()
                                     + "|" + DS.Tables[0].Rows[i]["IdTecnologia"].ToString();
            }

            return TecnicoTecnologia;
        }
        else
        {
            return null;
        }

    }
    catch (Exception ex)
    {
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on getTecnicosForTecnologia: " + ex.ToString());

        return null;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

Fig 39.3: Tecnico_TecnologiaPersistence parte 3

```
1 reference
public Tecnico[] getTecnicosForTecnologia(int idTecnologia)
{
    try
    {
        string command = "Select * from Tecnico_Tecnologia where idTecnologia = " + idTecnologia;

        SqlDataAdapter adapter = new SqlDataAdapter(command, conn);
        DataSet DS = new DataSet();

        adapter.Fill(DS);

        if ((DS != null) && (DS.Tables[0].Rows.Count > 0))
        {
            Tecnico[] Tecnicos = new Tecnico[DS.Tables[0].Rows.Count];
            TFM_Plat_Colaborativa.TecnicoPersistence T_P = new TecnicoPersistence();

            for (int i = 0; i < DS.Tables[0].Rows.Count; i++)
            {
                Tecnicos[i] = T_P.getTecnico(Convert.ToInt32(DS.Tables[0].Rows[i]["IdTecnico"]));
            }

            return Tecnicos;
        }
        else
        {
            return null;
        }

    }
    catch (Exception ex)
    {
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on getTecnicosForTecnologia: " + ex.ToString());

        return null;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

Fig 39.4: Tecnico_TecnologiaPersistence parte 4

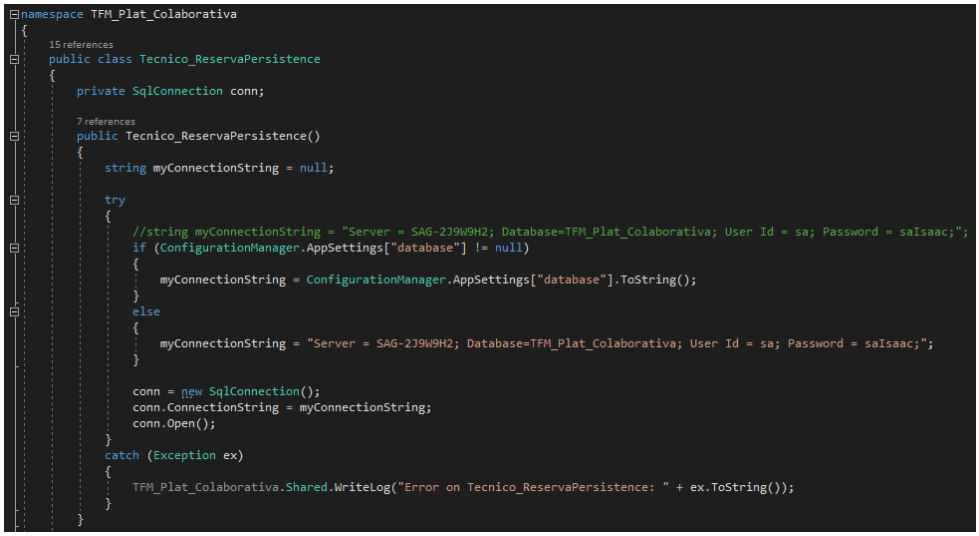
```
1 reference
public Tecnologia[] getTecnologiasForTecnico(int idTecnico)
{
    try
    {
        string command = "Select * from Tecnico_Tecnologia where idTecnico = " + idTecnico;

        SqlDataAdapter adapter = new SqlDataAdapter(command, conn);
        DataSet DS = new DataSet();

        adapter.Fill(DS);

        if ((DS != null) && (DS.Tables[0].Rows.Count > 0))
        {
            Tecnologia[] Tecnologias = new Tecnologia[DS.Tables[0].Rows.Count];
            TecnologiaPersistence T_P = new TecnologiaPersistence();

            for (int i = 0; i < DS.Tables[0].Rows.Count; i++)
            {
                Tecnologias[i] = T_P.getTecnologia(Convert.ToInt32(DS.Tables[0].Rows[i]["IdTecnologia"]));
            }

            return Tecnologias;
        }
        else
        {
            return null;
        }

    }
    catch (Exception ex)
    {
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on getTecnologiasForTecnico: " + ex.ToString());

        return null;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

Fig 39.5: Tecnico_TecnologiaPersistence parte 5

```
1 reference
public bool deleteTecnico_Tecnologia(int idTecnico_Tecnologia)
{
    SqlTransaction objTrans = null;

    try
    {
        string command = "Delete from Tecnico_Tecnologia where id = " + idTecnico_Tecnologia;

        objTrans = conn.BeginTransaction();

        SqlCommand cmd = new SqlCommand(command, conn, objTrans);
        int result = cmd.ExecuteNonQuery();

        if (result > 0)
        {
            objTrans.Commit();
            return true;
        }
        else
        {
            objTrans.Rollback();
            return false;
        }
    }
    catch (Exception ex)
    {
        objTrans.Rollback();
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on deleteTecnico_Tecnologia: " + ex.ToString());
        return false;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
}
```

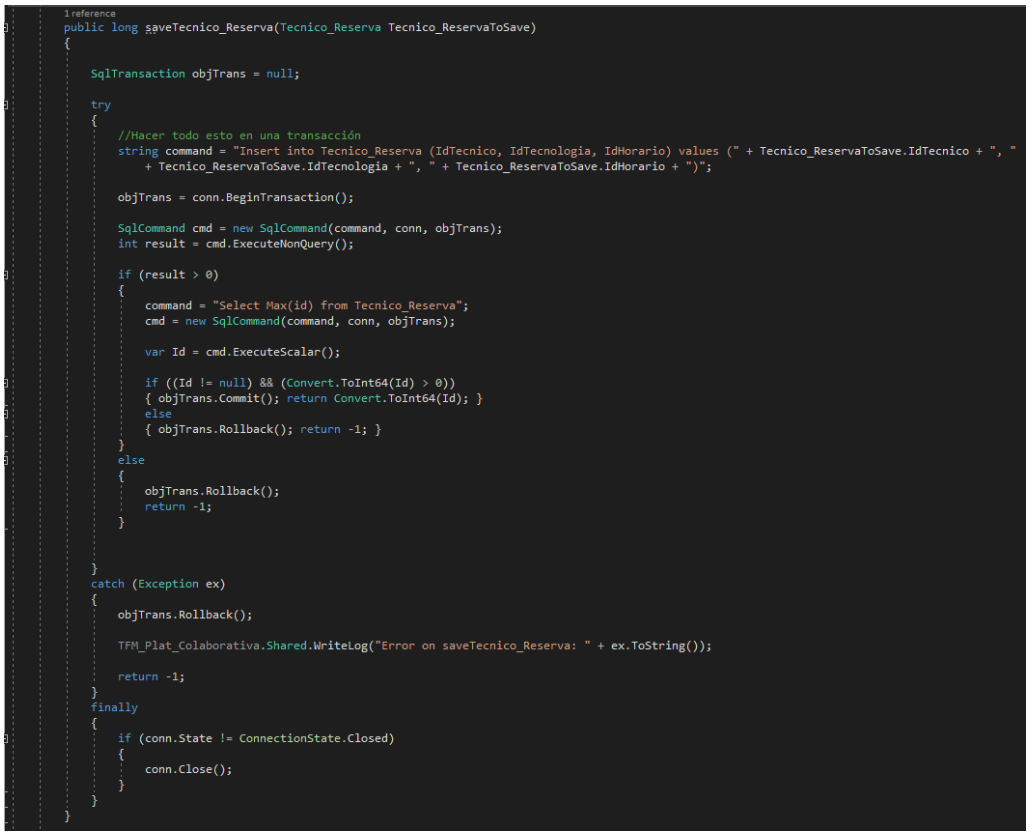
Fig 39.6: Tecnico_TecnologiaPersistence parte 6

Tecnico_ReservaPersistence: Es la clase encargada de realizar el acceso a datos del modelo *Tecnico_Reserva*. Como el resto de clases *Persistence* implementa un constructor que se encarga de instaurar y abrir la conexión con la base de datos. Entre sus métodos encontramos saveTecnico_Reserva (recibe un tipo de dato complejo *Tecnico_Reserva*, lo guarda en la tabla correspondiente, y devuelve el identificador del mismo en caso de éxito), getAllTecnico_Reserva_string (devuelve un *array* con los identificadores de todas las relaciones entre *Técnico*, *Tecnología*, y *Horario)*, getAllTecnico_Reserva_TR (devuelve un *array* del tipo *Tecnico_Reserva* con los identificadores de todas las relaciones entre *Técnico*, *Tecnología*, y *Horario)*, getReservasForTecnologia (recibe el identificador de *Tecnología* y devuelve un *array* con los identificadores de todas las relaciones entre *Técnico*, *Tecnología*, y *Horario* asociadas a esa tecnología), getReservasForTecnico (recibe el identificador de *Técnico* y devuelve un *array* con los identificadores de todas las relaciones entre *Técnico*, *Tecnología*, y *Horario* asociadas a ese técnico), getReservasForHorario (recibe el identificador de *Horario* y devuelve un *array* con los identificadores de todas las relaciones entre *Técnico*, *Tecnología*, y *Horario* asociadas a ese horario), por último deleteTecnico_Reserva (recibe el identificador del *Tecnico_Reserva* que queremos eliminar y devuelve un *boolean* con el resultado de la operación).



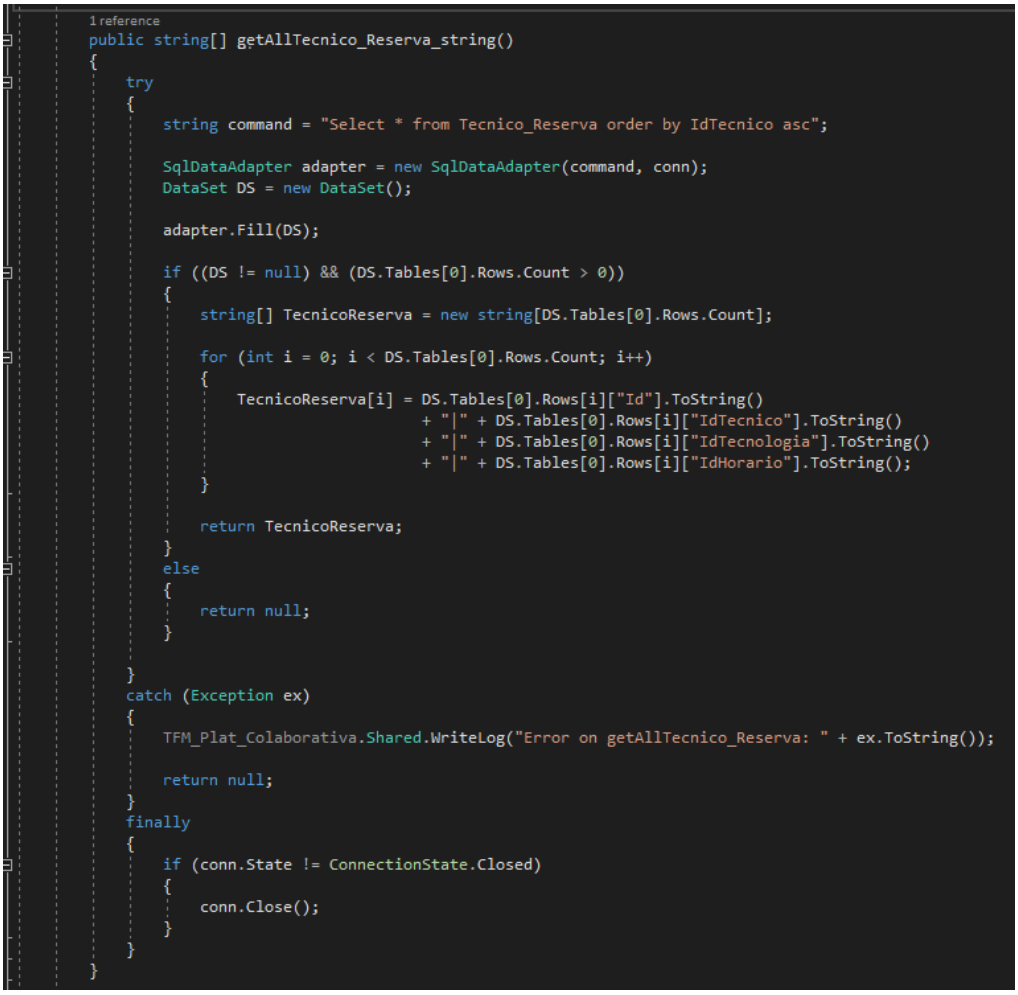
```
namespace TFM_Plat_Colaborativa
{
    15 references
    public class Tecnico_ReservaPersistence
    {
        private SqlConnection conn;

        7 references
        public Tecnico_ReservaPersistence()
        {
            string myConnectionString = null;

            try
            {
                //string myConnectionString = "Server = SAG-2J9W9H2; Database=TFM_Plat_Colaborativa; User Id = sa; Password = saIsaac;";
                if (ConfigurationManager.AppSettings["database"] != null)
                {
                    myConnectionString = ConfigurationManager.AppSettings["database"].ToString();
                }
                else
                {
                    myConnectionString = "Server = SAG-2J9W9H2; Database=TFM_Plat_Colaborativa; User Id = sa; Password = saIsaac;";
                }

                conn = new SqlConnection();
                conn.ConnectionString = myConnectionString;
                conn.Open();
            }
            catch (Exception ex)
            {
                TFM_Plat_Colaborativa.Shared.WriteLog("Error on Tecnico_ReservaPersistence: " + ex.ToString());
            }
        }
```

Fig 40.1: Tecnico_ReservaPersistence parte 1



```
        1 reference
        public long saveTecnico_Reserva(Tecnico_Reserva Tecnico_ReservaToSave)
        {

            SqlTransaction objTrans = null;

            try
            {
                //Hacer todo esto en una transacción
                string command = "Insert into Tecnico_Reserva (IdTecnico, IdTecnologia, IdHorario) values (" + Tecnico_ReservaToSave.IdTecnico + ", "
                    + Tecnico_ReservaToSave.IdTecnologia + ", " + Tecnico_ReservaToSave.IdHorario + ")";

                objTrans = conn.BeginTransaction();

                SqlCommand cmd = new SqlCommand(command, conn, objTrans);
                int result = cmd.ExecuteNonQuery();

                if (result > 0)
                {
                    command = "Select Max(id) from Tecnico_Reserva";
                    cmd = new SqlCommand(command, conn, objTrans);

                    var Id = cmd.ExecuteScalar();

                    if ((Id != null) && (Convert.ToInt64(Id) > 0))
                    { objTrans.Commit(); return Convert.ToInt64(Id); }
                    else
                    { objTrans.Rollback(); return -1; }
                }
                else
                {
                    objTrans.Rollback();
                    return -1;
                }


            }
            catch (Exception ex)
            {
                objTrans.Rollback();

                TFM_Plat_Colaborativa.Shared.WriteLog("Error on saveTecnico_Reserva: " + ex.ToString());

                return -1;
            }
            finally
            {
                if (conn.State != ConnectionState.Closed)
                {
                    conn.Close();
                }
            }
        }
```

Fig 40.2: Tecnico_ReservaPersistence parte 2

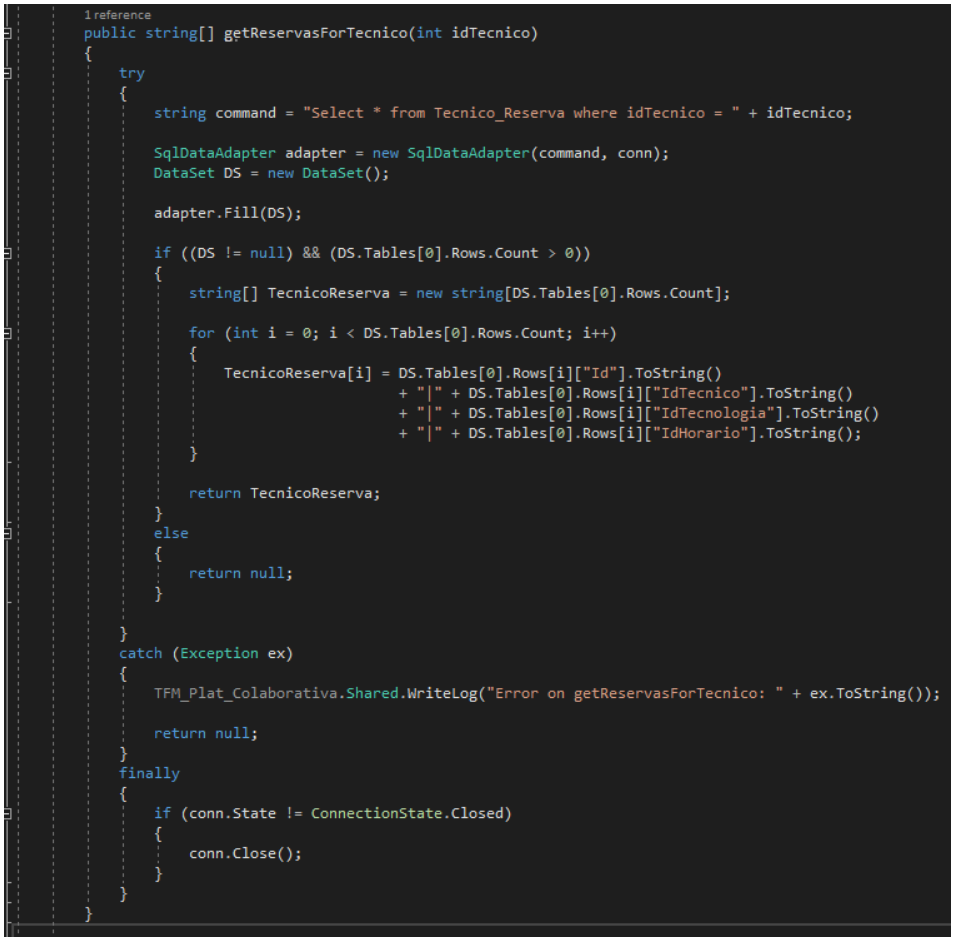


```
1 reference
public string[] getAllTecnico_Reserva_string()
{
    try
    {
        string command = "Select * from Tecnico_Reserva order by IdTecnico asc";

        SqlDataAdapter adapter = new SqlDataAdapter(command, conn);
        DataSet DS = new DataSet();

        adapter.Fill(DS);

        if ((DS != null) && (DS.Tables[0].Rows.Count > 0))
        {
            string[] TecnicoReserva = new string[DS.Tables[0].Rows.Count];

            for (int i = 0; i < DS.Tables[0].Rows.Count; i++)
            {
                TecnicoReserva[i] = DS.Tables[0].Rows[i]["Id"].ToString()
                                    + "|" + DS.Tables[0].Rows[i]["IdTecnico"].ToString()
                                    + "|" + DS.Tables[0].Rows[i]["IdTecnologia"].ToString()
                                    + "|" + DS.Tables[0].Rows[i]["IdHorario"].ToString();
            }

            return TecnicoReserva;
        }
        else
        {
            return null;
        }

    }
    catch (Exception ex)
    {
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on getAllTecnico_Reserva: " + ex.ToString());

        return null;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

Fig 40.3: Tecnico_ReservaPersistence parte 3

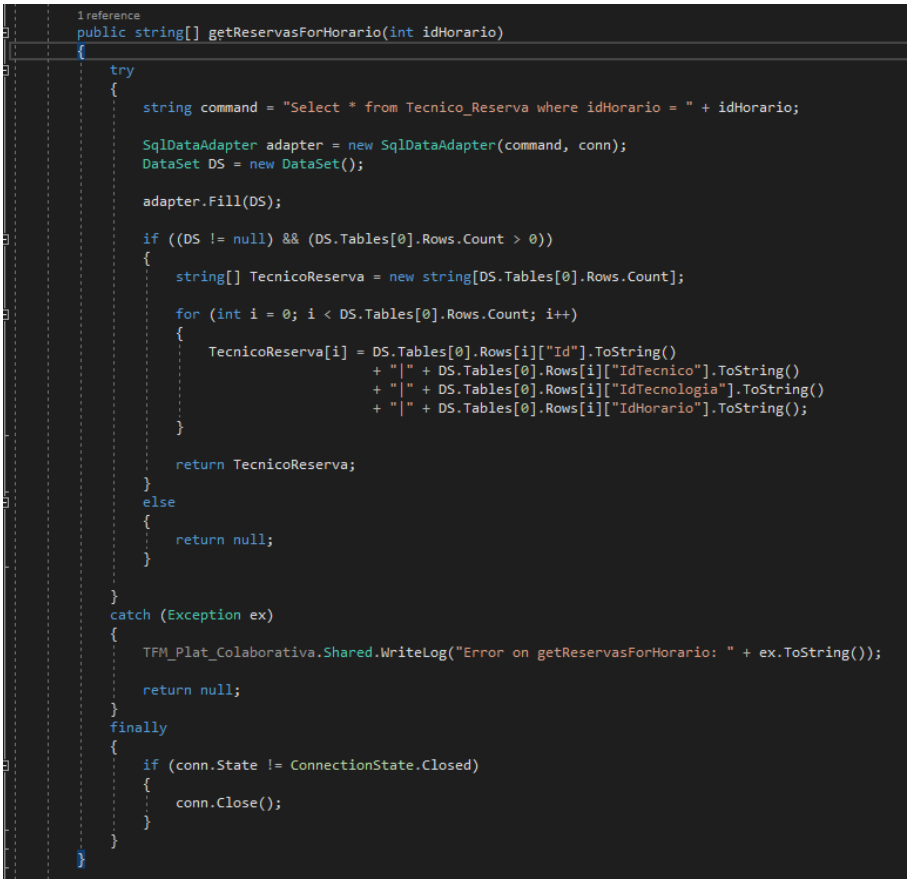
```
1 reference
public Tecnico_Reserva[] getAllTecnico_Reserva_TR()
{
    try
    {
        string command = "Select * from Tecnico_Reserva order by IdTecnico asc";

        SqlDataAdapter adapter = new SqlDataAdapter(command, conn);
        DataSet DS = new DataSet();

        adapter.Fill(DS);

        if ((DS != null) && (DS.Tables[0].Rows.Count > 0))
        {
            Tecnico_Reserva[] TecnicoReserva = new Tecnico_Reserva[DS.Tables[0].Rows.Count];

            for (int i = 0; i < DS.Tables[0].Rows.Count; i++)
            {
                TecnicoReserva[i] = new Tecnico_Reserva();
                TecnicoReserva[i].Id = Convert.ToInt32(DS.Tables[0].Rows[i]["Id"]);
                TecnicoReserva[i].IdTecnico = Convert.ToInt32(DS.Tables[0].Rows[i]["IdTecnico"]);
                TecnicoReserva[i].IdTecnologia = Convert.ToInt32(DS.Tables[0].Rows[i]["IdTecnologia"]);
                TecnicoReserva[i].IdHorario = Convert.ToInt32(DS.Tables[0].Rows[i]["IdHorario"]);
            }

            return TecnicoReserva;
        }
        else
        {
            return null;
        }

    }
    catch (Exception ex)
    {
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on getAllTecnico_Reserva: " + ex.ToString());

        return null;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

Fig 40.4: Tecnico_ReservaPersistence parte 4



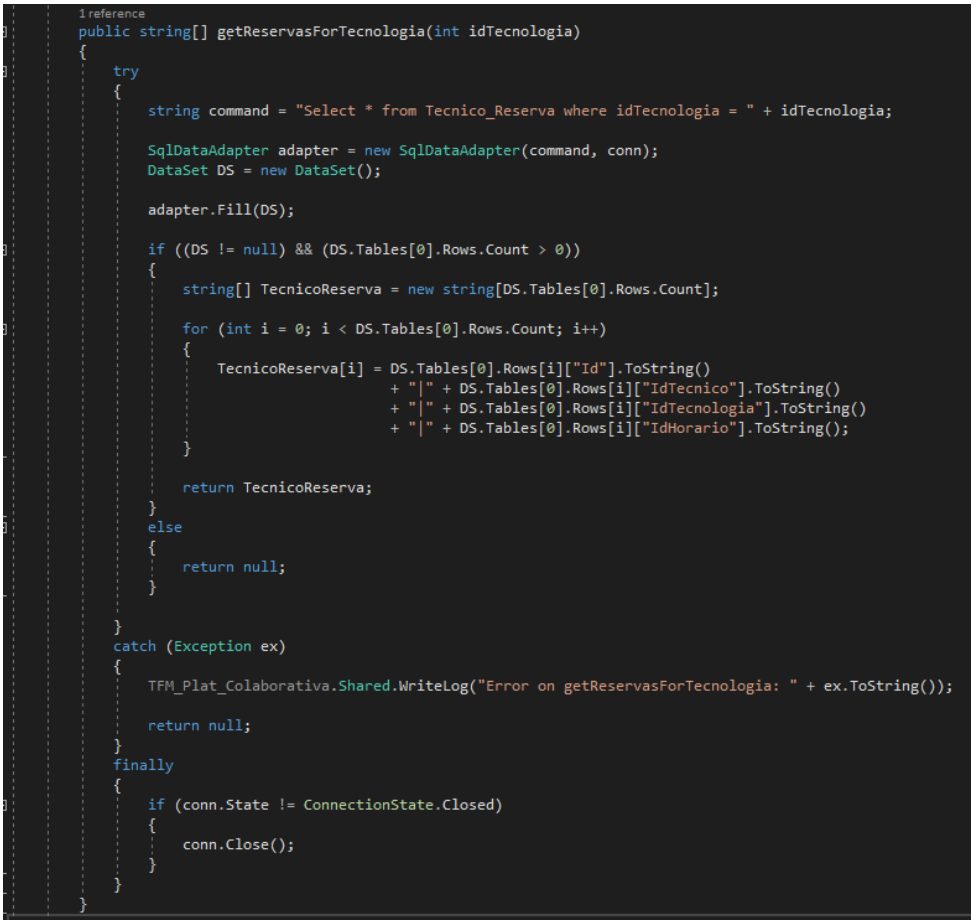
```
1 reference
public string[] getReservasForTecnico(int idTecnico)
{
    try
    {
        string command = "Select * from Tecnico_Reserva where idTecnico = " + idTecnico;

        SqlDataAdapter adapter = new SqlDataAdapter(command, conn);
        DataSet DS = new DataSet();

        adapter.Fill(DS);

        if ((DS != null) && (DS.Tables[0].Rows.Count > 0))
        {
            string[] TecnicoReserva = new string[DS.Tables[0].Rows.Count];

            for (int i = 0; i < DS.Tables[0].Rows.Count; i++)
            {
                TecnicoReserva[i] = DS.Tables[0].Rows[i]["Id"].ToString()
                                    + "|" + DS.Tables[0].Rows[i]["IdTecnico"].ToString()
                                    + "|" + DS.Tables[0].Rows[i]["IdTecnologia"].ToString()
                                    + "|" + DS.Tables[0].Rows[i]["IdHorario"].ToString();
                }

            return TecnicoReserva;
        }
        else
        {
            return null;
        }

    }
    catch (Exception ex)
    {
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on getReservasForTecnico: " + ex.ToString());

        return null;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

Fig 40.5: Tecnico_ReservaPersistence parte 5



```
1 reference
public string[] getReservasForHorario(int idHorario)
{
    try
    {
        string command = "Select * from Tecnico_Reserva where idHorario = " + idHorario;

        SqlDataAdapter adapter = new SqlDataAdapter(command, conn);
        DataSet DS = new DataSet();

        adapter.Fill(DS);

        if ((DS != null) && (DS.Tables[0].Rows.Count > 0))
        {
            string[] TecnicoReserva = new string[DS.Tables[0].Rows.Count];

            for (int i = 0; i < DS.Tables[0].Rows.Count; i++)
            {
                TecnicoReserva[i] = DS.Tables[0].Rows[i]["Id"].ToString()
                                    + "|" + DS.Tables[0].Rows[i]["IdTecnico"].ToString()
                                    + "|" + DS.Tables[0].Rows[i]["IdTecnologia"].ToString()
                                    + "|" + DS.Tables[0].Rows[i]["IdHorario"].ToString();
            }

            return TecnicoReserva;
        }
        else
        {
            return null;
        }

    }
    catch (Exception ex)
    {
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on getReservasForHorario: " + ex.ToString());

        return null;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

Fig 40.6: Tecnico_ReservaPersistence parte 6



```
1 reference
public string[] getReservasForTecnologia(int idTecnologia)
{
    try
    {
        string command = "Select * from Tecnico_Reserva where idTecnologia = " + idTecnologia;

        SqlDataAdapter adapter = new SqlDataAdapter(command, conn);
        DataSet DS = new DataSet();

        adapter.Fill(DS);

        if ((DS != null) && (DS.Tables[0].Rows.Count > 0))
        {
            string[] TecnicoReserva = new string[DS.Tables[0].Rows.Count];

            for (int i = 0; i < DS.Tables[0].Rows.Count; i++)
            {
                TecnicoReserva[i] = DS.Tables[0].Rows[i]["Id"].ToString()
                                  + "|" + DS.Tables[0].Rows[i]["IdTecnico"].ToString()
                                  + "|" + DS.Tables[0].Rows[i]["IdTecnologia"].ToString()
                                  + "|" + DS.Tables[0].Rows[i]["IdHorario"].ToString();
            }

            return TecnicoReserva;
        }
        else
        {
            return null;
        }

    }
    catch (Exception ex)
    {
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on getReservasForTecnologia: " + ex.ToString());

        return null;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
```

Fig 40.7: Tecnico_ReservaPersistence parte 7

```
1 reference
public bool deleteTecnico_Reserva(int idTecnico_Reserva)
{
    SqlTransaction objTrans = null;

    try
    {
        string command = "Delete from Tecnico_Reserva where id = " + idTecnico_Reserva;

        objTrans = conn.BeginTransaction();

        SqlCommand cmd = new SqlCommand(command, conn, objTrans);
        int result = cmd.ExecuteNonQuery();

        if (result > 0)
        {
            objTrans.Commit();
            return true;
        }
        else
        {
            objTrans.Rollback();
            return false;
        }
    }
    catch (Exception ex)
    {
        objTrans.Rollback();
        TFM_Plat_Colaborativa.Shared.WriteLog("Error on deleteTecnico_Reserva: " + ex.ToString());
        return false;
    }
    finally
    {
        if (conn.State != ConnectionState.Closed)
        {
            conn.Close();
        }
    }
}
}
```

Fig 40.8: Tecnico_ReservaPersistence parte 8

5.6.2.4 Shared y Web.config

La primera clase es común a toda la API y en ella se definirán los métodos que son comunes, compartidos y utilizados por el resto de clases, un claro ejemplo y que podemos ver a continuación es el método que escribe es log de la API en el path indicado en el archivo de configuración.

```
namespace TFM_Plat_Colaborativa
{
    32 references
    public class Shared
    {
        32 references
        public static void WriteLog(string Description)
        {
            try
            {
                //Setting the path for the Log
                string Path = "";
                if ((ConfigurationManager.AppSettings["LogPath"] != null) && (ConfigurationManager.AppSettings["LogPath"].ToString() != ""))
                {
                    if (ConfigurationManager.AppSettings["LogPath"].EndsWith("\\"))
                    {
                        Path = ConfigurationManager.AppSettings["LogPath"];
                    }
                    else
                    {
                        Path = ConfigurationManager.AppSettings["LogPath"] + "\\";
                    }
                }
                else
                {
                    Path = System.AppDomain.CurrentDomain.BaseDirectory.ToString() + "Log\\";
                }

                //Create the Path
                if (!(System.IO.Directory.Exists(Path)))
                {
                    System.IO.Directory.CreateDirectory(Path);
                }

                string FileName = DateTime.Now.ToShortDateString().Replace("/", "-") + "TFM_Plat_Colaborativa.txt";
                System.IO.StreamWriter WriteStreamLog = new System.IO.StreamWriter(Path + FileName, true);

                WriteStreamLog.WriteLine(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss\t") + Description);
                WriteStreamLog.Close();

            }
            catch (Exception)
            { }
        }
    }
}
```

Fig 41: Shared.cs

Igualmente en el archivo Web.config encontramos los aspectos básicos de configuración de nuestra API (como por ejemplo el número de ensamblado, versión de framework, ..) y además nos podemos permitir añadir tantos como deseemos para luego utilizarlos en nuestra API y poder cambiar ciertos valores de configuración sin tener que recompilar todo el proyecto y/o sin tener que poner una nueva versión en producción. Como ejemplo, si se quiere cambiar la base de datos (e incluso cambiar el motor de BBDD) con cambiar la cadena de conexión en el archivo de configuración bastaría, no haría falta una parada planificada de nuestra API ni recompilar el código.

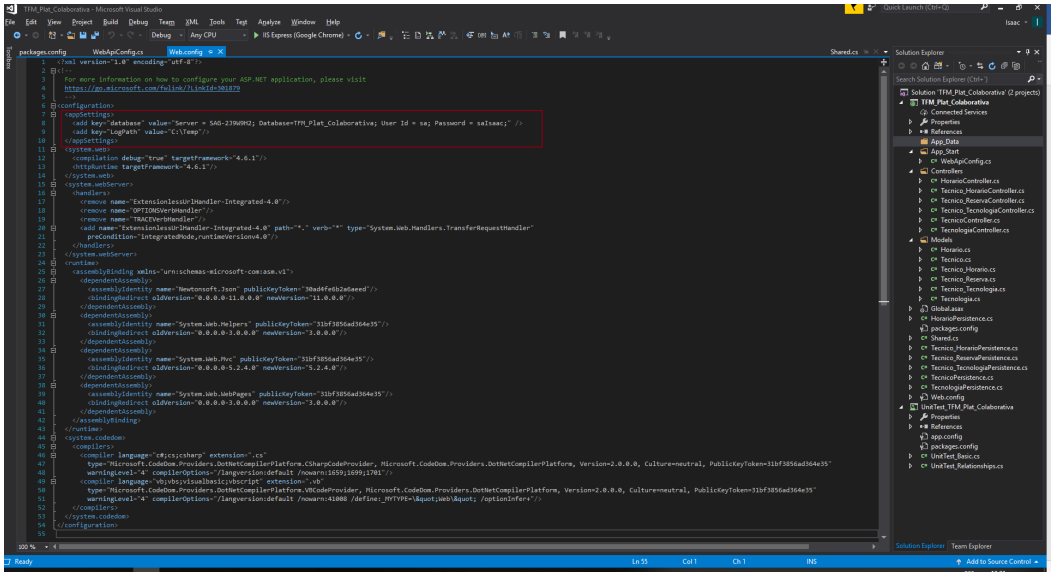


Fig 41.1: Web.config

```
<configuration>
  <appSettings>
    <add key="database" value="Server = SAG-2J9W9H2; Database=TFM_Plat_Colaborativa; User Id = sa; Password = saIsaac;" />
    <add key="LogPath" value="C:\Temp"/>
  </appSettings>
  <system.web>
```

Fig 41.2: Web.config – appSettings

5.6.3 Unit TEST

Con la ayuda de Unit TEST (pruebas unitarias) es posible comprobar los componentes individuales de los programas informáticos. Estos test permiten examinar el correcto funcionamiento de cada uno de los elementos antes de que ocupen su lugar en el concepto general de un programa. Además, ayudan a comprobar de forma relativamente rápida y fácil si el componente funciona según lo previsto por el desarrollador. Los Unit TEST son una de las formas más eficaces para descubrir la mayor cantidad posible de errores del código en las fases tempranas de desarrollo del software.

El unit testing juega un papel fundamental en el control de calidad de cualquier programa. En especial en el desarrollo de software mediante tecnologías *agile* se apuesta cada vez más por este método. Por lo general, las pruebas unitarias tienen como objetivo la comprobación frecuente de diversos componentes, es por esto que se realizan de forma automática. Así, con solo presionar un botón, los respectivos programas realizan varias pruebas unitarias al azar. Es común que el programa de prueba utilizado esté escrito en el mismo lenguaje del objeto de prueba.

El término Unit TEST se refiere al método de comprobación de las "unidades" (en inglés unit) más pequeñas del software. Los componentes más pequeños que pueden probarse y cuyos resultados con más significativos son los módulos. Es recomendable comprobarlos en las primeras fases de desarrollo, pues en la fase de prueba, el módulo aún se puede corregir de forma relativamente rápida y poco costosa. En fases posteriores, estos procesos están asociados a mayores gastos. Las pruebas unitarias se ocupan principalmente de las funcionalidades técnicas. Por lo general, el desarrollador es quien ejecuta las pruebas y se encarga de corregir errores y asegurar la correcta funcionalidad de los componentes. Al método o función que se pretende probar, se le conoce como sistema bajo prueba (System Under Test – SUT).

Cuando ejecutamos los test, estos devolverán un resultado de éxito o fracaso, identificándose rápidamente por verde o rojo en Visual Studio, este resultado dependerá de que los Assert se cumplan.

Por tanto lo primero que debemos hacer es añadir un nuevo proyecto de tipo *Unit Test Project* a nuestra solución. No obstante, el framework que nos permite implementar los Unit TEST no siempre viene instalado de forma nativa, con lo que tendremos que ir a *Manage NuGet Packages* y descargarlo en caso de que no lo tengamos:

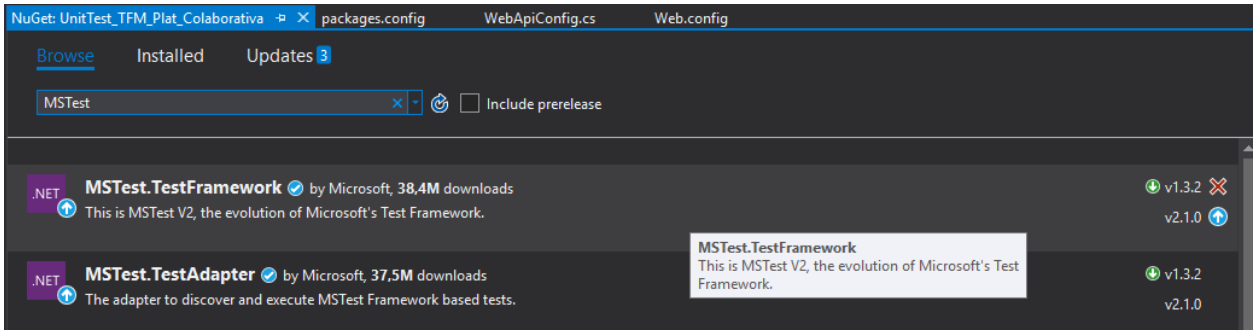


Fig 42: MSTest.TestFramework

En nuestro proyecto de Unit TEST tenemos dos clases de prueba que van a asegurar el buen funcionamiento de las clases básicas *UnitTest_Basic.cs* y de las clases relacionales *UnitTest_Relationships.cs*

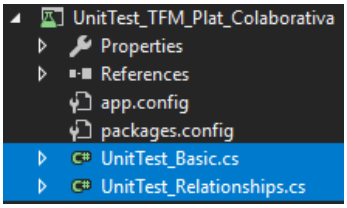


Fig 43: Clases para Unit TEST

En *UnitTest_Basic* probamos la funcionalidad básica de la clase *Horario* (damos de alta un horario, recuperamos la información del horario dado de alta y lo borramos), *Tecnología* (damos de alta una tecnología, recuperamos la información de la tecnología dada de alta y la borramos), y *Técnico* (damos de alta un técnico, recuperamos la información del técnico dado de alta y lo borramos).

```
[TestMethod]
0 references
public void basicHorario()
{
    int id = -1;

    Horario H = new Horario();
    H.Franja = "UnitTest_Basic_Horario";
    H.NumHoras = 9999;

    var HC = new HorarioController();
    var response = HC.Post(H);

    if (response.ToLower().Contains("ok: "))
    {
        try
        {
            id = Convert.ToInt32(response.Split(' ')[1]);
            Assert.AreEqual(1, 1);
        }
        catch
        {
            Assert.AreEqual(1, -1);
        }
    }
    Assert.AreNotEqual("Error", response);

    Console.WriteLine("El Id Horario utilizado es:" + id.ToString());

    H = HC.Get(id);

    if (H != null)
    {
        try
        {
            Assert.AreEqual(H.Id, id);
        }
        catch
        {
            Assert.AreEqual(1, -1);
        }
    }
    else
    {
        Assert.AreEqual(1, -1);
    }

    bool delete = HC.Delete(id);

    Assert.AreEqual(true, delete);
}
```

Fig 44.1: UnitTest_Basic parte 1

```
[TestMethod]
0 references
public void basicTecnologia()
{
    int id = -1;

    Tecnologia T = new Tecnologia();
    T.NombreTecnologia = "UnitTest_Basic_Tecnologia";
    T.PrecioHora = 999;

    var TC = new TecnologiaController();
    var response = TC.Post(T);

    if (response.ToLower().Contains("ok: "))
    {
        try
        {
            id = Convert.ToInt32(response.Split(' ')[1]);
            Assert.AreEqual(1, 1);
        }
        catch
        {
            Assert.AreEqual(1, -1);
        }
    }
    Assert.AreNotEqual("Error", response);

    Console.WriteLine("El Id Tecnologia utilizado es:" + id.ToString());

    T = TC.Get(id);

    if (T != null)
    {
        try
        {
            Assert.AreEqual(T.Id, id);
        }
        catch
        {
            Assert.AreEqual(1, -1);
        }
    }
    else
    {
        Assert.AreEqual(1, -1);
    }

    bool delete = TC.Delete(id);

    Assert.AreEqual(true, delete);
}
```

Fig 44.2: UnitTest_Basic parte 2

```
[TestMethod]
0 references
public void basicTecnico()
{
    int id = -1;

    Tecnico T = new Tecnico();
    T.Nombre = "UnitTest";
    T.Apellidos = "Basic_Tecnico";
    T.Direccion = "Address";
    T.eMail = "eMail";
    T.NumCuenta = "Bank";
    T.Telefono = "999";

    var TC = new TecnicoController();
    var response = TC.Post(T);

    if (response.ToLower().Contains("ok: "))
    {
        try
        {
            id = Convert.ToInt32(response.Split(' ')[1]);
            Assert.AreEqual(1, 1);
        }
        catch
        {
            Assert.AreEqual(1, -1);
        }
    }
    Assert.AreNotEqual("Error", response);

    Console.WriteLine("El Id Tecnologia utilizado es:" + id.ToString());

    T = TC.Get(id);

    if (T != null)
    {
        try
        {
            Assert.AreEqual(T.Id, id);
        }
        catch
        {
            Assert.AreEqual(1, -1);
        }
    }
    else
    {
        Assert.AreEqual(1, -1);
    }

    bool delete = TC.Delete(id);

    Assert.AreEqual(true, delete);
```

Fig 44.3: UnitTest_Basic parte 3

En *UnitTest_Relationships* probamos la funcionalidad básica de la clase *Técnico_Tecnología* (damos de alta una tecnología y un técnico, así como la relación entre ambos. Recuperamos la información de la clase *Tecnico_Tecnología* y comprobamos la información que hemos dado de alta. Si todo es correcto borramos la relación que hemos creado, y luego al técnico y a la tecnología que hemos añadido para la prueba), *Técnico_Horario* (damos de alta un horario y un técnico, así como la relación entre ambos. Recuperamos la información de la clase *Tecnico_Horario* y comprobamos la información que hemos dado de alta. Si todo es

correcto borramos la relación que hemos creado, y luego al técnico y al horario que hemos añadido para la prueba), y *Técnico_Reserva* (damos de alta un horario, un técnico y una tecnología, así como la relación entre ellos. Recuperamos la información de la clase *Tecnico_Reserva* y comprobamos la información que hemos dado de alta. Si todo es correcto borramos la relación que hemos creado, y luego al técnico, el horario, y la tecnología que hemos añadido para la prueba).

```
[TestMethod]
0 references
public void basic_TecnicoTecnologia()
{
    int id = -1;

    //Primero insertamos un técnico y una tecnología para que no de error de inexistencia
    //ni de claves duplicadas
    int id_Tecnologia = -1;

    Tecnologia Tec = new Tecnologia
    {
        NombreTecnologia = "UnitTest_Basic_Tecnologia",
        PrecioHora = 999
    };

    var TC = new TecnologiaController();
    var response_TC = TC.Post(Tec);

    if (response_TC.ToLower().Contains("ok: "))
    {
        try
        {
            id_Tecnologia = Convert.ToInt32(response_TC.Split(' ')[1]);
            Assert.AreEqual(1, 1);
        }
        catch
        {
            Assert.AreEqual(1, -1);
        }
    }
    Assert.AreNotEqual("Error", response_TC);

    int id_Tecnico = -1;

    Tecnico T = new Tecnico
    {
        Nombre = "UnitTest",
        Apellidos = "Basic_Tecnico",
        Direccion = "Address",
        eMail = "eMail",
        NumCuenta = "Bank",
        Telefono = "999"
    };
```

```
var T_C = new TecnicoController();
var response_T_C = T_C.Post(T);

if (response_T_C.ToLower().Contains("ok: "))
{
    try
    {
        id_Tecnico = Convert.ToInt32(response_T_C.Split(' ')[1]);
        Assert.AreEqual(1, 1);
    }
    catch
    {
        Assert.AreEqual(1, -1);
    }
}
Assert.AreNotEqual("Error", response_T_C);
//

Tecnico_Tecnologia T_T = new Tecnico_Tecnologia();
T_T.IdTecnico = id_Tecnico;
T_T.IdTecnologia = id_Tecnologia;

var TT_C = new Tecnico_TecnologiaController();
var response = TT_C.Post(T_T);

if (response.ToLower().Contains("ok: "))
{
    try
    {
        id = Convert.ToInt32(response.Split(' ')[1]);
        Assert.AreEqual(1, 1);
    }
    catch
    {
        Assert.AreEqual(1, -1);
    }
}
Assert.AreNotEqual("Error", response);

bool encontrado = false;

foreach(string T_T_Aux in TT_C.Get())
{
    if (T_T_Aux == id + "|" + id_Tecnico + "|" + id_Tecnologia)
    {
        encontrado = true;
        Assert.AreEqual(1, 1);
        break;
    }
}
```

```
    if (encontrado)
    {
        Assert.AreEqual(1, 1);
    }
    else
    {
        Assert.AreEqual(1, -1);
    }

    bool delete = TT_C.Delete(id);

    //Ahora borramos tanto el técnico, como la tecnología que hemos añadido para la prueba
    bool delete_Tec = TC.Delete(id_Tecnologia);
    Assert.AreEqual(true, delete_Tec);

    bool delete_T = T_C.Delete(id_Tecnico);
    Assert.AreEqual(true, delete_T);
    //

    Assert.AreEqual(true, delete);

}
```

Fig 45.1: UnitTest_Relationships parte 1

```
[TestMethod]
0 references
public void basic_TecnicoHorario()
{
    int id = -1;

    //Primero insertamos un técnico y un horario para que no de error de inexistencia
    //ni de claves duplicadas
    int id_Horario = -1;

    Horario H = new Horario
    {
        Franja = "UnitTest_Basic_Horario",
        NumHoras = 999
    };

    var HC = new HorarioController();
    var response_HC = HC.Post(H);

    if (response_HC.ToLower().Contains("ok: "))
    {
        try
        {
            id_Horario = Convert.ToInt32(response_HC.Split(' ')[1]);
            Assert.AreEqual(1, 1);
        }
        catch
        {
            Assert.AreEqual(1, -1);
        }
    }
    Assert.AreNotEqual("Error", response_HC);

    int id_Tecnico = -1;

    Tecnico T = new Tecnico
    {
        Nombre = "UnitTest",
        Apellidos = "Basic_Tecnico",
        Direccion = "Address",
        eMail = "eMail",
        NumCuenta = "Bank",
        Telefono = "999"
    };
```

```
var T_C = new TecnicoController();
var response_T_C = T_C.Post(T);

if (response_T_C.ToLower().Contains("ok: "))
{
    try
    {
        id_Tecnico = Convert.ToInt32(response_T_C.Split(' ')[1]);
        Assert.AreEqual(1, 1);
    }
    catch
    {
        Assert.AreEqual(1, -1);
    }
}
Assert.AreNotEqual("Error", response_T_C);
//

Tecnico_Horario T_H = new Tecnico_Horario();
T_H.IdTecnico = id_Tecnico;
T_H.IdHorario = id_Horario;

var TH_C = new Tecnico_HorarioController();
var response = TH_C.Post(T_H);

if (response.ToLower().Contains("ok: "))
{
    try
    {
        id = Convert.ToInt32(response.Split(' ')[1]);
        Assert.AreEqual(1, 1);
    }
    catch
    {
        Assert.AreEqual(1, -1);
    }
}
Assert.AreNotEqual("Error", response);

bool encontrado = false;

foreach (string T_H_Aux in TH_C.Get())
{
    if (T_H_Aux == id + "|" + id_Tecnico + "|" + id_Horario)
    {
        encontrado = true;
        Assert.AreEqual(1, 1);
        break;
    }
}
```

```
    if (encontrado)
    {
        Assert.AreEqual(1, 1);
    }
    else
    {
        Assert.AreEqual(1, -1);
    }

    bool delete = TH_C.Delete(id);

    //Ahora borramos tanto el técnico como el horario que hemos añadido para la prueba
    bool delete_Hor = HC.Delete(id_Horario);
    Assert.AreEqual(true, delete_Hor);

    bool delete_T = T_C.Delete(id_Tecnico);
    Assert.AreEqual(true, delete_T);
    //

    Assert.AreEqual(true, delete);

}
```

Fig 45.2: UnitTest_Relationship parte 2

```
[TestMethod]
0 references
public void basic_TecnicoReserva()
{
    int id = -1;

    //Primero insertamos un técnico, una tecnología y un horario para que no de error de inexistencia
    //ni de claves duplicadas
    int id_Horario = -1;

    Horario H = new Horario
    {
        Franja = "UnitTest_Basic_TecnicoReserva",
        NumHoras = 999
    };

    var HC = new HorarioController();
    var response_HC = HC.Post(H);

    if (response_HC.ToLower().Contains("ok: "))
    {
        try
        {
            id_Horario = Convert.ToInt32(response_HC.Split(' ')[1]);
            Assert.AreEqual(1, 1);
        }
        catch
        {
            Assert.AreEqual(1, -1);
        }
    }
    Assert.AreNotEqual("Error", response_HC);

    int id_Tecnico = -1;

    Tecnico T = new Tecnico
    {
        Nombre = "UnitTest",
        Apellidos = "Basic_Tecnico",
        Direccion = "Address",
        eMail = "eMail",
        NumCuenta = "Bank",
        Telefono = "999"
    };
```

```
var T_C = new TecnicoController();
var response_T_C = T_C.Post(T);

if (response_T_C.ToLower().Contains("ok: "))
{
    try
    {
        id_Tecnico = Convert.ToInt32(response_T_C.Split(' ')[1]);
        Assert.AreEqual(1, 1);
    }
    catch
    {
        Assert.AreEqual(1, -1);
    }
}
Assert.AreNotEqual("Error", response_T_C);

int id_Tecnologia = -1;

Tecnologia Tec = new Tecnologia
{
    NombreTecnologia = "UnitTest_Basic_Tecnologia",
    PrecioHora = 999
};

var TC = new TecnologiaController();
var response_TC = TC.Post(Tec);

if (response_TC.ToLower().Contains("ok: "))
{
    try
    {
        id_Tecnologia = Convert.ToInt32(response_TC.Split(' ')[1]);
        Assert.AreEqual(1, 1);
    }
    catch
    {
        Assert.AreEqual(1, -1);
    }
}
Assert.AreNotEqual("Error", response_TC);
//

Tecnico_Reserva T_R = new Tecnico_Reserva();
T_R.IdTecnico = id_Tecnico;
T_R.IdTecnologia = id_Tecnologia;
T_R.IdHorario = id_Horario;
```

```
    var TR_C = new Tecnico_ReservaController();
    var response = TR_C.Post(T_R);

    if (response.ToLower().Contains("ok: "))
    {
        try
        {
            id = Convert.ToInt32(response.Split(' ')[1]);
            Assert.AreEqual(1, 1);
        }
        catch
        {
            Assert.AreEqual(1, -1);
        }
    }
    Assert.AreNotEqual("Error", response);

    bool encontrado = false;

    foreach (string T_H_Aux in TR_C.Get())
    {
        if (T_H_Aux == id + "|" + id_Tecnico + "|" + id_Tecnologia + "|" + id_Horario)
        {
            encontrado = true;
            Assert.AreEqual(1, 1);
            break;
        }
    }

    if (encontrado)
    {
        Assert.AreEqual(1, 1);
    }
    else
    {
        Assert.AreEqual(1, -1);
    }

    bool delete = TR_C.Delete(id);
```

```
    //Ahora borramos tanto el técnico como el horario que hemos añadido para la prueba
    bool delete_Hor = HC.Delete(id_Horario);
    Assert.AreEqual(true, delete_Hor);

    bool delete_T = T_C.Delete(id_Tecnico);
    Assert.AreEqual(true, delete_T);

    bool delete_Tec = TC.Delete(id_Tecnologia);
    Assert.AreEqual(true, delete_Tec);
    //

    Assert.AreEqual(true, delete);

}
```

Fig 45.3: UnitTest_Relationships parte 3

Una vez ejecutados los Test, si estos han sido satisfactorios, nos encontraremos con una imagen similar a esta:

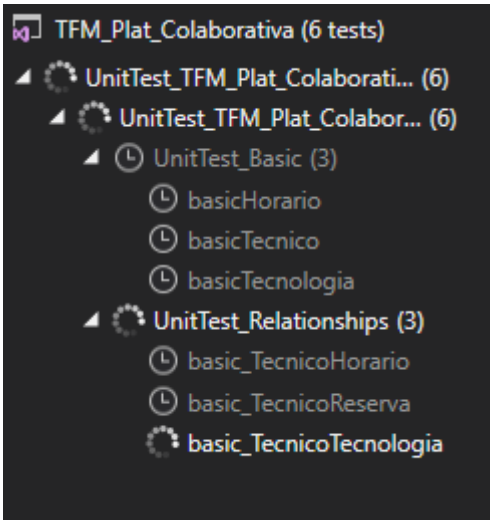


Fig 46: Unit TEST ejecutando

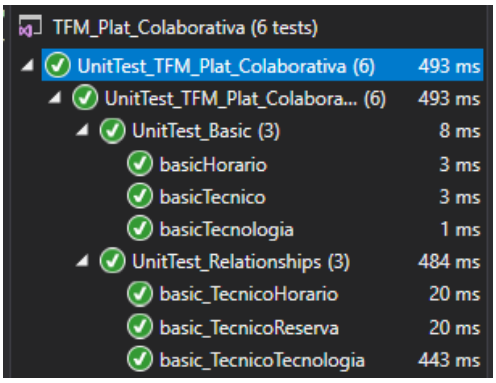


Fig 47: Unit TEST correctos

5.7 Pruebas con aplicación cliente (Postman)

Una vez que ya tenemos diseñado nuestra API REST, implementada, y en modo depuración para recibir peticiones, lo siguiente que tenemos que hacer es mediante el uso de un cliente, hacer esas peticiones. En este caso, por su versatilidad y sencillez se ha utilizado Postman, que se está convirtiendo (si no lo es ya) en la herramienta más habitual entre los desarrolladores que necesitan de un cliente para probar el consumo de sus servicios REST.

Peticiones POST, GET, DELETE para la clase *Tecnología*

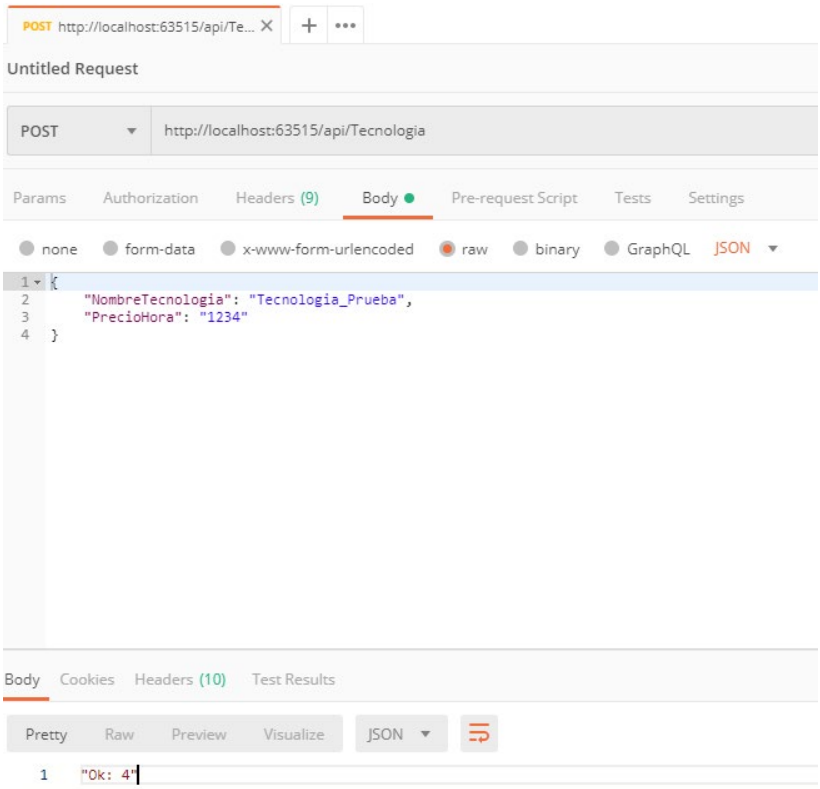


Fig 48: POST call para *Tecnología*

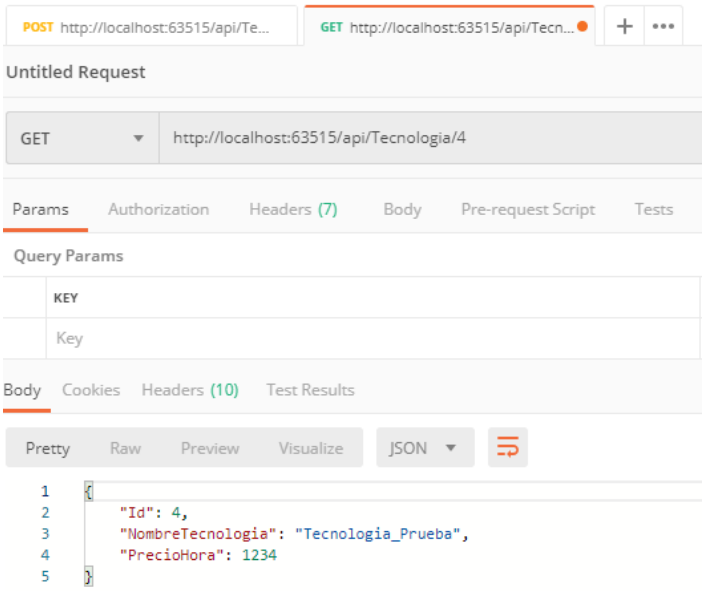


Fig 49: GET call para *Tecnología*

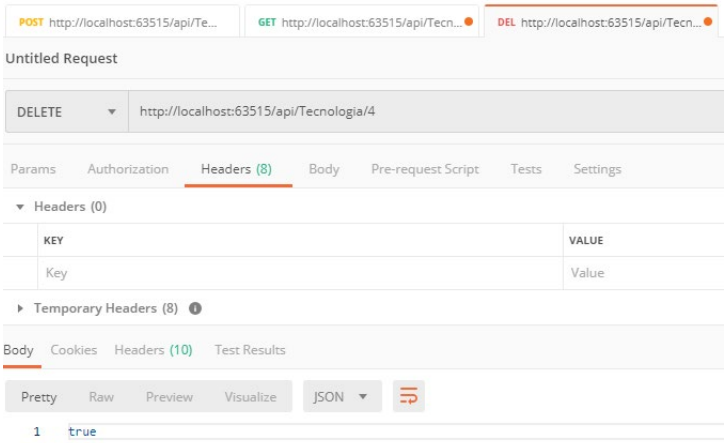


Fig 50: DELETE call para *Tecnología*

Peticiones POST, GET, DELETE para la clase *Horario*

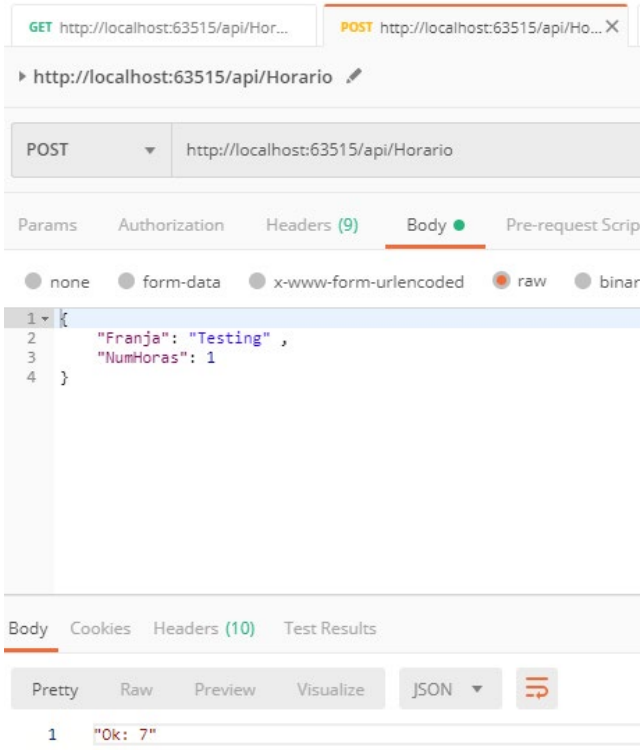


Fig 51: POST call para *Horario*

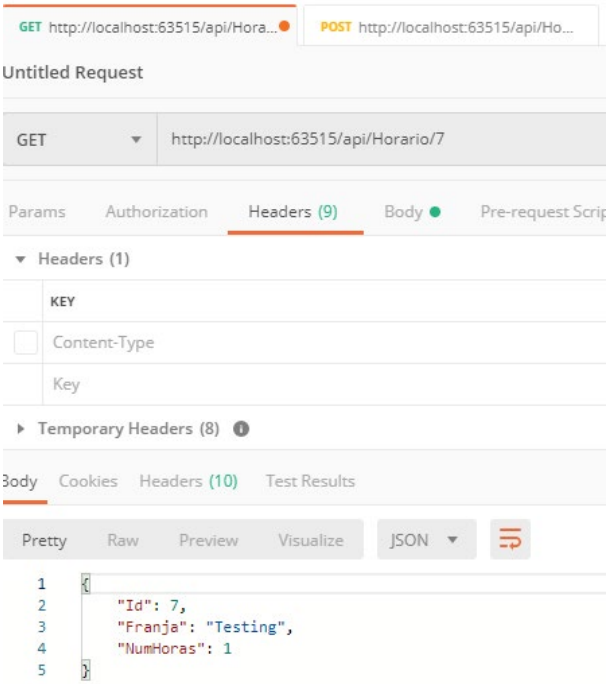


Fig 52: GET call for *Horario*

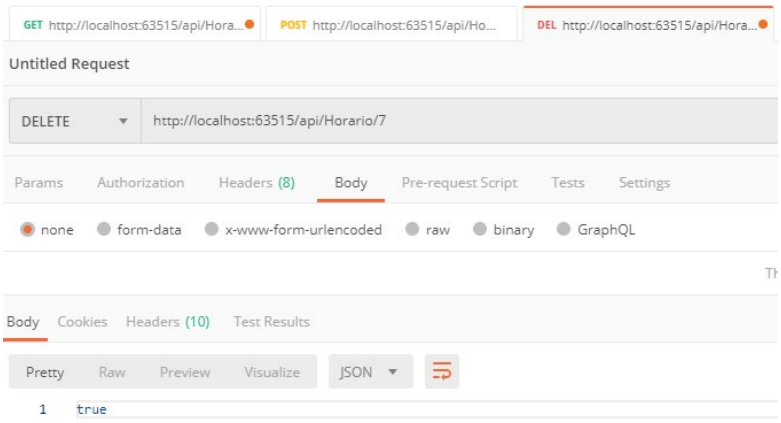


Fig 53: DELETE call for *Horario*

Peticiones POST, GET, DELETE para la clase *Técnico*

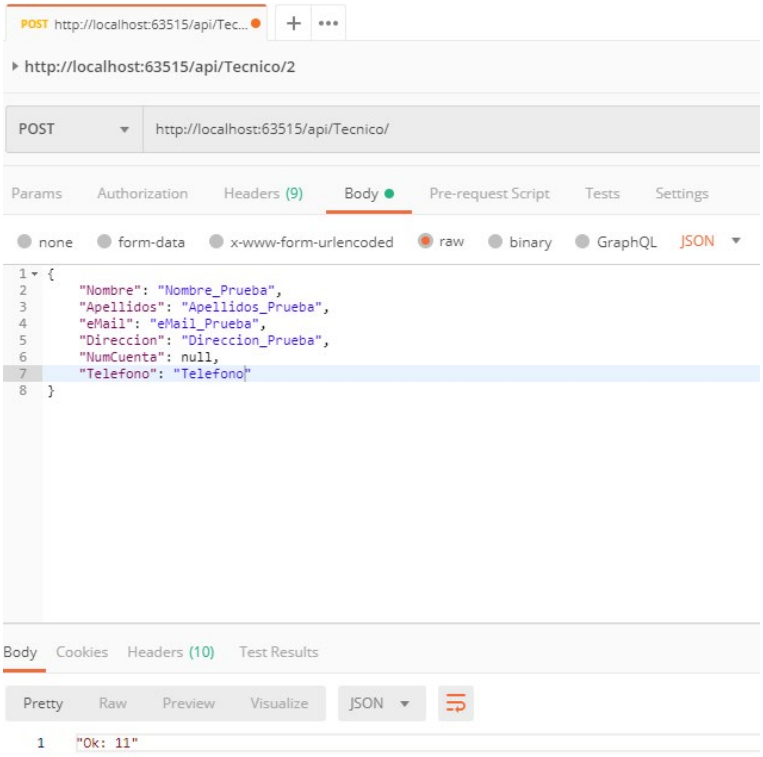


Fig 54: POST call para *Técnico*

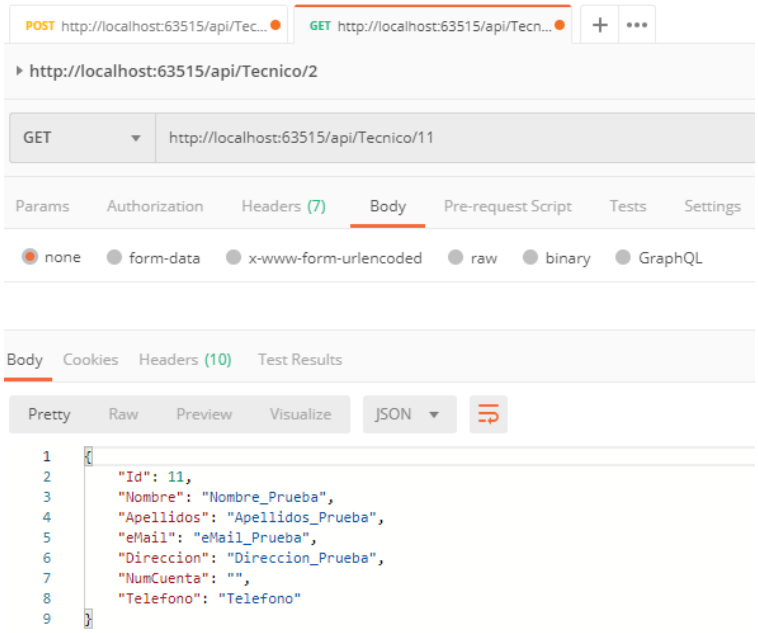


Fig 55: GET call para *Técnico*

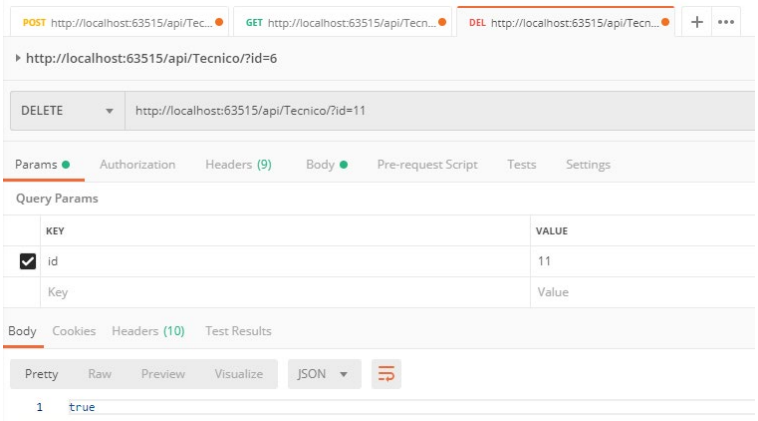


Fig 56: DELETE call para *Técnico*

Peticiones POST, GET, (GET)GetTecnicosForHorario, (GET)GetHorariosForTecnico, DELETE para la clase *Técnico_Horario*

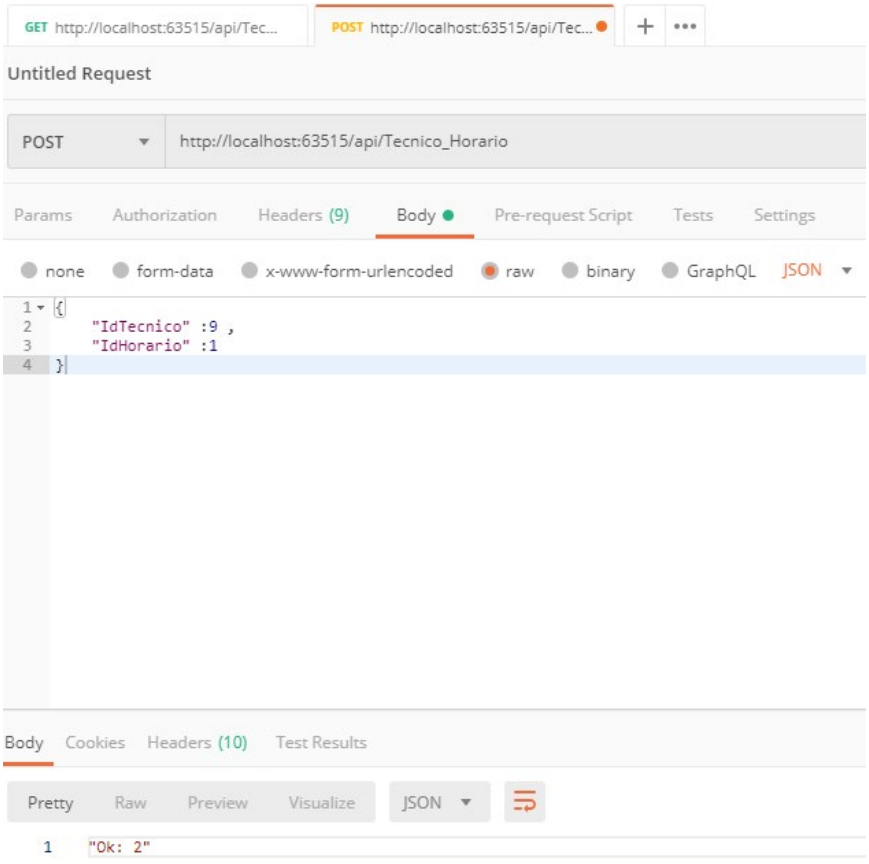


Fig 57: POST call para *Técnico_Horario*

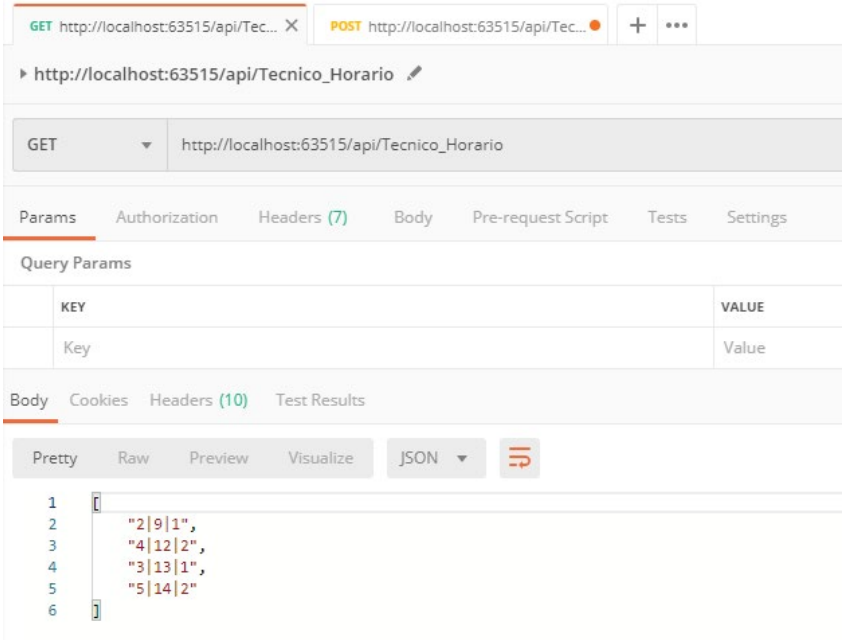


Fig 58: GET call para *Tecnico_Horario*

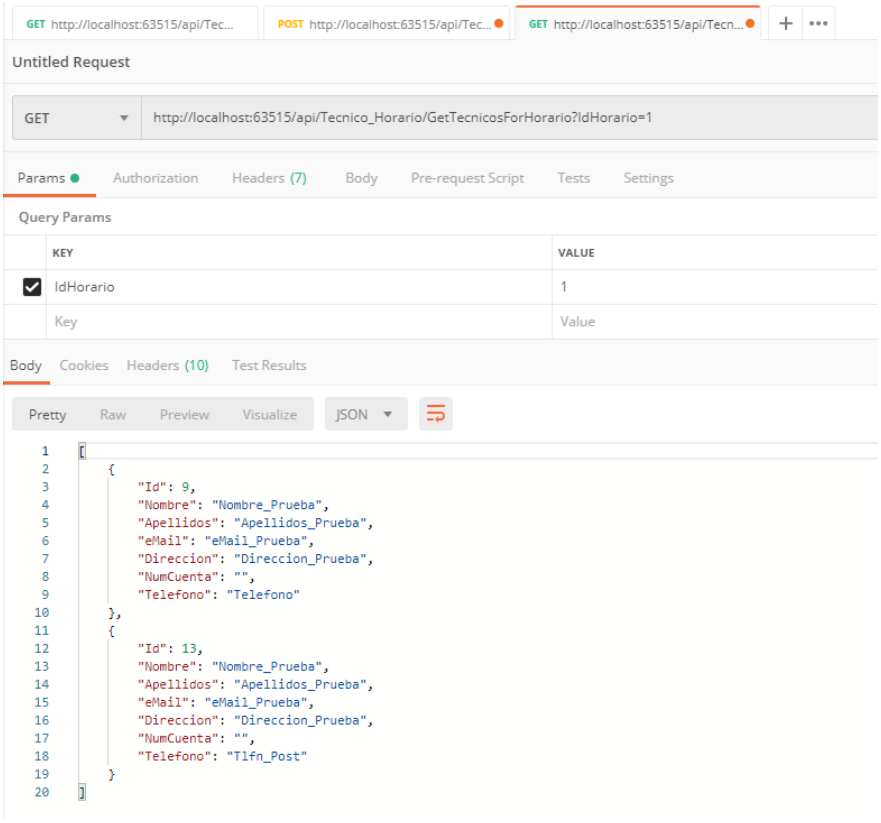


Fig 59: (GET)GetTecnicosForHorario call para *Tecnico_Horario*

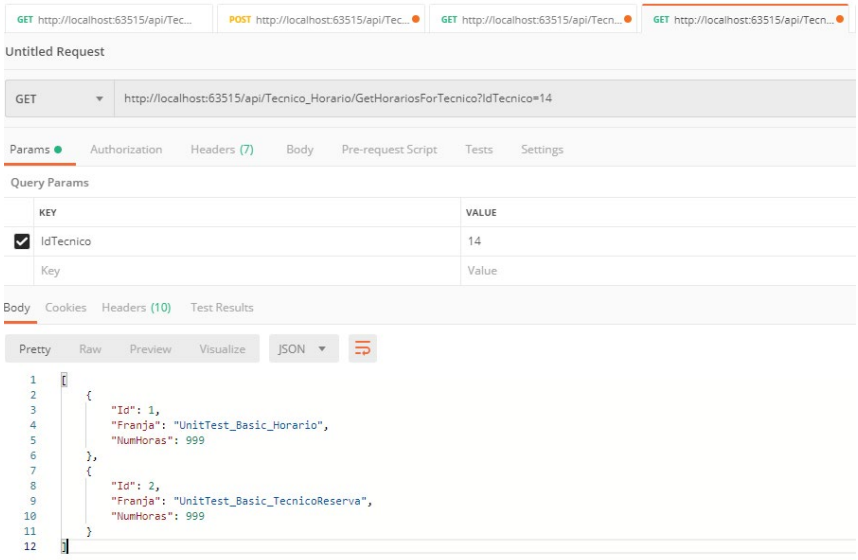


Fig 60: (GET)GetHorariosForTecnico call para *Tecnico_Horario*

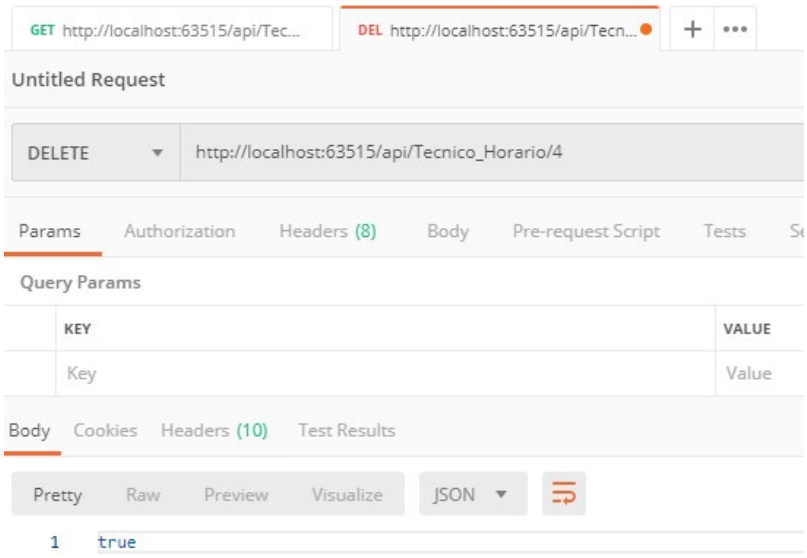


Fig 61: DELETE call para *Tecnico_Horario*

Peticiones POST,GET, (GET)GetTecnicosForTecnologia, (GET)GetTecnologiaForTecnico, DELETE para la clase *Técnico_Tecnología*

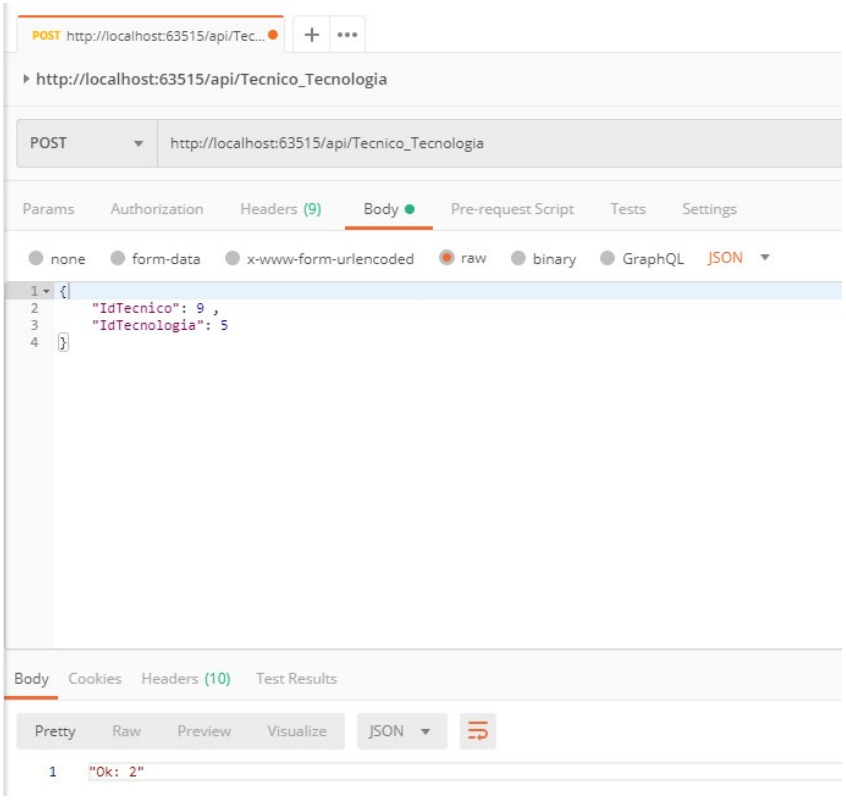


Fig 62: POST call para *Tecnico_Tecnologia*

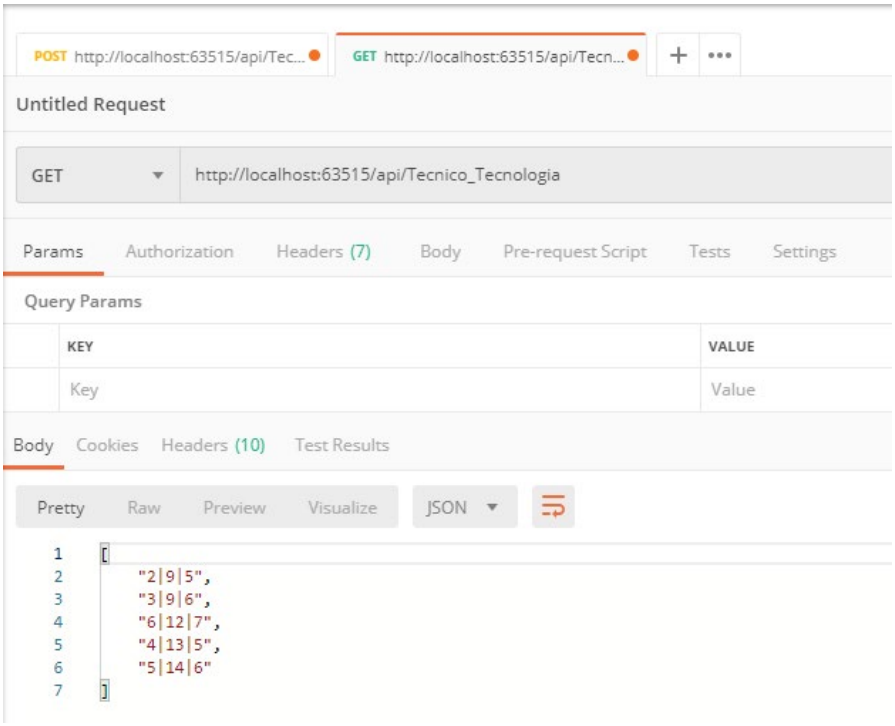


Fig 63: GET call para *Tecnico_Tecnologia*

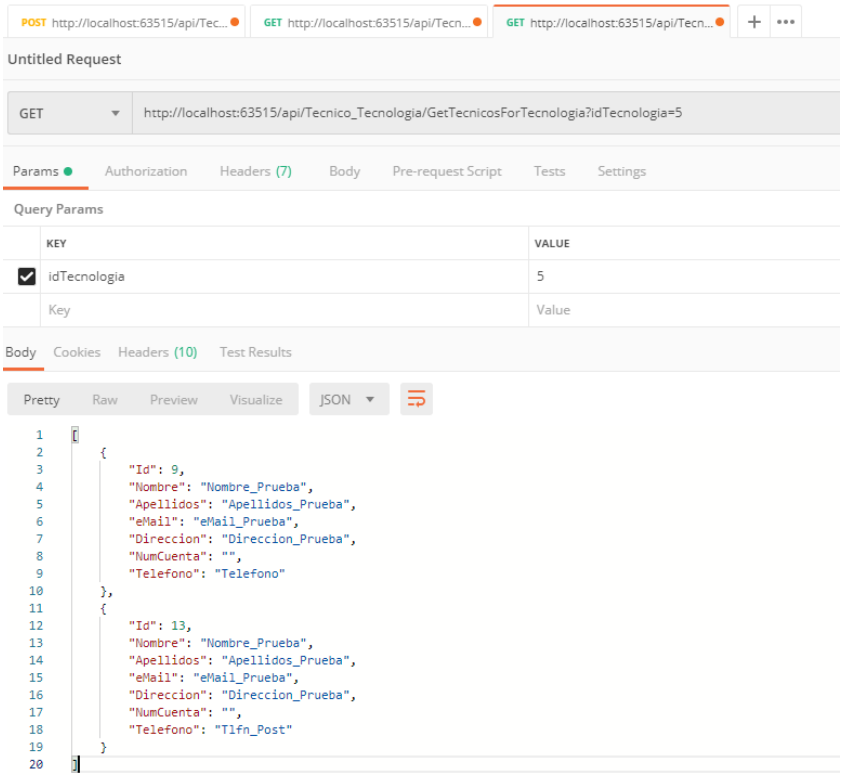


Fig 64: (GET)GetTecnicosForTecnologia call para *Tecnico_Tecnologia*

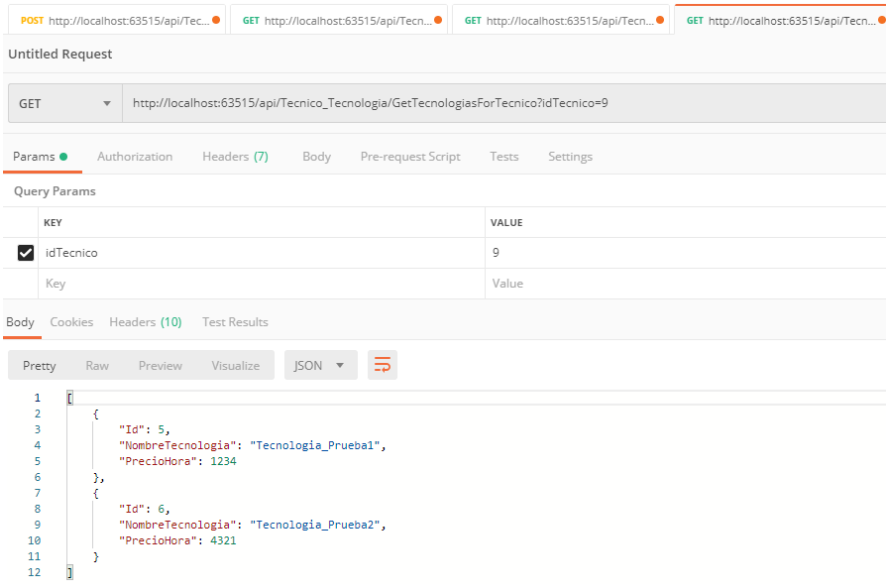


Fig 65: (GET)GetTecnologiasForTecnico call para *Tecnico_Tecnologia*

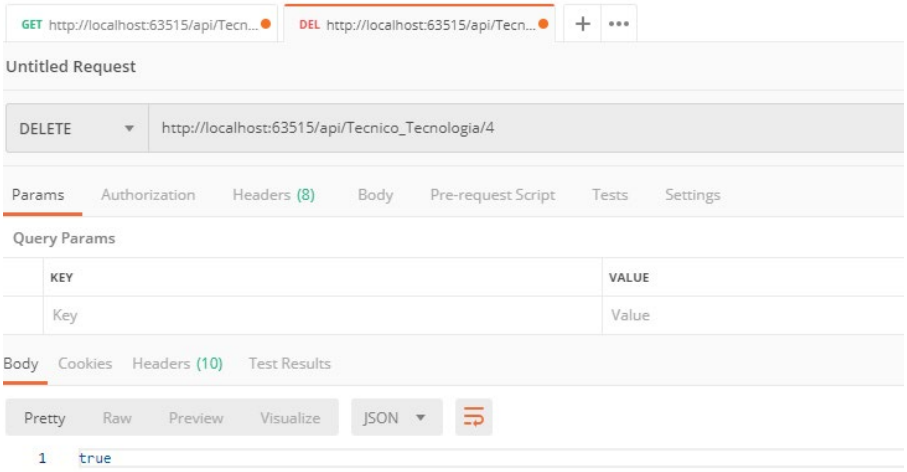


Fig 66: DELETE call para *Tecnico_Tecnologia*

Peticiones POST, GET, (GET)GetTecnicoReservaXML, (GET)GetReservasforTecnico, (GET)GetReservasforHorario, (GET)GetReservasforTecnologia, DELETE para la clase *Técnico_Reserva*

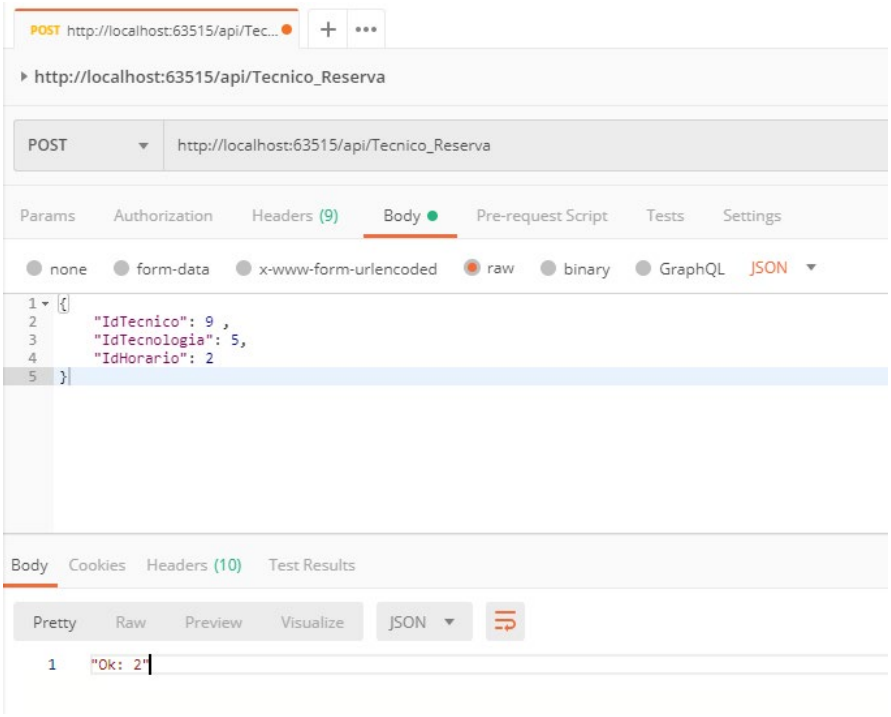


Fig 67: POST call para *Tecnico_Reserva*

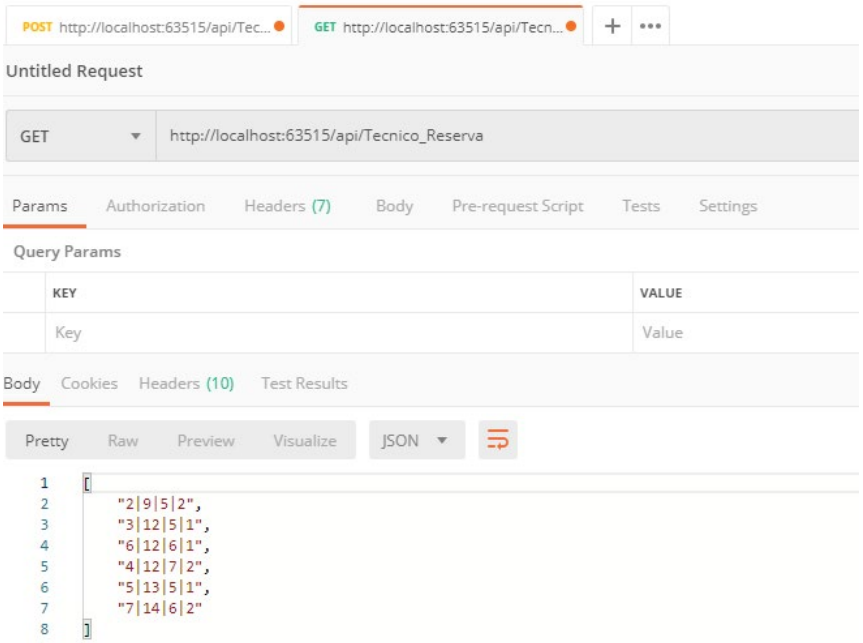


Fig 68: GET call para *Tecnico_Reserva*

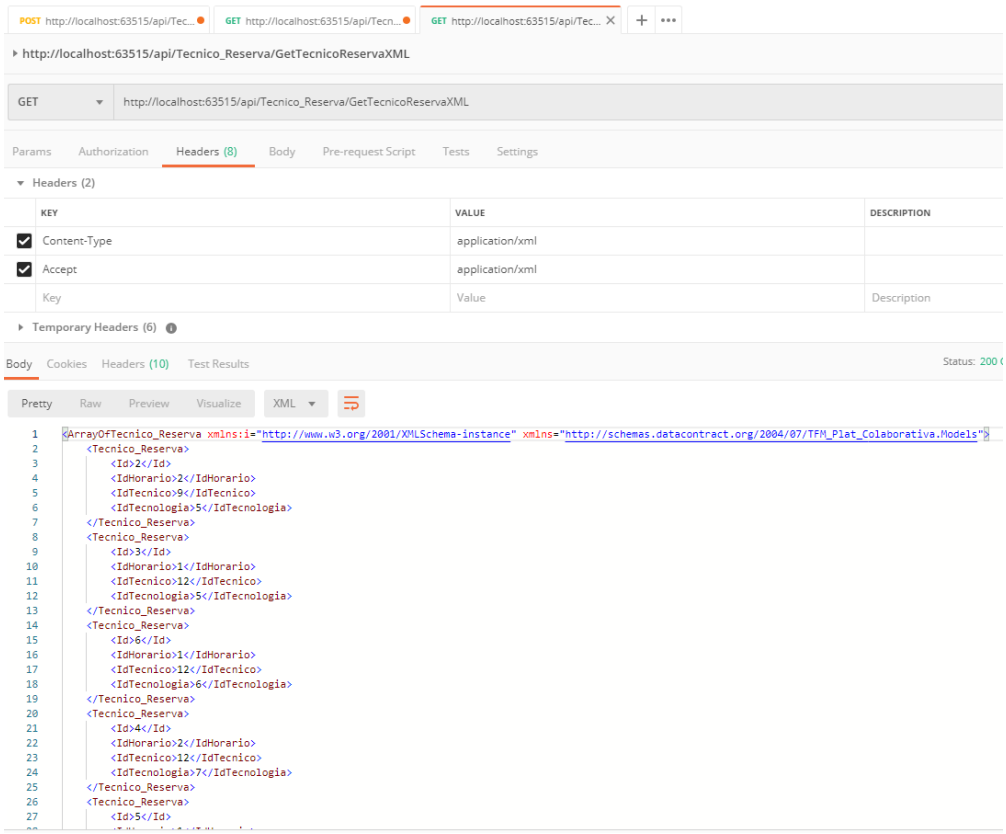


Fig 69: (GET)GetTecnicoReservaXML call para *Tecnico_Reserva*

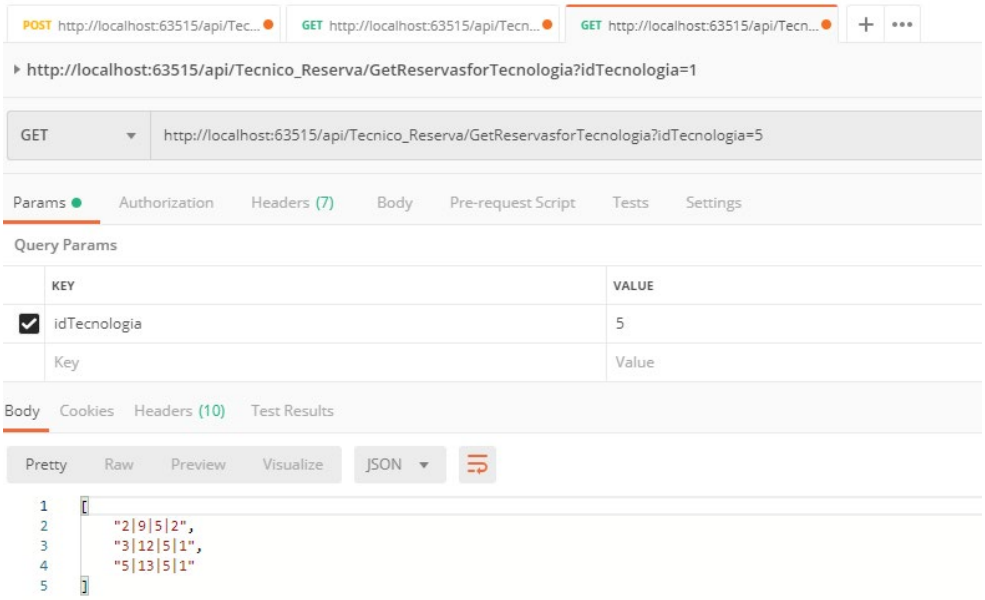


Fig 70: (GET)GetReservasForTecnologia call para *Tecnico_Reserva*

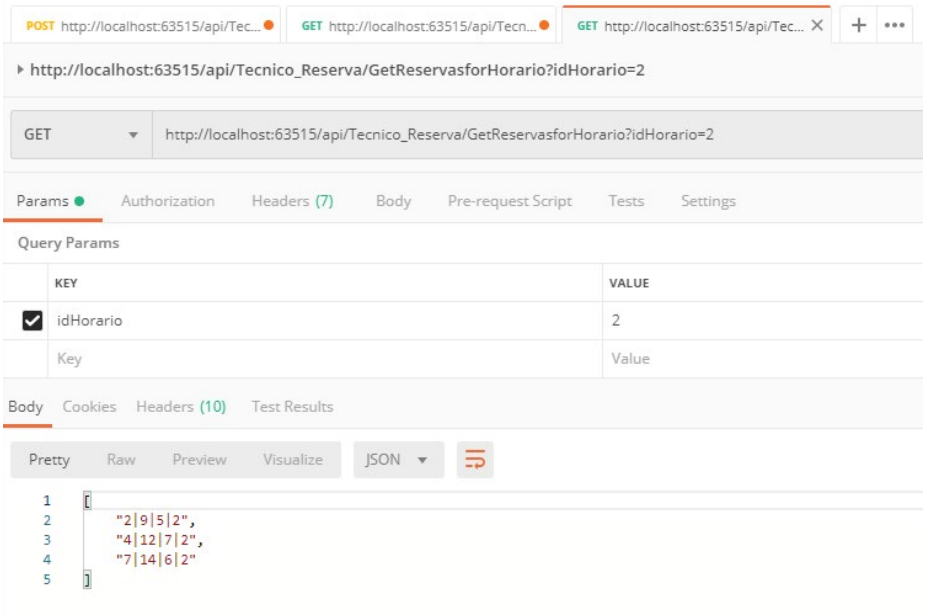


Fig 71: (GET)GetReservasForHorario call para *Tecnico_Reserva*

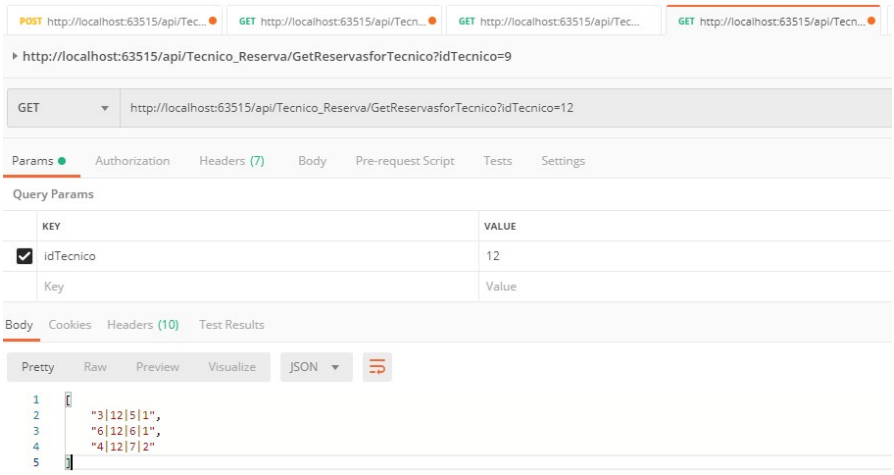


Fig 72: (GET)GetReservasForTecnico call para *Tecnico_Reserva*

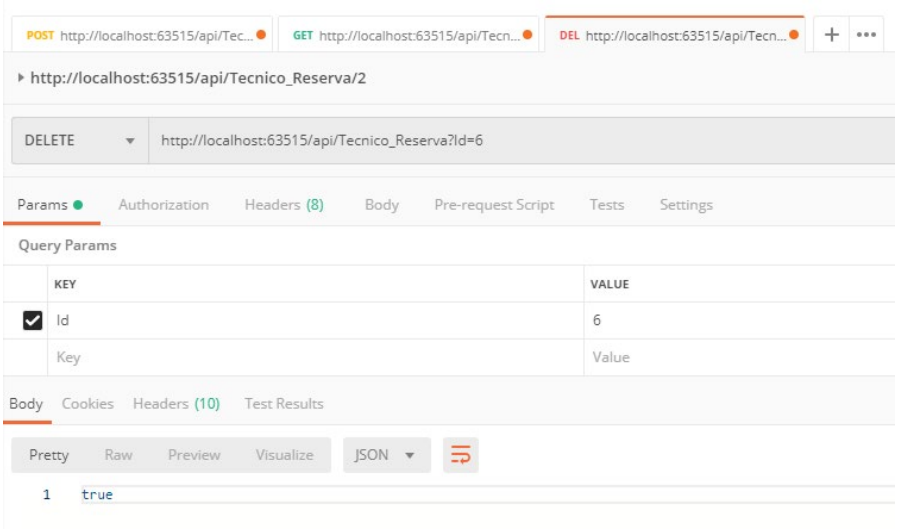


Fig 73: DELETE call para *Tecnico_Reserva*

5.8 Despliegue

Para publicar una API REST "sólo" se necesita un servidor web. En función del lenguaje en el que hayamos desarrollado nuestra API así nos servirán unos u otros, ya que es el servidor web el que deberá interpretar el código generado, así para java se utilizarán unos servidores, por ejemplo Apache, mientras que lo más habitual para desarrollos en .Net es IIS.

Vamos a suponer que la máquina donde vamos a desplegar nuestra API REST ya tiene instalado IIS, en ese caso dentro de Visual Studio haremos click con el botón derecho sobre el proyecto y en el menú contextual seleccionaremos (Publicar).

Dentro de las opciones que nos aparecen (como Azure, Linux, …) seleccionaremos IIS y en lugar de "Publicar Inmediatamente" seleccionaremos "Crear Perfil". Como podemos ver al crear el perfil, podemos desplegar nuestra API REST directamente a un servidor remoto, … en nuestro caso escogeremos "Sistema de Archivos" y en la ubicación seleccionaremos la carpeta donde queremos desplegarlo (en la mayoría de los casos, esta carpeta habrá sido creada previamente bajo C:\inetpub\wwwroot\...).

En la siguiente ventana seleccionamos la opción de configuración "Release" y en Runtime de destino seleccionaremos la que más nos convenga, aunque en nuestro caso será "Portatil" que es la utilizada para un ambiente local.

Tras guardar, comprobamos que la información que se nos muestra es correcta y damos a "Publicar", pudiendo ver el resultado de la publicación en la consola de Visual Studio.

Si la publicación ha sido exitosa, en la carpeta que seleccionamos anteriormente deberían aparecer los archivos necesarios para que nuestro servidor puede ejecutar la API REST.

Nos quedaría ya por último el iniciar nuestra API REST a través del panel de administración de IIS.

RESUMEN, CONCLUSIONES Y AMPLIACIONES FUTURAS

Como se ha podido ver a lo largo de todo el documento, las API REST se están convirtiendo (si no lo han hecho ya) en el estándar que utilizan las empresas para crear servicios, en parte por su uso eficiente y facilidad de implementación, y en parte por la simplicidad a la hora de integrar los elementos de las aplicaciones nuevas en la arquitectura ya existente.

Todas las empresas referentes utilizan un amplio abanico de APIs REST (Google, Facebook, ...) que en su mayor parte son de uso público lo cual nos permite ampliar nuestra red de conexión integrando sus servicios dentro de nuestras propias aplicaciones.

Además, hoy en día en el que *Cloud Computing* es un hecho consagrado, el uso de APIs REST se presenta como el modelo nativo para integrar con la nube, y es que la tendencia actual se basa en que las aplicaciones en la nube sigan una arquitectura de microservicios que consiste en desarrollar las aplicaciones software como una serie de pequeños servicios, cada uno ejecutándose de forma autónoma y comunicándose entre sí en la mayoría de los casos a través de peticiones HTTP. El cliente se torna cada vez más en lo que conocemos como "terminal" que lo único que hace es conectarse a estos y encargarse de presentar los datos al usuario, dejando el resto de las responsabilidades para los diferentes servicios que compongan la aplicación.

Todo lo que acabamos de comentar a gran escala queda reflejado de forma simplificada pero efectiva en la implementación de una API REST que hemos desarrollado. En dicha API se aprecian las ventajas que aporta este modelo de desarrollo, reglas que se le aplican, elementos principales y conceptos a tener en cuenta.

Obviamente la API se ha querido centrar en la parte pedagógica de la arquitectura REST, tratando de recibir llamadas de formas diversas, con información diversa (unas veces más compleja otras con tipos sencillos) y devolviendo respuestas igualmente dispares, desde tipos de datos estándar hasta datos complejos en diferentes formatos. Se ha tratado de utilizar el mayor número de conceptos aplicables de los que se han tratado dentro del máster. Destacar en este sentido las pruebas unitarias incluidas dentro de la solución.

Como futuras ampliaciones, la versatilidad y sobre todo la flexibilidad de las API REST nos permite tener pocas acotaciones a la hora de ampliar nuestros servicios REST. Si bien en nuestra API se antoja interesante el hecho de implementar la seguridad utilizando alguno de los modelos que hemos visto. Y en cuanto a funcionalidad, añadir la posibilidad de valorar a los técnicos, de forma que puedas ver sus *reviews* a la hora de contratar sería algo interesante. Pensando a mayor escala, un sistema de mensajería instantánea que avise a los técnicos cuando han sido reservados, o que implemente grupos sería una funcionalidad interesante.

BIBLIOGRAFÍA

[1]http://www.asp.net/web-api

(RESTful Services With ASP.NET MVC)

[2]http://msdn.microsoft.com/en-us/magazine/dd943053.aspx

ASP.NET MVC Forum

[3]http://forums.asp.net/1146.aspx

REST Intro - Build a REST Service in C# Visual Studio 2015

[4]https://www.youtube.com/watch?v=JeIE3jzAxHU

[5]https://www.youtube.com/watch?v=PJHyjBrGI-g&feature=youtu.be

[6]https://www.genbeta.com/desarrollo/

WCF Web HTTP Programming Model

[7]http://msdn.microsoft.com/en-us/library/bb412169.aspx

Hosting WCF Services in Asp.Net MVC Project

[8]http://forums.asp.net/t/1886640.aspx/1?Hosting+WCF+Services+in+Asp+Net+MVC+Project

[9]https://docs.microsoft.com/es-es/aspnet/core/web-api/advanced/formatting?view=aspnetcore-3.1

[10]https://www.youtube.com/channel/UCDZ7nnxAhuubIMw9fw0o-ZQ

[11]https://docs.microsoft.com/es-es/visualstudio/test/walkthrough-creating-and-running-unit-tests-for-managed-code?view=vs-2019

[12]https://docs.microsoft.com/es-es/visualstudio/test/using-microsoft-visualstudio-testtools-unittesting-members-in-unit-tests?view=vs-2019

[13]https://docs.microsoft.com/es-es/visualstudio/test/using-the-assert-classes?view=vs-2019

[14]https://docs.microsoft.com/es-es/visualstudio/test/using-the-assert-classes?view=vs-2019

[15]https://docs.microsoft.com/en-us/aspnet/web-api/overview/testing-and-debugging/unit-testing-with-aspnet-web-api

[16]https://www.linkedin.com/pulse/even-config-file-dependency-unit-test-which-you-mock-viswanathan/

[17]https://forums.asp.net/t/1173572.aspx?preventing+duplicate+entry+for+a+given+foreign+key+in+db+table

[18]https://www.ttandem.com/blog/la-economia-de-las-apis-ya-esta-aqui/

[19]https://www.redhat.com/es/topics/api

[20]https://www.paradigmadigital.com/dev/definir-prototipar-api-rest/

[21]https://elbauldelprogramador.com/buenas-practicas-para-el-diseno-de-una-api-restful-pragmatica/#1-requisitos-fundamentales-para-la-api

[22]https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

[23]https://blog.restcase.com/4-most-used-rest-api-authentication-methods/

[24]https://blog.restcase.com/restful-api-authentication-basics/

[26]https://www.itdo.com/blog/cual-es-el-mejor-metodo-de-autentificacion-en-un-api-rest/

[27]https://www.arsys.es/blog/programacion/autenticacion-api-rest-token/

[28]https://www.monografias.com/docs114/telecomunicaciones-arquitectura-cliente-servidor/telecomunicaciones-arquitectura-cliente-servidor.shtml#arquitecta

[29]https://blog.nubox.com/que-es-una-api-y-como-puede-mejorar-mi-negocio

[30]http://ramonabadypuntonet.org/2017/04/01/que-son-los-unit-test/

[31]https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/el-papel-del-unit-test-en-el-desarrollo-de-software/

[32]**https://luisdavidxamshap.wordpress.com/2018/07/13/publicar-api-rest-net-core-en-iis-local/**