Documentación Covid API

Autor:

Álvaro Maleno Alférez

Sumario

1. Definiciones y especificación de requerimientos

1.1 Definición general del proyecto

1.1.1 Qué aplicación se ha desarrollado

Las funcionalidades principales de la aplicación son dos: la primera, garantizar el seguro almacenamiento de los datos de usuario y la segunda, ofrecer datos fiables y "a la carta" acerca de la evolución de la pandemia mundial de Covid-19. Esto último quiere decir que los usuarios podrán solicitar los datos para el país o los países que deseen y dentro de un período que transcurre entre dos fechas elegibles también por el usuario.

1.1.2 Para qué se ha desarrollado la aplicación

Lo primero, para ofrecer los datos bajo demanda y solamente aquellos que se deseen obtener con la mayor prontitud posible y con el menor número de complicaciones para el usuario que se ha podido conseguir. Y segundo, e igual de importante, garantizando la seguridad en el tratamiento de los datos. Por supuesto, la posibilidad de emplear la seguridad en otras aplicaciones así como el acceso a los datos tratados aquí ha supuesto una parte importante del esfuerzo generado.

1.1.3 ¿Quién o quiénes son los usuarios?

Existen dos tipos de usuario:

1. Todos aquellos desarrolladores o empresas de desarrollo que requieran incorporar cierta seguridad en el trato de los datos del proceso de autentificación de usuario. Direcciones de correo electrónico personales, contraseñas, direcciones, números de teléfono, etc. Todo ello se encuentra automatizado ya en la API de seguridad que contiene este proyecto a modo de bloque. Cualquier dato que sea necesario resulta fácilmente desarrollable con una mínima cantidad de trabajo. Por tanto, todo aquel que requiera implementar un proceso de autentificación de usuario es, a su vez, un posible usuario de esta aplicación.

2. Científicos de datos que requieran de un acceso ágil y veloz a la información acerca de la evolución de la pandemia global, páginas web que deseen mostrar información en tiempo real, cargando solamente los datos que les son imprescindibles lo más rápida y menos pesadamente posible, desarrolladores que quieran ofrecer un producto sencillo a agencias de viajes, cuadernos de bitácora - blogs – o pequeños negocios que precisen o deseen, como servicio adicional y forma de solidarizarse con las víctimas mostrar los datos desde su escaparate web. En definitiva, todo aquel que ofrezca una presencia en línea de sus productos, negocios, artículos, etc.

En cualquier caso el nivel de conocimientos informáticos relativos a la programación ha de situarse entorno a un rango que varía entre principiante y medio. Si bien cualquier principiante puede fácilmente acceder a su uso, aquellos que se encuentren en un nivel algo más avanzado encontrarán también ciertas ventajas y posibles aplicaciones distintas a las que ya posee.

1.2 Especificación de requerimientos

1.2.1 Requisitos generales

Tras el diseño de la aplicación subyacen varias ideas. La primera, ya común, relativa a la reutilización del software. Diseñar y programar aplicaciones informáticas es muy costoso. Si nos imaginamos la cantidad de trabajo que es necesario para construir una pequeña vivienda nos estaríamos acercando más o menos exactamente, dependiendo del caso, a la construcción de una aplicación informática. Así, poseer componentes que puedan ser fácilmente desplegables y que funcionen como bloques de construcción – como las paredes de un edificio prefabricado – no solamente se hace imprescindible para que una empresa pueda funcionar, sino que es muy deseable, ya que con cada iteración de desarrollo sobre el producto éste puede perfeccionarse.

La arquitectura de micro servicios surge como paradigma de enfoque y propuesta práctica que trata de facilitar la composición de aplicaciones. Cada bloque, al ser independiente de los demás, puede funcionar por sí mismo para una o varias aplicaciones. Puede copiarse con facilidad y puede crecer manteniendo la integridad sin socavar su rehusabilidad.

La segunda idea a tener en cuenta para este proyecto ha sido el enfoque en la usabilidad y la facilidad de uso para el usuario. Esto quiere decir que la aplicación debe de tener suficiente flexibilidad como para cubrir, de la manera más simple posible para el usuario, todas sus posibles demandas. El modelo de datos a emplear, así como el número de pasos a seguir han sido pensados para que no haya ninguna labor que resulte compleja. La automatización en el despliegue, funcionamiento y actualización ha sido otra herramienta empleada para facilitar su uso. Por último, la velocidad en la transmisión de la información es un punto clave que ha sido tratado y puesto en valor en todo momento.

Por supuesto, la salvaguarda de los datos de las personas que deciden depositar su confianza en la aplicación ha sido un punto crucial en torno al cual se ha erigido este proyecto. Por ello, un componente de seguridad con cifrado variable en el tiempo se ha desarrollado sin escatimar esfuerzos.

1.2.2 Requisitos funcionales

1.2.2.1 Seguridad

1. Proceso de alta de usuario cifrado. Jamás se recibirán o enviarán datos sensibles sin encriptar.

2. Obtención de clave de cifrado para realizar el registro de un nuevo usuario. Mediante una url única se le ofrecerá al usuario la posibilidad de obtener una clave pública de cifrado perecedera con la cual encriptar sus datos por primera vez.

3. Poseer unas claves pública y privada para uso propias de la aplicación que variarán cada determinado número de usos.

4. Vincular una pareja de claves privada y pública únicas con cada usuario. Cada usuario poseerá, dentro del sistema, sus propias claves de encriptado y desencriptado.

5. Otorgar su correspondiente clave pública a cada usuario. Cada usuario registrado deberá poder conservar una clave pública con la que encriptar cada una de sus autentificaciones.

6. Recibir peticiones encriptadas de autentificación de usuario.

7. Validar la autentificación de un usuario desencriptando la comunicación recibida con su clave privada única.

8. Almacenar las claves pública y privada de usuario encriptadas con una clave propia de la aplicación que será perecedera.

9. Actualizar la encriptación de las claves de usuario cada vez que la aplicación cambia de claves pública y privada con las que funcionar.

10. Cambiar de claves pública y privada propias de la aplicación cada cierto número de usos.

11. Otorgar un token JWT perecedero cada 24 horas. Se trata de disminuir la necesidad de exponer la información sensible de usuario y facilitar el acceso a los datos del sistema.

12. El proceso de generación de tablas de usuario y de aplicación en base de datos deberá estar automatizado.

1.2.2.2 Organización y obtención de los datos

1. No se podrá acceder a los mismos sin haber sido dado de alta en el sistema y estar autentificado.

2. El modelo de datos deberá estar lo más simplificado posible.

3. Todas las tablas necesarias para el funcionamiento de la aplicación se generarán automáticamente.

4. La información externa procedente únicamente de fuentes oficiales será recogida de forma automática por la aplicación que se encargará, también, de actualizarla en su base de datos.

5. Toda petición de información se deberá servir en la mayor brevedad de tiempo posible. Para ello será imprescindible la implementación de una memoria caché que se ocupe de los volúmenes grandes de datos.

6. Los datos se servirán dentro de un rango de fechas elegible por el usuario.

7. El usuario podrá escoger qué separador usar para las fechas de inicio y fin.

8. El formato de fecha será el europeo – dd/MM/yyyy.

9. Los datos se servirán para uno, varios o todos los países disponibles en base de datos.

10. Se podrá solicitar una lista que contenga todos los países disponibles para consulta.

11. Se podrá solicitar una lista que contenga todas las fechas disponibles para consulta.

1.2.2.3 De carácter más General

1. Las peticiones se atenderán en formato JSON.

2. El paralelismo será empleado con frecuencia con el objetivo de acelerar los procesos.

3. Existirán una API rest para la seguridad, una para el acceso y almacenamiento de los datos y una tercera para interactuar con el usuario.

1.2.3 Información general acerca del proyecto.

Todas las partes componentes de este proyecto, así como el modelo de datos y las funcionalidades han sido desarrolladas de manera original. Es decir, no existía un proyecto previo del que haya heredado funcionalidad o característica alguna.

Los métodos de programación, si bien son recientes – como la programación asíncrona o el empleo de micro servicios – han sido escogidos cuidadosamente con la idea de que el código no deba de sufrir ningún cambio con futuras actualizaciones de las tecnologías de las que depende. Esto quiere decir que se aboga por la sencillez en la escritura de código.

1.2.4 Alcance y limitaciones de las funcionalidades de este proyecto.

Si bien el proyecto resulta ambicioso, fundamentalmente por dos motivos – la necesidad de que todos los procesos se encuentren automatizados y el deseo de establecer una seguridad robusta – han podido cumplirse todos los requisitos funcionales impuestos.

Cabe realizar una mención especial a la parte de seguridad del mismo sistema, pues, que disponga de un cifrado de datos y claves que varía en el tiempo de manera automática ha supuesto un esfuerzo considerable. No obstante, es también el principal punto a tratar si se habla de falta de un mayor desarrollo, ya que la aplicación se encuentra configurada únicamente para guardar un número determinado de campos relativos a la información de usuario. Si bien esta característica resulta fácilmente ampliable, su realización ha quedado fuera de este proyecto.

1.3 Procedimientos de instalación y prueba

1.3.1 Entorno operativo

La aplicación ha sido desarrollada para funcionar sobre dos sistemas operativos: Windows y Linux.

- Windows: Se trata de un sistema operativo para ordenadores desarrollado y distribuido por Microsoft, de licencia privativa y cuya instalación contiene un conjunto de aplicaciones de uso común.

- GNU/Linux: Se trata de la denominación que reciben una serie de sistemas operativos con núcleo Unix. Son multiplataforma, multiusuario y multitarea y de licencia libre. Los proyectos que lo componen son, entre otros, GNU y Linux.

Para que el desarrollo y la implementación de la aplicación sea independiente del sistema operativo escogido tanto por quien la desarrolle como por quien la mantenga y, asegurar que en un futuro los medios empleados no derivarán en coste alguno por convertirse en privativos, se han escogido dos tecnologías con licencia de código abierto y gratuito.

Para almacenar los datos el motor de Postgresql, por rapidez, sencillez y gratuidad. Mientras que, para el desarrollo de la aplicación y su ejecución .Net Core, la versión de código abierto del entorno de desarrollo de Microsoft.

- Postgresql: Se trata de un gestor de base de datos relacional de código abierto. Se ha publicado bajo licencia PostgreSQL. Ésta es similar a las licencias BSD y MIT.

- .Net Core: Es un marco de trabajo informático gratuito y de código abierto que funciona en los sistemas operativos Windows, Linux y MacOS. Ha sido desarrollado por Microsoft bajo licencia MIT.

De forma adicional se han empleado ciertas herramientas dentro del marco de trabajo que facilitan la labor de programación, como el Nuget de .Net Newtonsoft, la transmisión de datos y modelado de los mismos

otras, como JSON y la dismiución de autentificaciones mediante la generación de token, por último, como JWT.

- Newtonsoft: Es un marco de trabajo para .Net orientado a la alta eficiencia en el tratamiento y procesado de archivos JSON.

- JSON: Se trata de un formato de texto sencillo utilizado en el intercambio de datos entre programas informáticos. Se considera un formato independiente del lenguaje de programación JavaScript, a pesar de consistir en una notación literal de sus objetos. Su nombre completo es "JavaScript Object Notation".

- JWT: Es un estándar de tokens de acceso que facilitan la propagación de identidad y privilegios de usuario a lo largo de una o varias aplicaciones sin necesidad de realizar sucesivas autentificaciones. Estos tokens son perecederos.

En relación al diseño y modelado de datos se ha optado por seguir las recomendaciones del estándar de código abierto OpenAPI, por lo que el mismo ha sido realizado bajo su marco de trabajo.

- OpenAPI: Actualmente mantenida por la Linux Foundation esta especificación ha sido creada para generar archivos de interfaz legibles por máquina con el objetivo de describir, consumir y visualizar servicios web RESTfull.

Para generar la documentación relativa a los diagramas UML y diseñar, con ellos, la aplicación, se ha optado por la utilización – por su gratuidad – de draw.io .

- draw.io: Se trata de una aplicación web, de uso gratuito, previo registro que facilita la generación de diferentes tipos de diagramas desde plantillas y símbolos incluídos.

Todo proyecto informático requiere de una adecuada gestión de versiones de la aplicación y un correcto organizador del mismo. Github es la herramienta que aúna ambas funcionalidades interconectándolas de una manera cómoda para el equipo de desarrollo.

- Github: Se trata de una aplicación para alojar proyectos que utiliza el sistema de control de versiones Git. Está escrito en el lenguaje de programación Ruby on Rails y actualmente pertenece a Microsoft.

- Git: Es un programa informático de control de versiones. Fue diseñado por Linus Torvalds, orientándose hacia la eficiencia y la confiabilidad en el mantenimiento de las distintas aplicaciones informáticas, especialmente cuando éstas constan de grandes archivos de código fuente.

- Docker: Se trata de un servicio de contenerización de aplicaciones gracias al cual el ambiente de desarrollo y funcionamiento de la aplicación puede aislarse del equipo en el que esta se ejecute. Es un servicio open source desarrollado por Google.

Hay también que mencionar LibreOffice y el lenguaje de marcado markdown como herramientas empleadas a la hora de elaborar toda la documentación generada así como la aplicación web designrr para maquetar resultados en formato pdf.

1.3.2 Metodología de trabajo

Como metodología de desarrollo se ha escogido Kanban. Su significado es el de “letrero” en japonés. Fue desarrollado para hacer llegar la información desde el punto de venta del producto hasta la fábrica del mismo de manera inmediata. Cada vez que un cliente retiraba un producto de su estante una tarjeta que lo representaba era enviada al fabricante que pasaba a crear un producto idéntico que lo reemplazase en tienda, devolviendo, así, la tarjeta al lugar de origen de la misma.

Al adaptarse al ámbito informático este sistema de tarjetas pasó a significar el desarrollo de una funcionalidad nueva de la aplicación a la que se refieren o, simplemente, la necesidad de subsanar un error. De manera más avanzada, las aplicaciones son desarrolladas funcionalidad a funcionalidad. Cada una de ellas

puede representarse en una tarjeta virtual que sigue un flujo desde preparada para ser realizada, hasta realizada, pasando por siendo realizada. Cada vez que se ha finalizado una funcionalidad nueva, ésta se prueba y, si todo marcha bien, se pasa a la siguiente.

Puede identificarse aquí el denominado método "bola de nieve". Consistiría en realizar una mínima parte funcional del programa – igual que una pequeña bola de nieve – a la que se le irían sumando cada vez más funcionalidades hasta haber completado toda la aplicación.

1.3.2 Requisitos no funcionales

0001	*Eficiencia en el servicio de los datos relativos a la pandemia*
1.0	*1.0 - Junio 2021*
Dependencias	*Base de Datos Relacional de la Aplicación* *.Net Core 3.0 y lenguaje de programación C#* *PostgreSql* *Sistema Operativo* *Técnica de programación*
Descripción	*El servicio de los datos se ha de realizar con la mayor prontitud posible. Éste debe de responder en un tiempo menor a 400 ms.*
Importancia	*Alta*
Prioridad	*Media*
Estado	*Fase final del desarrollo.*

0002	*Capacidad para gestionar peticiones simultáneas*
1.0	*1.0 - Junio 2021*
Dependencias	*Base de Datos Relacional de la Aplicación* *.Net Core 3.0 y lenguaje de programación C#* *PostgreSql* *Sistema Operativo* *Técnica de programación*
Descripción	*No debe de existir error alguno en caso de que n peticiones se realicen al mismo tiempo. Como mínimo debe de ser capaz de gestionar 50 peticiones simultáneas.*
Importancia	*Alta*
Prioridad	*Media*
Estado	*Fase final del desarrollo.*

0003	*Seguridad en el almacenamiento de los datos*
1.0	*1.0 - Junio 2021*
Dependencias	*Base de Datos Relacional de la Aplicación* *.Net Core 3.0 y lenguaje de programación C#* *PostgreSql* *Sistema Operativo* *Técnica de programación*
Descripción	*Los datos sensibles relacionados con el almacenaje del usuario y su contraseña, deben de ser seguros y permanecer aislados de la aplicación tanto como sea posible. Para ello serán encriptados con una clave privada única para el usuario y ésta clave, a su vez, se encriptará con una clave de aplicación que variará cada determinado número de usos.*
Importancia	*Máxima*
Prioridad	*Alta*
Estado	*Fase final del desarrollo.*

0004	*Facilidad en el uso*
1.0	*1.0 - Junio 2021*
Dependencias	*Base de Datos Relacional de la Aplicación* *.Net Core 3.0 y lenguaje de programación C#* *PostgreSql* *Sistema Operativo* *Técnica de programación*
Descripción	*La aplicación debe de permitir una facilidad de empleo garantizando que su uso sea lo más sencillo posible. De ahí que únicamente admita un tipo de petición que centralice todas las posibilidades de demanda de datos. El modelo de los mismos se ha de simplificar hasta el extremo.*
Importancia	*Máxima*
Prioridad	*Alta*
Estado	*Fase final del desarrollo.*

0005	*Disponible siempre*
1.0	*1.0 - Junio 2021*
Dependencias	*Base de Datos Relacional de la Aplicación* *.Net Core 3.0 y lenguaje de programación C#* *PostgreSql* *Sistema Operativo* *Técnica de programación*
Descripción	*Debe de ser posible usar la aplicación a cualquier hora del día cualquier día del año. En caso de falla ésta no debe detenerla.*
Importancia	*Máxima*
Prioridad	*Alta*
Estado	*Fase final del desarrollo.*

0006	*Independencia del SO*
1.0	*1.0 - Junio 2021*
Dependencias	*Base de Datos Relacional de la Aplicación* *.Net Core 3.0 y lenguaje de programación C#* *PostgreSql* *Sistema Operativo* *Técnica de programación*
Descripción	*La aplicación debe funcionar en más de un sistema operativo. En este caso, Windows y Linux.*
Importancia	*Máxima*
Prioridad	*Alta*
Estado	*Fase final del desarrollo.*

1.3.3 Obtención e instalación.

La aplicación puede obtenerse desde el repositorio siguiente:

- **https://github.com/alvaroMaleno/CovidAPI**

Para la instalación pueden seguirse los distintos manuales de usuario accesibles desde el archivo README.md del directorio raíz. También puede seguirse el descrito a continuación.

1. Puesta en marcha

Instalando dependencias

Para que este API pueda funcionar son necesarias las siguientes dependencias:

SDK de .Net Core 3.1

ASP de .Net Core 3.1

.Net Core 3.1

PostgreSQL 11.8

Pueden seguirse los siguientes tutoriales:

https://docs.microsoft.com/es-es/dotnet/core/install/linux

https://www.postgresql.org/docs/11/installation.html

2. Arrancando la aplicación

Primero se ha de inicializar el componente DataAccess_API. Se encuentra en el directorio ***APIs*/DataAccess_API** .

DataAccess_API:

El primer paso será la configuración de una base de datos sobre la cual realizar la persistencia y consulta de la información referente a los nuevos usuarios. Desde la página oficial de PostgreSQL puede consultarse una guía completa: https://www.postgresql.org/docs/11 .

Una vez generada una base de datos, será necesario configurar el programa para que acceda a la misma. Para ello se ha generado un archivo .json en el cual es posible introducir las credenciales y url de la base de datos. La API se encargará de gestionar esa información tras el primer arranque sin que el usuario tenga que realizar ninguna tarea adicional.

La ruta al archivo es **/APIs/DataAccess_API/DAOs/Connection/connectionProperties.json** . Se trata de sustituir la información de cada campo por la generada en el paso anterior:

```
{
   "server": "introducir url",
   "port": "introducir puerto",
   "userId": "introducir usuario",
   "pass": "introducir contraseña",
   "dataBase": "introducir base de datos"
}
```

Levantar el componente:

1. Navegar hasta la ruta **APIs/DataAccess_API**.
2. Ejecutar el comando **dotnet build** desde la consola de comandos. Ref: https://docs.microsoft.com/es-es/dotnet/core/tools/dotnet-build

3. Ejecutar el comando **dotnet run** desde la consola de comandos. Ref: https://docs.microsoft.com/es-es/dotnet/core/tools/dotnet-run
4. Es posible sustituir los dos pasos anteriores por al arranque desde un IDE.
5. Esperar varias horas. La aplicación generará todas las bases de datos necesarias para su empleo e insertará todos los datos necesarios previa consulta de los mismos a organismos externos a la misma. Si tras varios minutos no se observase la creación de ninguna tabla en base de datos se recomienda detener la aplicación y comprobar la conexión al servidor de base de datos.

Una vez en marcha el microservicio anterior será imprescindible arrancar el componente encargado de la seguridad Security_API. Se encuentra en el directorio ***APIs*/Security_API** .

Security_API:

Al igual que en el componente anterior, el primer paso será la configuración de una base de datos sobre la cual realizar la persistencia y consulta de la información referente a los nuevos usuarios. Desde la página oficial de PostgreSQL puede consultarse una guía completa: https://www.postgresql.org/docs/11 .

Una vez generada una base de datos, será necesario configurar el programa para que acceda a la misma. Para ello se ha generado un archivo .json en el cual es posible introducir las credenciales y url de la base de datos. La API se encargará de gestionar esa información tras el primer arranque sin que el usuario tenga que realizar ninguna tarea adicional.

La ruta al archivo es **/APIs/Secutiry_API/DAOs/Connection/connectionProperties.json**. Se trata de sustituir la información de cada campo por la generada en el paso anterior:

```
{
   "server": "introducir url",
   "port": "introducir puerto",
   "userId": "introducir usuario",
   "pass": "introducir contraseña",
   "dataBase": "introducir base de datos"
}
```

Levantando la Aplicación:

1. Navegar hasta la ruta **APIs/Security_API**.
2. Ejecutar el comando dotnet build desde la consola de comandos. Ref: https://docs.microsoft.com/es-es/dotnet/core/tools/dotnet-build
3. Ejecutar el comando dotnet run desde la consola de comandos. Ref: https://docs.microsoft.com/es-es/dotnet/core/tools/dotnet-run
4. Es posible sustituir los dos pasos anteriores por al arranque desde un IDE.
5. Esperar varios minutos. La aplicación generará todas las bases de datos necesarias para su empleo e insertará todos los datos necesarios. Si tras varios minutos no se observase la creación de ninguna tabla en base de datos se recomienda detener la aplicación y comprobar la conexión al servidor de base de datos.

Por último solamente resta poner en funcionamiento el componente unitario Covid_API:

Navegar hasta la ruta **APIs/Covid_API**.

1. Ejecutar el comando dotnet build desde la consola de comandos. Ref: https://docs.microsoft.com/es-es/dotnet/core/tools/dotnet-build
2. Ejecutar el comando dotnet run desde la consola de comandos. Ref: https://docs.microsoft.com/es-es/dotnet/core/tools/dotnet-run
3. Es posible sustituir los dos pasos anteriores por al arranque desde un IDE.

1.3.4 Pruebas de software

Para esta aplicación se ha realizado un enfoque orientado al comportamiento. El planteamiento es muy simple: la única prueba válida es la que se le realiza a la aplicación en funcionamiento. Quedan así descartadas técnicas de prueba conocidas como de mockeo y todas aquellas que no ejecuten el algoritmo que compone la aplicación fuera de un caso de uso real de la misma pues poco o nada demuestran.

La ventaja de poseer distintos componentes es que pueden probarse separadamente facilitando la especificación de las pruebas al realizar pues, al centrar el objetivo se previenen desviaciones innecesarias y surgen más puntos a comprobar.

1.3.4.1 Pruebas al componente de Seguridad

El Método Get para obtener clave pública previa al registro de usuario funciona.	**001**	
	Prueba de funcionamiento	**Obligatoria**
Descripción: Se han de realizar varias peticiones Get a la url identificativa del servicio. Éstas deben devolver una clave pública con la que encriptar la información de usuario antes de solicitar un alta. La clave debe de ser la misma para los primeros X usos.		
Prerrequisitos: Haber instalado todas las dependencias. Poner en marcha el componente Security_API de la aplicación.		
Pasos:	**Datos**:	
Introducir url	url	
Obtener respuesta	Clave Pública de Aplicación	
Repetir Proceso		

Registro de usuario satisfactorio	**002**	
	Prueba de funcionamiento	**Obligatoria**
Descripción: Se realizará el registro de varios usuario, con mismo y distinto correo electrónico. En caso de estar registrado se comprobará la actualización de los datos. En caso de no estarlo, se recibirá una clave pública única para cada usuario.		
Prerrequisitos: Haber instalado todas las dependencias. Poner en marcha el componente Security_API de la aplicación. Obtener la clave pública de encriptado de la aplicación mediante método Get.		
Pasos:	**Datos**:	
Obtener Clave Pública de Aplicación.	Clave Pública	
Encriptar los datos de usuario con la clave pública obtenida.	Modelo Usuario	
Establecer el elemento new del modelo de usuario a true.	Modelo Usuario	
Enviar petición POST	Modelo Usuario. Url	
Recibir respuesta	Clave Pública	
Repetir el proceso.		

Validación de usuario	**003**	
	Prueba de funcionamiento	**Obligatoria**
Descripción: Con un grupo de usuarios creados con anterioridad se realizará su autentificación obteniendo, true en caso afirmativo y false en caso negativo.		
Prerrequisitos: Haber instalado todas las dependencias. Poner en marcha el componente Security_API de la aplicación. Poseer un grupo de usuarios registrados cada uno con su clave pública única.		
Pasos:	**Datos**:	
Encriptar datos de usuario con clave única.	Clave Pública. Modelo Usuario	
Establecer el elemento new del modelo de usuario a false.	Modelo Usuario	
Añadir la clave pública única al modelo de usuario.	Modelo Usuario. Clave Pública.	
Enviar petición POST.	Modelo Usuario. Url	
Recibir respuesta.	Clave Pública	
Repetir el proceso.		

La claves pública de aplicación varía cada cierto número de usos así como la encriptación en base de datos de los datos de usuario.	**004**	
	Prueba de funcionamiento	**Obligatoria**
Descripción: Se realizará la petición de obtención de clave pública de aplicación y/o se registrarán o autentificarán usuarios tantas veces como sean necesarias para que se produzca el cambio de claves privada y pública de aplicación. Por defecto está establecido que sean 100.		
Prerrequisitos: Haber instalado todas las dependencias. Poner en marcha el componente Security_API de la aplicación.		
Pasos:	**Datos**:	
Obtener Clave Pública de Aplicación y/o registrar o autentificar usuario.	Clave Pública. Modelo Usuario	
Repetir tantas veces como sea necesario.	Clave Pública. Modelo Usuario	
Comprobar que la respuesta de Obtención de clave pública ha variado.	Clave Pública	
Comprobar que en base de datos se han producido las actualizaciones necesarias.	Modelo Usuario. Conexión Base de datos.	
Repetir el proceso.		

1.3.4.2 Pruebas al componente de acceso a Datos

Devuelve País Solicitado	**001**	
	Prueba de funcionamiento	**Obligatoria**
Descripción: Se realizarán diez peticiones para diez países distintos y se comprobará que en cada una de las respuestas los datos pertenezcan al país solicitado.		
Prerrequisitos: Haber instalado todas las dependencias. Poner en marcha el componente DataAcccess_API de la aplicación. Haber esperado el tiempo suficiente para que los datos hayan sido insertados en base de datos.		
Pasos:	**Datos**:	
Crear Petición introduciendo el código ISO2 de país en el campo correspondiente. Es necesario introducir, también, el nombre del método correspondiente al que se dirige la aplicación.	Modelo POST	
Enviar Petición	Modelo POST	
Recibir Respuesta	Modelo de salida de datos	
Comprobar Respuesta	Modelo de salida de datos	
Repetir Proceso		

Devuelve Países Por Fecha Solicitados	**002**	
	Prueba de funcionamiento	**Obligatoria**
Descripción: Se realizarán diez peticiones para 100 países distintos de diez en diez y para diferentes rangos de fechas. Posteriormente se comprobará que en cada una de las respuestas los datos pertenezcan a los países solicitados y estén dentro del rango de fechas especificado.		
Prerrequisitos: Haber instalado todas las dependencias. Poner en marcha el componente DataAcccess_API de la aplicación. Haber esperado el tiempo suficiente para que los datos hayan sido insertados en base de datos.		
Pasos:	**Datos**:	
Crear Petición introduciendo una lista de códigos ISO2 de país en el campo correspondiente del modelo. Es necesario introducir, también, el nombre del método correspondiente al que se dirige la aplicación.	Modelo POST	
Enviar Petición	Modelo POST	
Recibir Respuesta	Modelo de salida de datos	
Comprobar Respuesta	Modelo de salida de datos	
Repetir Proceso		

Devuelve datos para todos los países disponibles dentro de un rango de fechas	**003**	
	Prueba de funcionamiento	**Obligatoria**
Descripción: Se almacenarán todos los países disponibles en base de datos y, posteriormente se enviarán 100 peticiones que solicitarán datos para todos los países dentro de un rango de fechas. Se comprobará que existen datos para todos los países y que esos datos estarán dentro del período solicitado.		
Prerrequisitos: Haber instalado todas las dependencias. Poner en marcha el componente DataAcccess_API de la aplicación. Haber esperado el tiempo suficiente para que los datos hayan sido insertados en base de datos.		
Pasos:	**Datos**:	
Solicitar listado de países introduciendo el nombre del método adecuado dentro del campo apropiado de la petición POST.	Listado de países	
Crear Petición	Modelo POST	
Enviar Petición	Modelo POST	
Recibir Respuesta	Modelo de salida de datos	
Comprobar Respuesta	Modelo de salida de datos	
Repetir Proceso		

Devuelve Listado de países	**004**	
	Prueba de funcionamiento	**Obligatoria**
Descripción: Se realizarán 100 peticiones del listado completo de países.		
Prerrequisitos: Haber instalado todas las dependencias. Poner en marcha el componente DataAcccess_API de la aplicación. Haber esperado el tiempo suficiente para que los datos hayan sido insertados en base de datos.		
Pasos:	**Datos**:	
Crear Petición	Modelo POST	
Enviar Petición	Modelo POST	
Recibir Respuesta	Modelo de salida de datos	
Comprobar Respuesta	Modelo de salida de datos	
Repetir Proceso		

Devuelve Listado de fechas disponibles	**005**	
	Prueba de funcionamiento	**Obligatoria**
Descripción: Se realizarán 100 peticiones del listado completo de fechas. Se comprobarán su coincidencia a contar desde el primer día de registro de las mismas hasta el día de realización de la prueba.		
Prerrequisitos: Tener en pleno funcionamiento la aplicación.		
Pasos:	**Datos**:	
Crear Petición	Modelo POST	
Enviar Petición	Modelo POST	
Recibir Respuesta	Modelo de salida de datos	
Comprobar Respuesta	Modelo de salida de datos	
Repetir Proceso		

1.3.4.3 Pruebas de Conjunto

Repetir pruebas realizadas en el componente de Seguridad	**001**	
	Prueba de funcionamiento	**Obligatoria**
Descripción: Se repetirán las pruebas realizadas al componente de seguridad, esta vez, empleando la url del componente Covid_API que gestiona el conjunto.		
Prerrequisitos: Haber instalado todas las dependencias. Poner en marcha todos los componentes de la aplicación. Haber cumplido los prerrequisitos de cada uno de ellos.		
Pasos:	**Datos**:	
Levantar todos los componentes	Modelo POST	
Repetir Pruebas para cada componente usando el modelo de datos propio de Covid_API.	Modelo POST	

Recibir Token JWT tras autentificar un usuario registrado	**002**	
	Prueba de funcionamiento	**Obligatoria**
Descripción: Se probará a identificar un conjunto de usuarios previamente registrados y cada vez que la identificación sea exitosa se recibirá un JWT token como respuesta. En caso negativo se mostrará un error HTTP 401.		
Prerrequisitos: Haber instalado todas las dependencias. Poner en marcha todos los componentes de la aplicación. Haber cumplido los prerrequisitos de cada uno de ellos. Poseer varios usuarios registrados.		
Pasos:	**Datos**:	
Enviar petición de autorización a la url correspondiente.	Modelo POST de Usuario.	
Comprobar que la respuesta recibida posibilita la petición de datos al introducir el JWT token en el correspondiente apartado de cabecera.	Modelo POST de Usuario y de CovidData.	
Repetir todo el proceso para cada uno de los usuarios.		

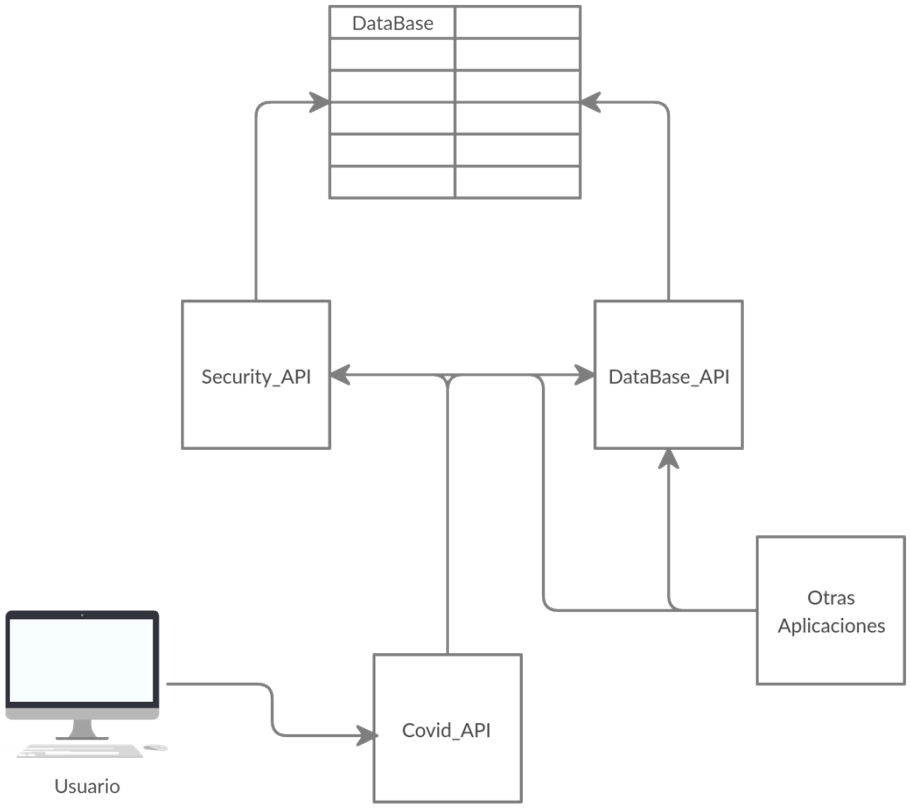
Repetir pruebas de acceso a datos	**003**	
	Prueba de funcionamiento	**Obligatoria**
Descripción: Se repetirán todas las pruebas de acceso a datos esta vez empleando el modelo y url propios del componente Covid_API encargado de interactuar con el usuario y gestionar el conjunto.		
Prerrequisitos: Haber instalado todas las dependencias. Poner en marcha todos los componentes de la aplicación. Haber cumplido los prerrequisitos de cada uno de ellos. Haber obtenido un token de acceso tras haber identificado un usuario.		
Pasos:	**Datos**:	
Enviar petición de autorización a la url correspondiente.	Modelo POST de Usuario.	
Obtener token de autentificación JWT.	Modelo POST de Usuario y de CovidData.	
Repetir todo el proceso de pruebas del componente de acceso a datos.		

2. Arquitectura

Como ya se ha mencionado, se ha seguido una arquitectura orientada a microservicios. Ésta consiste en una perspectiva para el desarrollo de aplicaciones informáticas mediante la cual son construidas uniendo un conjunto de pequeños servicios que se ejecutan de manera autónoma dentro de su propio proceso. Normalmente se comunican entre sí a través de una API con recursos HTTP. Puede encontrarse más información desde esta url: https://es.wikipedia.org/wiki/Arquitectura_de_microservicios .

Esto quiere decir que cada uno de los servicios que componen la aplicación posee su propia jerarquía de clases y distribución de las mismas por lo que será necesario contemplarlos por separado.

2.1. Diagrama de módulos:



El diagrama anterior muestra una descripción general acerca de como está estructurada la aplicación y cómo se relacionan entre sí sus componentes. En él puede apreciarse una de las ventajas de usar la arquitectura orientada a microservicios: otras aplicaciones pueden emplear los componentes para funcionar.

2.2 Descripción jerárquica:

La aplicación se divide en dos servicios componentes, con capacidad para funcionar independientemente y uno que hace las funciones de orquestador del conjunto ofreciéndole sus funcionalidades al usuario. Se percibe, así, que existe un componente dependiente de otros dos para funcionar a pesar de tener la función de orquestarlos.

Así, el microservicio Security_API puede tratarse como una parte constituyente del sistema con capacidad para funcionar independientemente de las demás. Únicamente se relacionará mediante peticiones HTTP con el componente que ofrece sus funcionalidades al usuario/os . En este caso se trata de Covid_API. Por supuesto contará con su propio acceso a una base de datos a determinar, en última instancia por el usuario aunque, es preferible que ésta sea independiente del resto del sistema pues almacenará información más sensible.

Por otro lado, el componente DataAccess_API posee la responsabilidad de almacenar, gestionar, generar y recuperar todos los datos relativos a la pandemia de Covid-19. Únicamente dependerá de una conexión a base de datos pues el despliegue de tablas, realización de consultas, actualizaciones y recogida de los datos es responsabilidad suya y por lo tanto se encuentra automatizado. Se comunicará con el componente que actuará ofreciendo sus servicios al usuario o usuarios Covid_API.

Finalmente el orquestador: Covid_API. Sus funciones son las de actuar como referente de comunicación con el usuario/os ofreciendo las funcionalidades de los dos componentes anteriores. Además, añadirá la responsabilidad de gestión de la propagación de identidad generando los JWT tokens una vez autentificado el usuario correspondiente. También es responsabilidad suya no permitir el acceso a los datos a usuarios no registrados.

2.3 Diagrama de módulos:

2.3.1 DataAccess_API

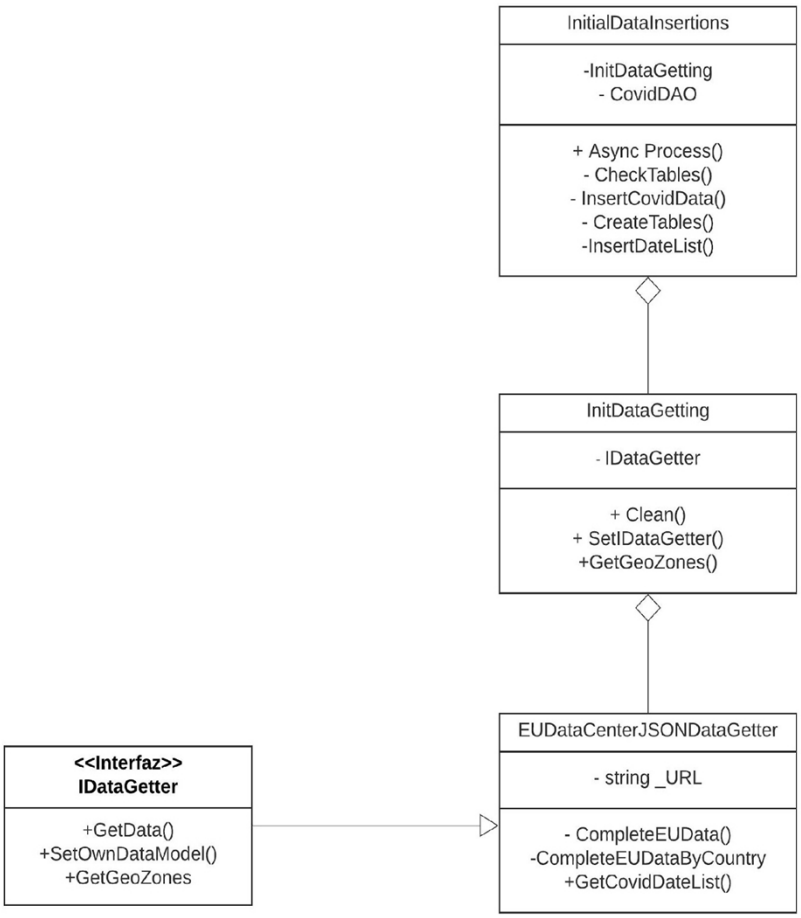
2.3.1.1 DataInsertions

Se trata de un módulo diseñado para gestionar la manipulación y gestión de datos desde el exterior de la aplicación. Sus responsabilidades son:

El diseño se ha realizado para que sea ampliable en un futuro dependiendo de las diversas fuentes empleadas para obtener los datos. En este caso el centro de datos Europeo ha sido el tomado como base por lo que es la misma clase de objeto la que se encarga de obtener los datos de su proveedor y de completar éstos con los diferentes servicios existentes a nivel global pero, al basarse en una interfaz, en cualquier momento puede generarse una clase de obtenedor de datos distinta que la implemente y componer con ella los niveles superiores que, al fin y al cabo, serán los que se sirvan de ella.

Hay que mencionar que todos los procesos producidos dentro de este módulo se ejecutarán indefinidamente dentro de un hilo distinto al del programa principal. Este hilo será lanzado al poner en marcha la aplicación y quedará en funcionamiento indefinidamente mientras la misma se encuentre actuando. Una vez cumplidas sus funciones quedará en suspenso durante veinticuatro horas, tras las cuales volverá a inicializarse para actualizar la información diaria.

El diagrama de clases es el siguiente:



- Interfaz IDataGetter:

Define los métodos que debe de implementar cualquier clase de objeto encargada de recolectar los datos desde un servidor externo. Son tres los pasos a realizar:

1. Obtener los datos en el formato externo.

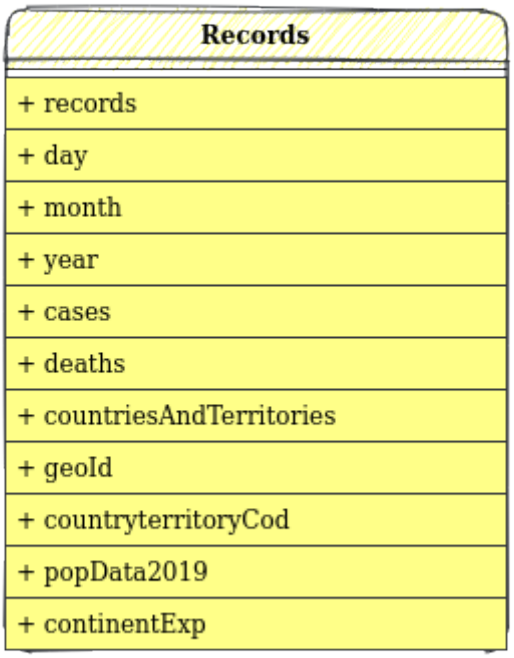
2. Convertirlos al formato interno.

3. Servírselos al resto de clases de la aplicación.

- EUDataCenterJSONDataGetter:

Se trata de la clase encargada de implementar la recogida de datos desde un servidor propio de la Unión Europea. El atributo principal es, precisamente, la url desde la que este servicio se encuentra disponible. Incluye tres métodos adicionales a los de la interfaz. Han sido generados para completar la información no disponible entrecruzándola con la de otros servicios disponibles. Además, sirve una lista de fechas disponibles para consulta.

El modelo de datos propio del centro de datos de la UE:



El modelo de datos con el que es completado el anterior:

- InitDataGetting:

Esta clase, compuesta con una interfaz IDataGetter que será su único atributo, tiene como funciones servirle los datos obtenidos por la interfaz al resto de la aplicación. Además, será la encargada de liberar la memoria del ordenador una vez esos datos hayan sido insertados en base de datos y, tiene un método que puede variar qué clase de obtenedor de datos será instanciado en su interior como atributo dependiendo el nombre del proveedor de los mismos.

- InitialDataInsertions:

Es el consumidor preferente de los servicios presentados anteriormente y única clase accesible desde el exterior. Su proceso es asíncrono, por lo que se lanza dentro de un hilo independiente del que sigue resto de la aplicación. Se va a encargar de comprobar la existencia

de las tablas necesarias en base de datos creándolas en caso de que no hayan sido generadas con anterioridad.

Gestionará también la recogida de los mismos manejando la clase InitDataGetting y efectuando la inserción en base de datos con la ayuda de la clase DAO correspondiente que será descrita a continuación. Una vez completado su proceso principal – comprobación de tablas, generación de las mismas si corresponde, recolección de datos, persistencia – pasará a un segundo plano hasta cumplidas veinticuatro horas, tras las cuales volverá a iniciarlo manteniendo los datos de la aplicación siempre actualizados.

2.3.1.2 CovidDAO

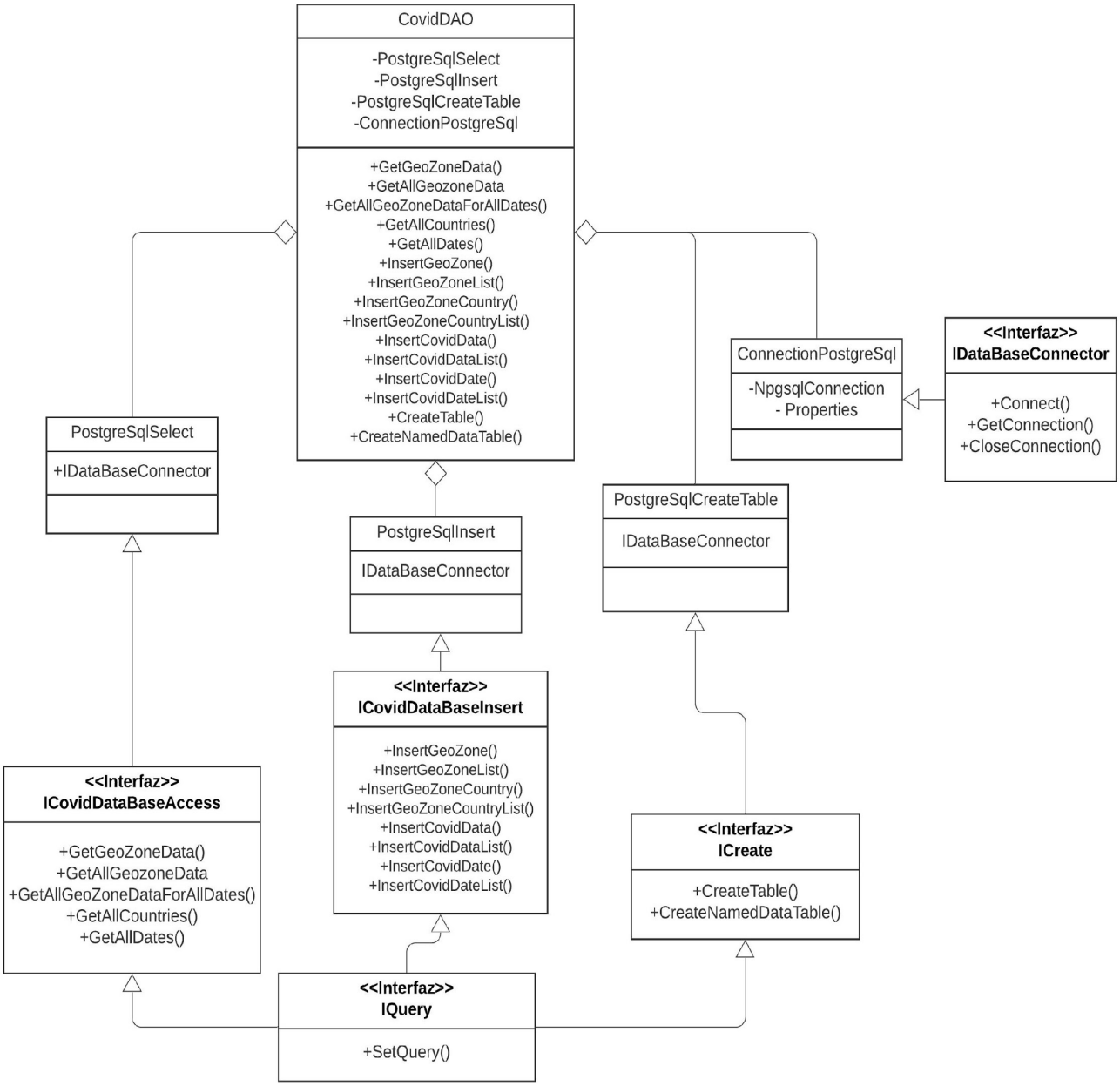
<<Data Access Object>> por sus siglas en inglés. Se trata de un objeto cuya responsabilidad es gestionar el acceso a los datos de la aplicación. Como clase abstracta facilitará el cambio de proveedor y medio de datos dentro de la aplicación pues, en caso de producirse uno o varios cambios simplemente será necesario reimplementar este objeto sin necesidad de modificar el resto de la aplicación.

La de este API se trata de una implementación por composición. La clase abstracta define todos los métodos a emplear que serán adoptados por diferentes interfaces con el objetivo de distinguir entre las operaciones de conexión, creación, inserción y selección de datos.

Cada interfaz resultado del proceso de disgregación la implementará un objeto propio para cada servidor externo de base de datos y operación y serán estos objetos los que compondrán una clase que implementará todos los métodos abstractos de la clase general.

El resultado será una clase mayor con métodos heredados de otra clase abstracta padre que será tratada como esa clase de la que hereda dentro de la aplicación. Así, únicamente sería necesaria la modificación de las instanciaciones de la misma, no su manipulación.

Como se trata de una implementación base no se han usado patrones auxiliares como el de la Factoría que obligaría a generar la clase de objeto esperada dentro de un único gestor por lo que la modificación del servidor de acceso a datos se simplificaría notablemente. No obstante, el requisito indispensable mínimo para que su implementación tuviese sentido no se cumple ya que únicamente se ha contemplado un proveedor de acceso a datos.



- IDataBaseConnector:

Es una interfaz que obliga a la implementación de las funcionalidades relativas a la conexión con base de datos: la conexión, la obtención del objeto que gestiona esta por parte del que lo va a utilizar y el cierre de la conexión.

- ConnectionPostgreSql:

Se trata de la implementación de la interfaz de conexión para la base de datos de PostgreSQL. Será uno de los atributos de una clase más general y sus funcionalidades servirán a otras clases para cumplir con sus correspondientes responsabilidades.

- IQuery:

Se trata de una interfaz cuya única funcionalidad consiste en la generación de un objeto consulta empleable por cada uno de sus implementadores. Garantiza que cada implementador sepa generar sus propias consultas dependiendo de los datos recibidos del exterior únicamente en aquellos campos preseleccionados.

- ICreate:

Cualquier clase de objeto encargado de gestionar la creación de tablas deberá de implementar esta interfaz que obligará a la generación de dos funcionalidades: la creación de tablas y la creación de tablas cuyos nombres han sido pasados por parámetro con lo que necesita de la implementación de la interfaz <<IQuery>>.

- PostgresqlCreateTable:

Implementa la interfaz anterior. Necesita de una conexión a base de datos PostgreSQL para funcionar. Va a componer una clase mayor que delegará en ésta las operaciones de creación de tablas por lo que debe de ser autosuficiente a la hora de generar sus propias consultas y componerlas en base a la mínima cantidad de datos recibidos desde el exterior. Para ello dispondrá de su propia ruta hacia las consultas previamente establecidas y almacenadas en formato JSON dentro de su carpeta CreateTablesQueries.

- ICovidDataBaseInsert:

Se trata de una interfaz que compromete a implementar las operaciones necesarias de inserción en base de datos para garantizar la persistencia de la información de la aplicación. La inserción de uno o varios datos referentes a una zona concreta, la inserción de uno o varios países disponibles para consulta así como la inserción de una o varias fechas para la o las que existe/en datos.

- PostgreSQLInsert:

Implementa la interfaz anterior para la base de datos PostgreSQL. Requiere, para su funcionamiento, de un atributo conexión que gestione la misma con base de datos. Es capaz de, en base a los datos de entrada generar sus propias consultas a partir del uso de unas plantillas a las que tiene acceso y han sido preparadas, en formato JSON para su uso exclusivo.

- ICovidDataBaseAccess:

Declara y obliga a la implementación de las operaciones de consulta y recuperación de información desde base de datos. Estas operaciones pueden ser para datos de un país concreto dentro de un rango de fechas, para recuperar los datos de varios países dentro de un rango de fechas, para recuperar la información de todos los países para todas las fechas disponibles, para

la recuperación de una lista de países de los que se dispone información o para la recuperación de una lista de fechas para las que existen datos disponibles.

- PostgreSqlSelect:

Implementa la interfaz de acceso a datos para la base de datos PostgreSQL. Requiere de una conexión con la misma como medio de transmisión de consultas. Dichas consultas, como ocurre con el resto de clases de este módulo son accesibles únicamente para esta clase que las cargará desde un archivo JSON.

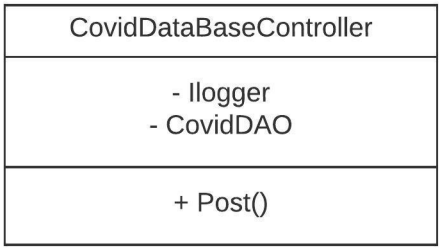
- CovidDAO:

Se trata de una clase abstracta que define todas las operaciones necesarias para comunicarse con la base de datos. A lo largo de toda la aplicación se usará una referencia a ella para ejecutar cualquier consulta que sea requerida por el flujo mismo de ella. Existe una implementación llamada PostgreSqlCovidDAO que está compuesta por cada una de las interfaces operacionales definidas anteriormente. Para cada uno de sus métodos delegará la operación a una de sus clases componentes que será la responsable de la misma.

2.3.1.3 DataAccessControler

Se trata del controlador encargado de recibir las peticiones HTTP destinadas a la aplicación, redirigir la información de entrada hacia el lugar correcto y elaborar la respuesta que los usuarios recibirán.

Diagrama de clase:

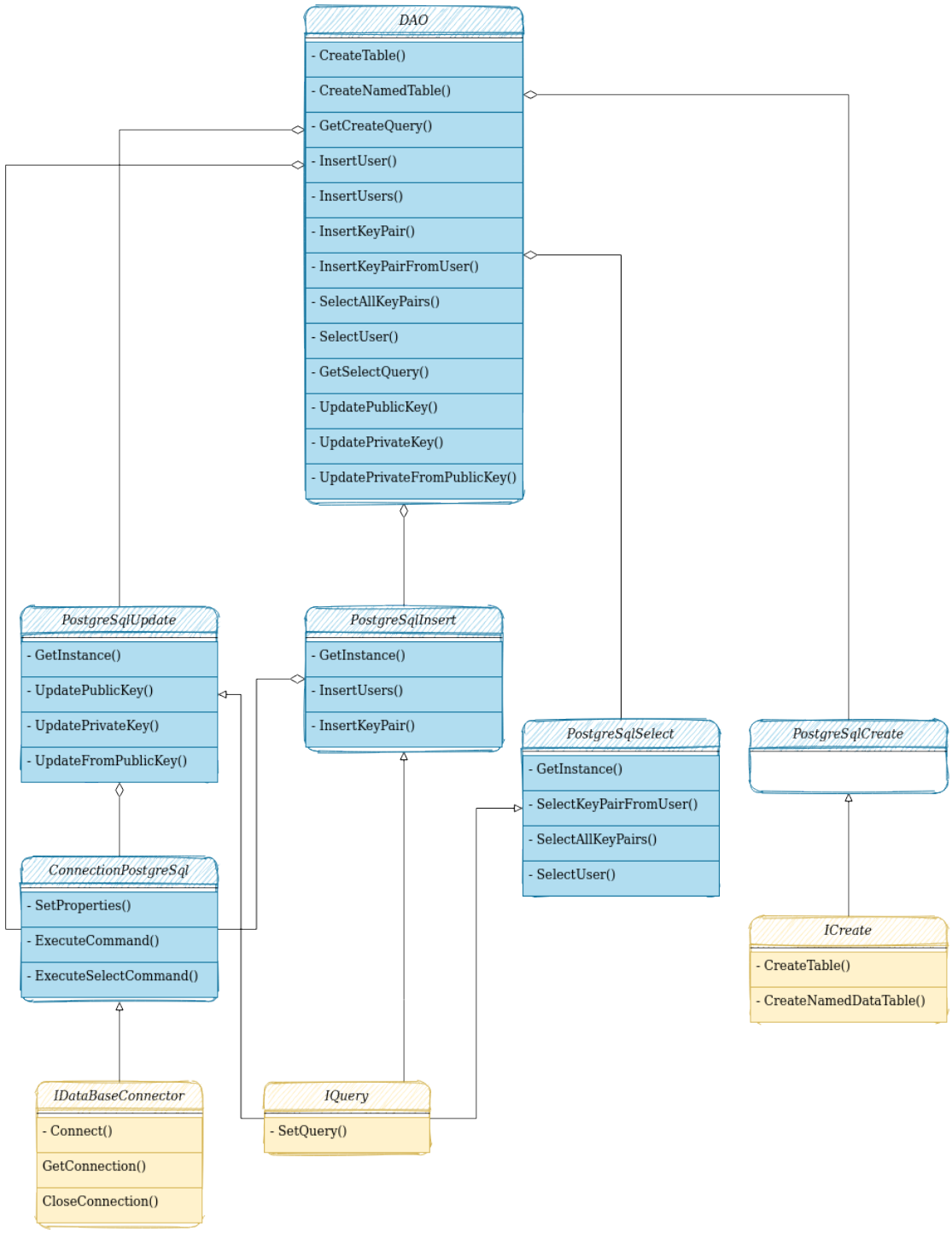


Únicamente dispone de dos atributos, una interfaz gestora de logs y archivos de histórico de operaciones y un CovidDAO que será empleado para acceder a los datos insertados en base de datos.

2.3.2 Security API

2.3.2.1 Security DAO

Al igual que en la anterior parte componente del sistema para la gestión de todo cuanto se encuentre relacionado con la base de datos – creación de tablas, inserción, actualización, borrado y selección de datos de las mismas – es gestionado por un objeto DAO o <<Data Access Object>>. Siguiendo la misma línea este objeto ha sido compuesto por un grupo de clases en cada una de las cuales se delega una responsabilidad. El objetivo es alcanzar la flexibilidad en la gestión de los datos por parte de la aplicación.


DAO
- CreateTable()
- CreateNamedTable()
- GetCreateQuery()
- InsertUser()
- InsertUsers()
- InsertKeyPair()
- InsertKeyPairFromUser()
- SelectAllKeyPairs()
- SelectUser()
- GetSelectQuery()
- UpdatePublicKey()
- UpdatePrivateKey()
- UpdatePrivateFromPublicKey()
PostgreSqlUpdate
- GetInstance()
- UpdatePublicKey()
- UpdatePrivateKey()
- UpdateFromPublicKey()
PostgreSqlInsert
- GetInstance()
- InsertUsers()
- InsertKeyPair()
PostgreSqlSelect
- GetInstance()
- SelectKeyPairFromUser()
- SelectAllKeyPairs()
- SelectUser()
PostgreSqlCreate
ConnectionPostgreSql
- SetProperties()
- ExecuteCommand()
- ExecuteSelectCommand()
ICreate
- CreateTable()
- CreateNamedDataTable()
IDataBaseConnector
- Connect()
GetConnection()
CloseConnection()
IQuery
- SetQuery()

- Interfaz IQuery:

Compromete a la generación de un objeto de consulta propio a la clase que lo implemente.

- Interfaz IDataBaseConnector:

Es una interfaz pensada para definir las operaciones fundamentales de gestión de la conexión a base de datos: conectar, desconectar y obtener el objeto que representa la conexión.

- ConnectionPostgreSQL:

Se trata de la implementación de la interfaz de conexión para la base de datos de PostgreSql. Además de las operaciones relativas a la interfaz es capaz de cargar las propiedades de conexión desde un archivo en formato JSON y de ejecutar más de un tipo de sentencia sql.

- PostgreSQLInsert:

Tiene como responsabilidad la gestión de las operaciones de inserción en una base de datos PostgreSql. Se trata de un objeto del que solamente puede existir una copia dentro de la aplicación y que ofrecerá la posibilidad de insertar nuevos usuarios y nuevos pares de claves privada y pública. Posee como propiedad un objeto ConnectionPostgreSQL para manejar sus consultas y es capaz de establecer las mismas a través de un objeto de consulta que carga desde su propia carpeta contenedora, quedando así restringidas las consultas que realiza.

- PostgreSQLUpdate:

Esta clase maneja las actualizaciones de información de la base de datos. Es un objeto de única copia dentro de la aplicación. Está compuesto por una conexión - ConnectionPostgreSQL – e implementa la interfaz IQuery por lo que es capaz de establecer sus propias consultas en base a la información recibida del exterior y las plantillas en formato JSON de las que dispone. Puede actualizar una clave pública, una clave privada y un conjunto de datos tomando como base una clave pública.

- PostgreSQLInsert:

Es una clase cuya responsabilidad consiste en la recuperación de la información de usuarios y claves privada y pública de una base de datos PostgreSql. En este caso la conexión es recibida desde el exterior al demandar la consulta esperando así una nueva conexión cada vez que se realiza la llamada a uno de sus métodos. Es capaz de precargar una serie de plantillas con las consultas que realizará desde un archivo JSON situado en su misma ubicación, modificándolas para adaptarlas a la demanda de información de cada situación particular. Puede recuperar de base de datos el par de claves pública y privada de un usuario, todas las claves públicas y privadas existentes en base de datos y la información relativa a un usuario.

- Interfaz ICreate:

Define las operaciones que la clase responsable de la creación de tablas ha de llevar a cabo.

- PostgreSQLCreate:

Es la clase responsable de crear las tablas en base de datos empleando sus propias plantillas de creación. Como éste proceso se realiza una única vez ha quedado al margen de los demás y autogestiona completamente la obtención de las tablas a generar desde un archivo propio, los campos de dichas tablas y los nombres de éstas y sus columnas que quedarán lo menos expuestos posible. Recibirá una conexión desde el exterior en el momento de su creación. Sus métodos no serán rehutilizables.

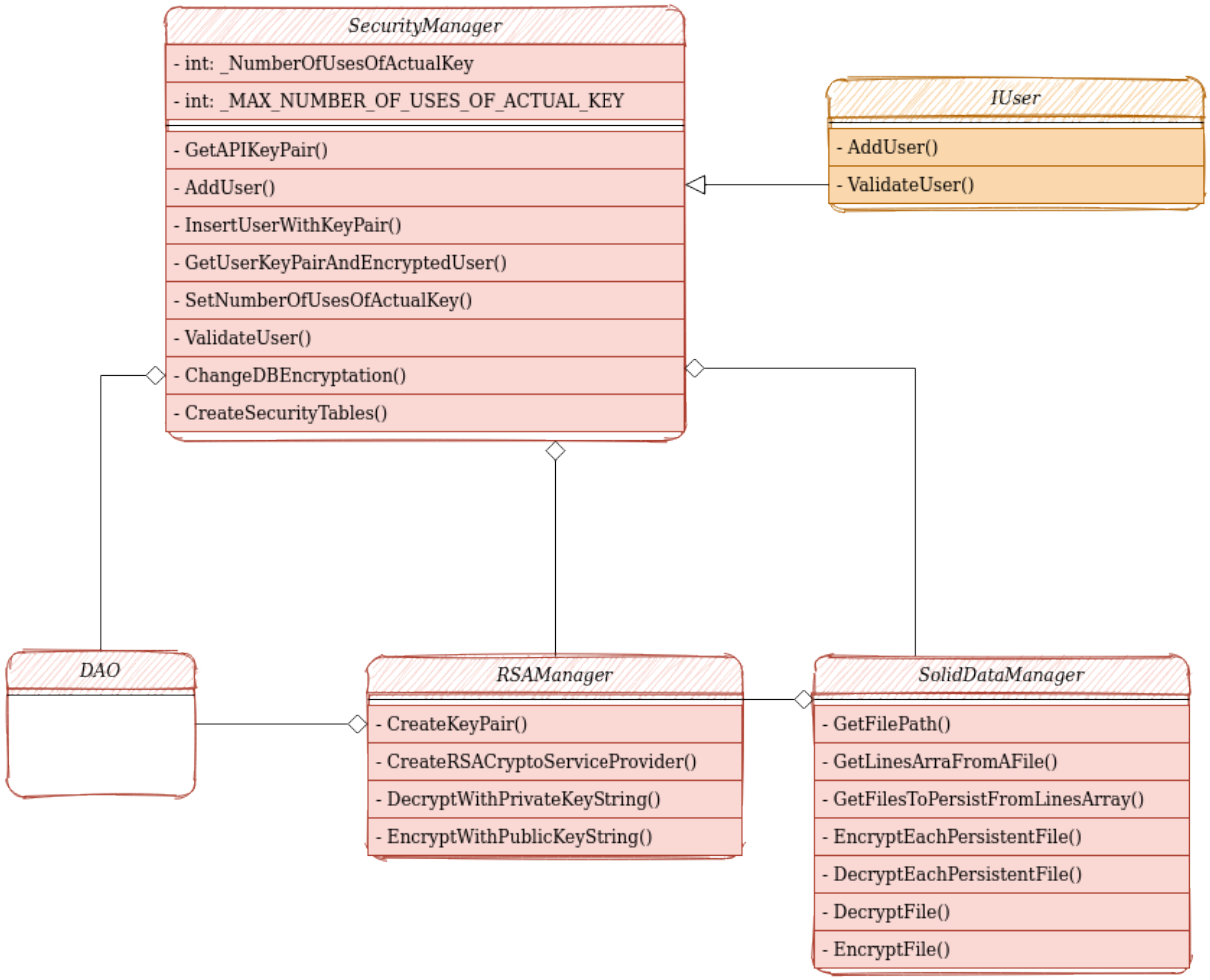
- DAO:

Se trata del objeto que compuesto por todos los anteriores gestiona completamente el acceso a la base de datos de la aplicación. Es una clase abstracta que recibirá una implementación particular para cada gestor de base de datos que decida emplearse, en este caso PostgreSql. Se trata de una abstracción del proceso de gestión de base de datos por parte de la aplicación garantizando que exista cierta flexibilidad a la hora de modificar el proveedor de los servicios de persistencia.

Sus operaciones coinciden con los métodos de sus clases componentes añadiendo la posibilidad de obtener los objetos de consulta para las operaciones de selección y de creación de tablas y posibilitar la generación de las mismas dentro de un ambiente de campos y propiedades encriptadas.

2.3.2.2 Security Manager

Se trata de una clase que, compuesta por otras tres, gestiona todo el aspecto de la seguridad en el acceso, creación, actualización y validación de datos de usuario así como de la misma dentro de la aplicación. Está pensada para emplear un conjunto de claves pública y privadas que encripten toda la información que la aplicación necesita para su funcionamiento y gestionar la renovación de dicha encriptación así como de desencadenar los procesos de generación de tablas y validación y creación de usuarios.



- RSAManager:

Es un gestor de claves pública y privada y de su empleo. Las crea en caso necesario y encripta a través de una clave pública y desencripta a través de una clave privada. Se vale de un objeto DAO para la gestión, persistencia y el uso de las claves pública y privada internas de la aplicación empleadas como medio de seguridad interna.

- SolidDataManager:

Gestiona la manipulación y lectura de archivos en disco sólido o disco duro valiéndose de un RSAManager para realizar las encriptaciones y desencriptaciones de dichos archivos garantizando la hermeticidad de la información que pueda considerarse sensible dentro de la aplicación. Las plantillas de consulta, el nombre de las tablas empleadas o las propiedades de conexión a base de datos son ejemplos de archivo sensible.

- Interfaz IUser:

Define las operaciones de generación y validación de usuario que serán empleadas por la clase responsable de las mismas.

- SecurityManager:

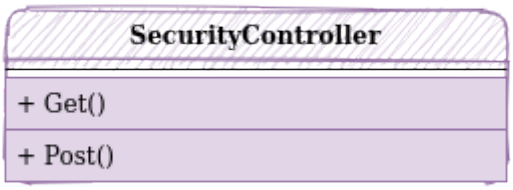
Compuesta por las anteriores y un DAO gestiona todas las operaciones de seguridad de la aplicación, desde la comprobación de la existencia de las tablas y su generación hasta la creación y validación de los usuarios delegando determinadas operaciones en sus clases componentes.

Realiza varios procesos. Una de sus propiedades lleva la cuenta del número de veces que las claves privadas y públicas de uso interno de la aplicación han sido empleadas persistiendo en disco duro esta misma propiedad que se carga desde el mismo al inicializarse la clase. Una vez

alcanzado el número máximo desencadena el proceso de cambio de claves y de encriptación de toda la base de datos hacia las nuevas y su sustitución en el registro propio de la aplicación. También inserta nuevos usuarios tras encriptar sus datos y posteriormente los valida y recupera de base de datos las claves pública y privada que serán empleadas para la creación de nuevos usuarios.

2.3.2.3 Security Controller

Se trata del controlador generado para gestionar las peticiones HTTP que recibirá la aplicación desde el exterior. El método Get suministra una clave pública con la que encriptar los datos sensibles de los nuevos usuarios y el método Post que recibe los usuarios que van a ser dados de alta o deben de validarse.

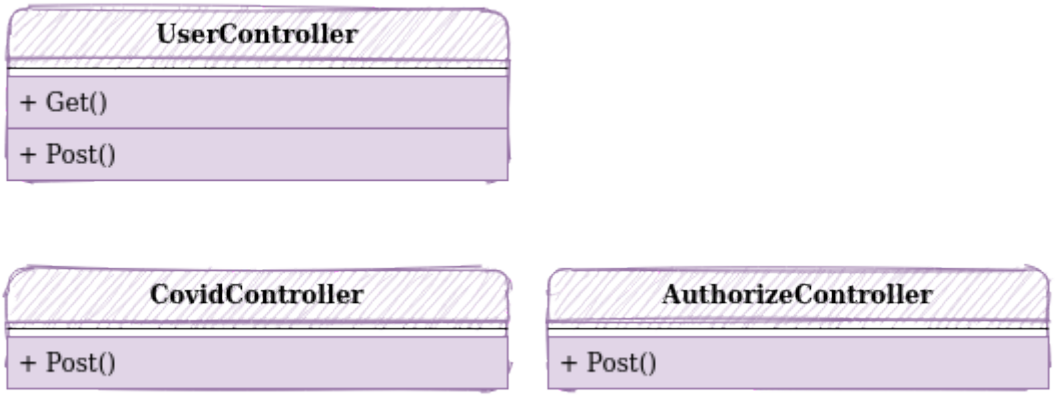


2.3.3 Covid API

Es un módulo diseñado únicamente para realizar de interfaz con el usuario. Por medio del mismo éste último puede interactuar con los dos módulos ya descritos. Solamente añade un servicio extra de tokenización mediante el cual el usuario obtiene un BearerToken válido durante 24h con el que evitar sucesivas autenticaciones. El resto de servicios son un simple redireccionamiento hacia los módulos que componen la aplicación. Ofrece además la implementación de una memoria caché que favorece la rápida obtención de los datos en peticiones de peso.

2.3.3.1 Controladores

Se encargan de recibir las peticiones POST o GET y redireccionarlas al componente correspondiente.



- UserController:

Redirecciona hacia el controlador de usuarios del módulo Security API.

- AuthorizeController:

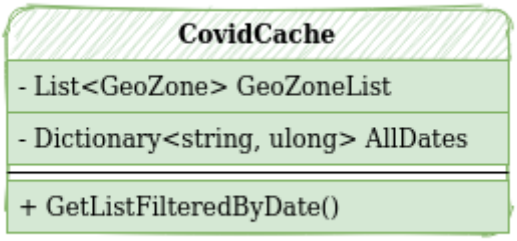
Tokeniza la autentificación empleando para ello el servicio del mismo nombre del módulo Security API.

- CovidController:

Ofrece los datos relativos a la pandemia de Covid-19 empleando la memoria caché o el redireccionamiento hacia el microservicio DataAccess API.

2.3.3.2 CovidCache

Extrae el conjunto de los datos disponibles en base de datos y los ofrece de manera rápida filtrando por rango de fecha desde memoria RAM.

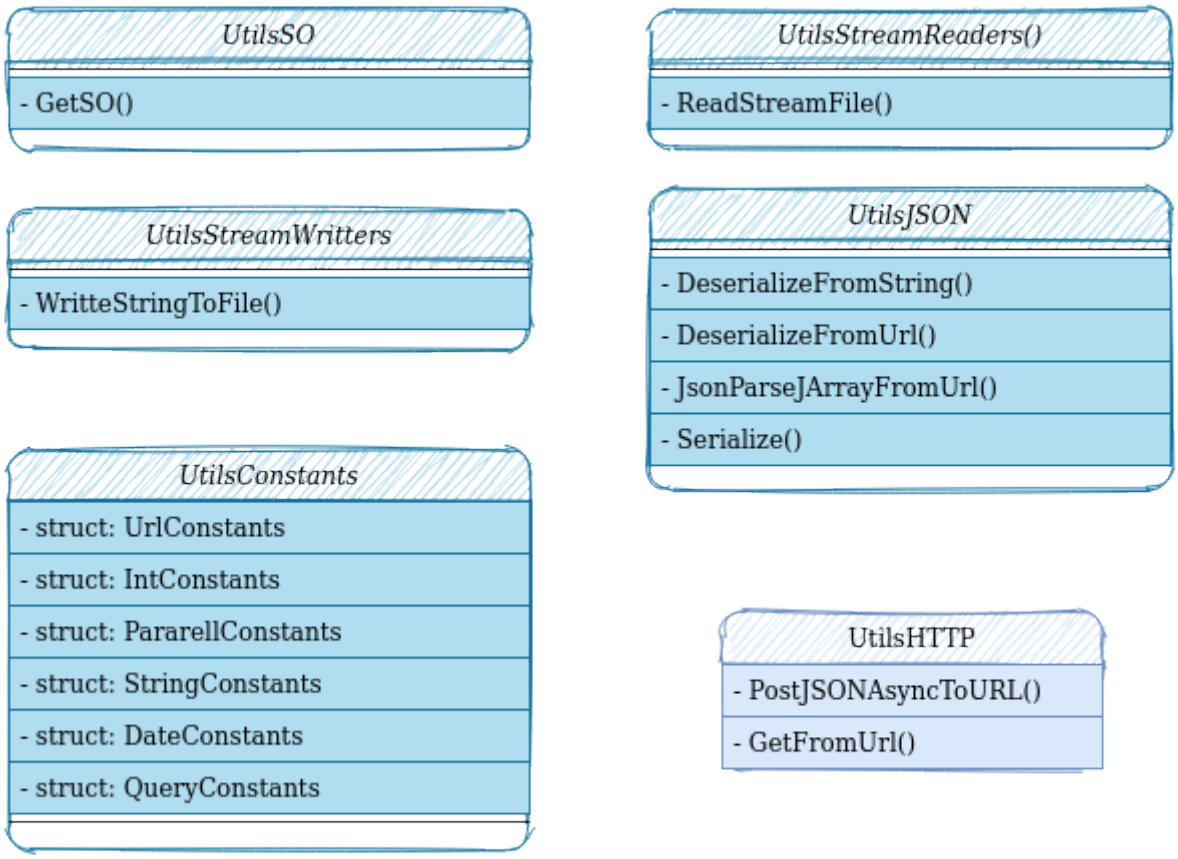


2.3.3.3 TokenGenerator

Cumple las funciones relativas a la generación de tokens y su almacenamiento.

2.3.4 Clases comunes

Existen una serie de clases comunes a todos los servicios. Éstas incluyen funciones de serializado y deserializado de objetos Json desde urls externas, archivos o cadenas de texto, constantes, operaciones de lectura y escritura en disco o reconocimiento del sistema operativo. Éstas clases son las siguientes:



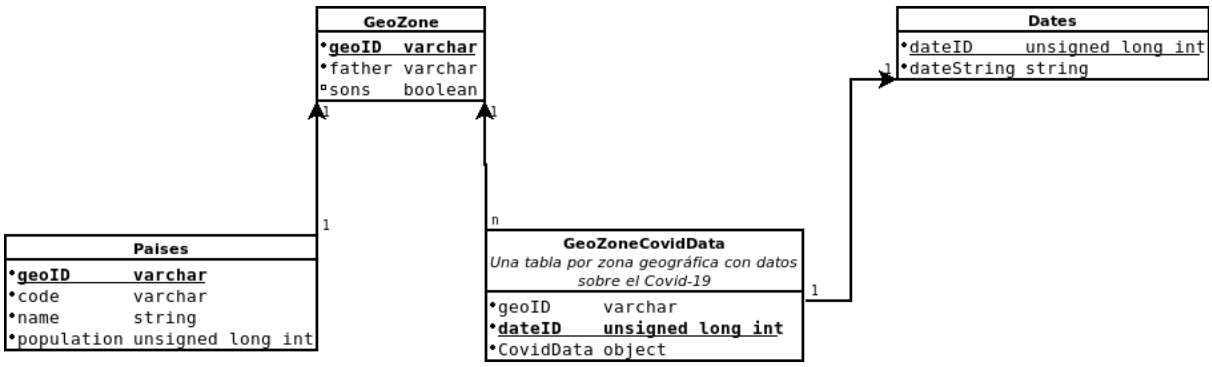
2.3.5 Dependencias Externas

Únicamente se emplea como dependencia externa al propio marco de trabajo de .Net Core una librería llamada Newtonsoft. Ésta es utilizada en la serialización y deserialización de objetos Json por su versatilidad y facilidad. Además, complementa perfectamente la que el marco de trabajo incluye de manera nativa. Su documentación oficial puede encontrarse en:

https://www.newtonsoft.com/json/help/html/Introduction.htm

3 Modelo Entidad Relacional

3.1 Modelo E/R Datos Covid



3.2 Modelo E/R Base de Datos Usuarios