Covid API

# 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. Se levantará un servicio API rest para interactuar con el usuario.**

**2. Las peticiones se atenderán en formato JSON.**

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

**4. 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 enla 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: Winows y Linux.

**- Windows:** Se trata de un sistema operativo para ordenadores desarrollado y distribuido por Micorsoft, 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 archivosd e 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 distitas aplicaciones informáticas, especialmente cuando éstas constan de grandes archivos de código fuente.

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.

### 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/SecureCovidAPI**

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.