Informe de Avance Sprint 3

Datacript Equipo 1 24/10/2023

Integrantes

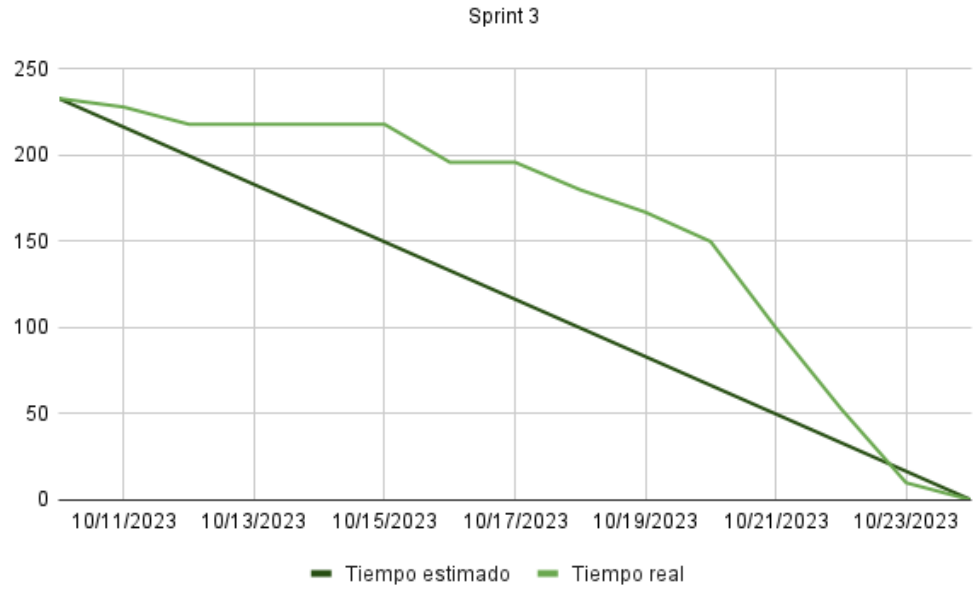
Apellido y Nombre	Rol
Dikenstein, Leandro	Product Owner
Perez, Giannina	Scrum Master
Prieto, Lucas	Líder Técnico
Benitez, Micaela	Developer
Benitez, Yamila	Developer
Clauser, Nahuel	Developer
Torrico, Franco	Developer
Gómez, Federico	Tester

Funcionalidad Comprometida

	Requerimientos prometidos	Completo	Responsable
1	Actualizaciones de SW y Backup de BBDD	SI	Perez, Giannina (SM)
2	Definición de Medidas de Seguridad para APIs	SI	Perez, Giannina (SM)
3	Definición de comunicación entre APIs	SI	Prieto, Lucas (LT)
4	Ingreso de Datos de Prueba	SI	Clauser, Nahuel (Dev)
5	Integración Diagnóstico con BBDD	SI	Torrico, Franco (Dev)
6	Crear usuarios con Roles y Permisos	SI	Prieto, Lucas (LT)
7	Gestión de Admin / Auditor - Carga de informes e historial	SI	Benitez, Yamila (Dev)

8	Gestión de Admin / Auditor - Imágenes y Resultados	SI	Prieto, Lucas (LT)
9	Gestión de Admin - Modificación y Baja de usuario	SI	Clauser, Nahuel (Dev)
10	Gestión de Profesional de la Salud	SI	Benitez, Micaela (Dev)
11	Gestión de Médico	SI	Torrico, Franco (Dev)
12	Alta de establecimientos médicos	SI	Prieto, Lucas (LT)
13	Periodicidad de 60 días para contraseñas	SI	Benitez, Yamila (Dev)
14	Implementación de Cifrado con AES	SI	Torrico, Franco (Dev)
15	Test de API REST	SI	Gómez, Federico (Tester)
16	Prueba de penetración	SI	Gómez, Federico (Tester)
17	Prueba de integración para conexión con modelos	Si	Prieto, Lucas (LT))
18	Gestión del Proyecto	SI	Perez, Giannina (SM)

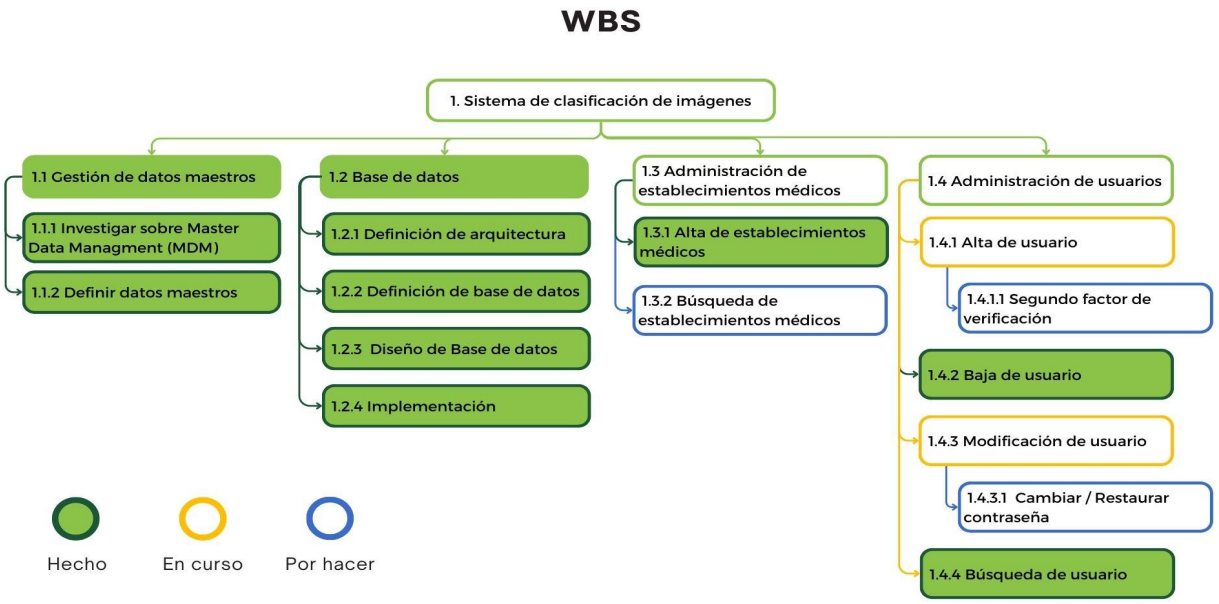
Burndown Chart

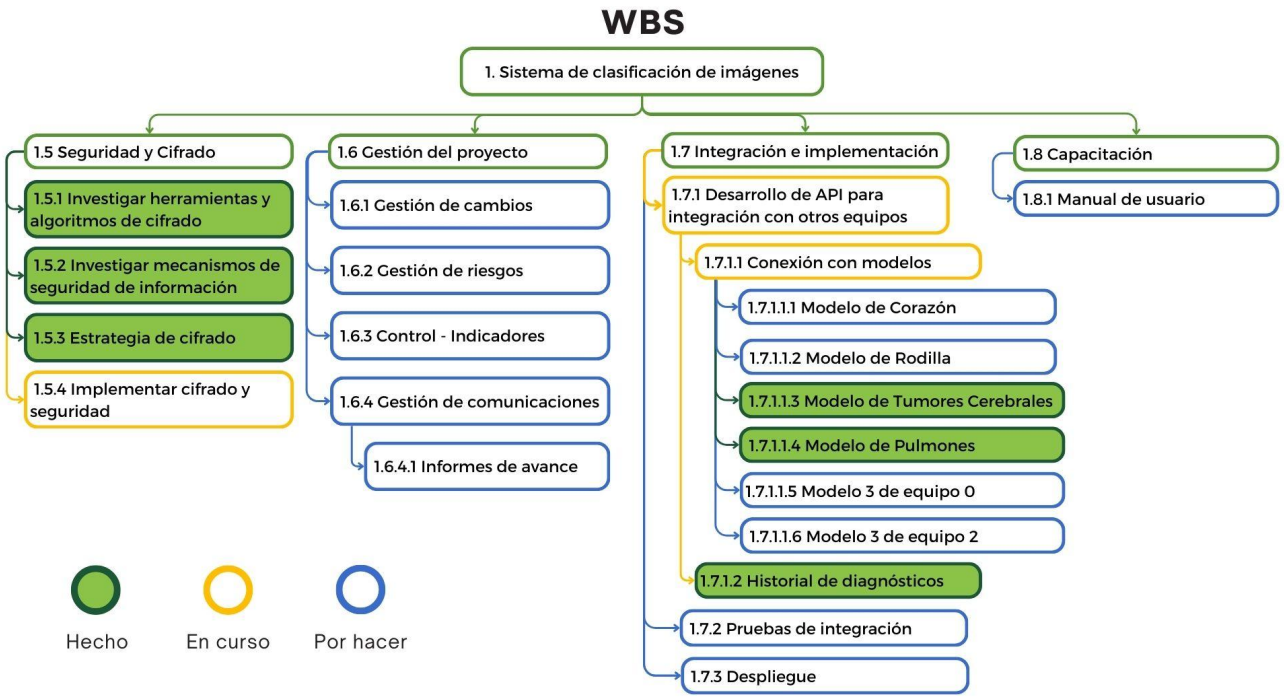


WBS

Durante este sprint realizamos una actualización de la WBS dando un poco más de detalle a las tareas de administración de usuarios y desarrollo de la api. Las partes de la wbs que tuvimos trabajando durante este sprint fueron (en verde las terminadas y en amarillo las que siguen en desarrollo):

- **1.2.4** Base de datos - Implementación. Terminamos de implementar la base de datos según lo diagramado en el DER y de esta manera empezamos a hacer pruebas desde la api.
- **1.4.1** Alta de usuario
- **1.4.2** Baja de usuario
- **1.4.3** Modificación de usuario
- **1.4.4** Búsqueda de usuario
- **1.5.4** Implementación de cifrado y seguridad
- **1.7.1.1.3** Conexión con modelos - Cerebro
- **1.7.1.1.3** Conexión con modelos - Pulmones
- **1.7.1.2** Historial de diagnosticos





Actividades realizadas

1. **Actualizaciones de SW**

   Otorgar actualizaciones frecuentes a nuestro sistema asegura el correcto funcionamiento del mismo, además de mantener seguros nuestros datos. Por esta razón, hemos optado por implementar actualizaciones de versión de PostgreSQL al final de cada Sprint restante.

   Un riesgo detectado en esta instancia es que ocurra una excepción al momento de la actualización, debido a la incompatibilidad de la versión de la base de datos antigua con la versión nueva. Con el objetivo de mitigar este riesgo, tendremos en cuenta el siguiente paso previo: Instalaremos un paquete que contendrá la versión anterior de postgres. Luego de esto, se moverá la base de datos a un nuevo directorio y se inicializará la base de datos con la nueva versión. Una vez hecho esto, migraremos la información de la versión antigua a la nueva versión mediante la herramienta "pg_upgrade". Finalmente, ya se podrá iniciar la nueva base de datos.

   Con el fin de definir una **planificación de actualizaciones a futuro**, pues primero se debe implementar el sistema en un entorno profesional para manejar datos reales, tendremos en cuenta la siguiente gestión:

   - **Limpieza de datos:** Se deberán borrar los perfiles de acceso de los médicos o profesionales de la salud que ya no pertenezcan a una entidad médica.
   - **Gestión de duplicados**: Implementaremos herramientas que detectan de forma automática posibles duplicados, para proceder a unificarlos en un único perfil con la información más actualizada.

Backup de Base de Datos

La herramienta pgAdmin nos permite gestionar de una manera simple los Backups, generando un archivo de respaldo de la base de datos con las opciones de configuración que apliquemos. Por esta razón, pgAdmin es donde llevaremos a cabo este proceso.

Por otro lado, para esta instancia estaremos realizando un Restore de prueba con la misma herramienta, con la cual crearemos una nueva base de datos y llevaremos a cabo el proceso de restauración a partir del archivo de backup generado anteriormente.

Estos proceso se pueden llevar a cabo mediante la interfaz de pgAdmin o mediante los siguientes comandos:

- **Backup:** pg_dump -U usuario -W -h host basename > basename.sql
- **Restore:** psql -U username -W -h host basename < basename.sql

En donde:

- U se refiere al usuario propietario de la base de datos o el usuario postgres.
- W para solicitar el password del usuario antes especificado.
- H para indicar cuál es el servidor PostgreSQL
  - local: localhost
  - remota: IP del servidor PostgreSQL

El riesgo que hemos detectado en esta actividad es que si el backup no está actualizado, al momento de hacer un restore se pueden perder datos importantes. Para mitigar este riesgo, se realizará un backup semanal.

2. Definición de Medidas de Seguridad para APIs

Decidimos utilizar la herramienta JSON Web Token (JWT) como medida de seguridad para nuestra API. JWT, además de ser compatible con PostgreSQL, ofrece una forma segura y simple de transmitir datos entre aquellos usuarios que tengan autorización para acceder a nuestra API.

Durante nuestra investigación, nos encontramos con que un JWT consta de tres partes:

- **Header:** Encabezado dónde se indica, al menos, el algoritmo y el tipo de token.
- **Payload:** Donde aparecen los datos de usuario y privilegios, así como toda la información que queramos añadir, todos los datos que creamos convenientes.
- **Signature:** Una firma que nos permite verificar si el token es válido.

1. Petición POST para enviar el usuario y contraseña, y realizar el login.
2. Se comprueba que el usuario y contraseña son correctos, y de serlos, el servidor genera un token JWT para devolverlo al usuario, quien lo almacena de forma segura, por lo general en las cookies o en el almacenamiento local del navegador.
3. Se devuelve el JWT.
4. En las solicitudes subsiguientes a la API, el cliente incluye este token en el encabezado de la solicitud. Authorization: Bearer XXXXXXX, siendo Bearer el tipo de prefijo seguido de todo el contenido del token.
5. El servidor verifica la firma del token para asegurarse de su autenticidad y confiabilidad en el usuario. Además, el token puede contener información adicional sobre el usuario, como roles y permisos, lo que facilita la autorización de recursos específicos.
6. Se autoriza la solicitud si la firma es válida y si le corresponde a ese usuario, el servidor responde con el recurso protegido.

3. Definición de Comunicación entre APIs

Durante este periodo estuvimos en contacto con los demás equipos para definir cómo se realizará la comunicación entre los servicios de cada equipo. Principalmente los modelos de cerebro y pulmones ya están funcionales.
Para este caso la comunicación para las predicciones comenzará cuando desde el front se realice una nueva consulta de predicción, la cual nos llegará a nosotros a través de los métodos POST **predecir/cerebro** o **predecir/pulmones** de nuestra API. Recibiremos la imagen con los datos necesarios para el funcionamiento del modelo, junto con el id del usuario que realizó la consulta y el id del médico que se le cargará el resultado a su nombre. Desde nuestra API llamaremos a la API del modelo con la imagen para esperar su resultado, para luego guardar todos estos datos en la base de datos en una tabla “diagnosticos” y enviar la respuesta al front en el caso de necesitar mostrarla.
En general, la comunicación entre servicios se realizará de la misma manera, recibiendo la petición del front, comunicándonos con los modelos para recibir su predicción/respuesta, guardar los datos y enviar la respuesta al front.
Por lo que actualmente se encuentra funcionando el circuito con el modelo de cerebro y con el modelo de pulmones, y desde nuestro lado ya tenemos el boceto de cómo será la conexión con los demás modelos, a la espera que estos definan qué datos específicos van a necesitar para cada uno y el desarrollo de sus endpoints para efectivizar la conexión.

4. Ingreso de Datos de Prueba

Ingresamos una cantidad considerable de filas en cada tabla de la BBDD con datos inventados, con el objetivo de poder realizar todas las pruebas necesarias de conexión.

5. Integración Diagnóstico con BBDD

Esta actividad fue un primer acercamiento a la tabla diagnóstico, que creamos con el fin de poder realizar pruebas de integración. Esta tabla se modificó luego de recibir los datos que necesitarían los modelos.

Estos fueron los cambios que se aplicaron en los siguientes endpoints:

- **“/Diagnosticos/deleteAll”:** Borraba el JSON provisoriamente.
  Se cambió por **“/Delete/{id}”**, el cual borra en la BBDD el id seleccionado.

Respuestas:

200: "Diagnóstico eliminado correctamente"
500: “No se pudo eliminar el diagnóstico”

- **POST: “/Diagnosticos”:** Sigue cumpliendo la misma función de ingresar diagnóstico, pero en vez de guardar en archivo JSON con respuesta JSON, ahora es insertar datos a la base datos con respuesta JSON.

  Respuestas:

  201: 'Éxito'
  500: 'Error al enviar el diagnóstico'

  Los atributos que se pueden esperar en el envío de datos JSON son:

  - **Usuario ID:** es el usuario al que pertenece el diagnóstico.
  - **Edad:** indicar la edad de la persona.
  - **Peso:** indicar el peso de la persona.
  - **Altura CM:** ingresar la altura registrada que tuvo la persona.
  - **Sexo:** indicar el sexo de la persona.
  - **Sección del cuerpo:** ingresar la parte del cuerpo sometida al diagnóstico.
  - **Condiciones Previas:** ingresar una cadena de texto que describa la información relevante del diagnóstico médico.
  - **Imagen:** en este caso imagen ingresar la palabra imagen, ya que esperamos trabajar con imágenes los próximos pasos.

- **GET “/Diagnosticos/{id_diagnostico}”** proporciona un JSON con la información del diagnóstico indicado por ID que haya incluido en la base de datos integrada. Realiza consultas a la base de datos de tabla diagnóstico.

  Respuestas:

  200: 'Éxito'
  204: 'No existe diagnóstico con el id seleccionado'

  Lista de diagnósticos/resultados:

  - **id:** identifica de manera única el diagnóstico
  - **datos_complementarios:** contiene una cadena de texto que describe la información relevante para el modelo de machine learning.
  - **imagen:** por ahora devuelve cadena de bytes
  - **fecha:** fecha en la que se realizó el diagnóstico
  - **resultado:** clasificación que realizó el modelo

- **usuario_id:** rol del usuario
- **usuario_medico_id:** medico que realizo el diagnostico
- **modelo_id:** modelo utilizado para realizar el diagnóstico

- **GET "Diagnosticos/all"** proporciona un JSON con una lista con información de todos los diagnósticos que hayan incluidos en la base de datos integrada, el que cumple el rol de guardado antes de la integración a la base todavía sigue siendo "NULL". no está la tabla en la db

  respuestas: 200: 'Éxito', 204: 'No hay diagnósticos para mostrar'

  - **id:** identifica de manera única el diagnóstico
  - **Usuario ID:** es el usuario al que pertenece el diagnóstico
  - **id rol:** es el rol del usuario

6. **Crear usuarios con Roles y Permisos**

   Se crearon usuarios en la base de datos para todos los miembros del equipo y para la API que accede a esta base. A cada usuario se le asignaron permisos correspondientes a las necesidades que demanden sus desarrollos. Esto con el propósito de tener seguridad en el acceso y modificación de los datos y poder identificar que usuario realizó una consulta en el caso de que suceda algún problema.

7. **Gestión de Admin / Auditor - Carga de informes e historial**

   - **Cargar imagen e informe:**

     Agregamos el endpoint **/predecir/cerebro** que recibe una imagen y datos relevantes para el modelo de clasificación mediante una solicitud POST. Este endpoint se conecta con la API del equipo 0, a la cual le enviamos la imagen y nos devuelve la respuesta del modelo con los distintos tumores de cerebros y la probabilidad de tenerlo.

     Además, se conecta con la base de datos para guardar el diagnóstico una vez obtenido el response de la API del equipo 0.

   - **Consultar Historial:**

     Actualizamos el endpoint **GET /Diagnosticos/historial** que recibe el id de un usuario y su id_rol.

     Si id_rol = 1 (el usuario es auditor) se devuelven todos los diagnósticos que están cargados en la base de datos

     Si id_rol = 4 (el usuario es médico) se devuelven solo los diagnósticos que están cargados a su nombre

     Los atributos que se pueden esperar en la respuesta son:

     - **id:** Identifica de manera única el diagnóstico.
     - **datos_complementarios:** Contiene una cadena de texto que describe la información relevante para el modelo de machine learning.

- **imagen:** Por ahora devuelve cadena de bytes.
- **fecha:** Fecha en la que se realizó el diagnóstico.
- **resultado:** Predicción que realizó el modelo.
- **usuario_id:** Rol del usuario.
- **usuario_medico_id:** Médico que realizó el diagnóstico.
- **modelo_id:** Modelo utilizado para realizar el diagnóstico.

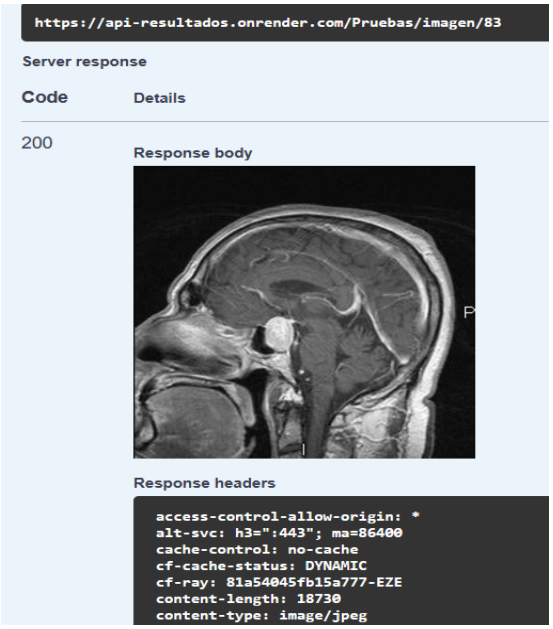
8. Gestión de Admin / Auditor - Imágenes y Resultados

El auditor tiene la posibilidad de realizar predicciones con imágenes para lo cual también utilizará los endpoints de predicción como el **predecir/cerebro** o **predecir/pulmones** que ya están desarrollados, los cuales generan un diagnóstico que se guarda junto a su imagen y resultado. Por otro lado, el auditor no tiene permiso para ver los resultados ya que solo el médico puede hacerlo.

Para esto implementamos en el código verificaciones de rol en las consultas de historial o de un diagnóstico en particular, para no enviar la información de los resultados dados por los modelos en caso de que no sea un médico quien los esté solicitando.

En el siguiente sprint estaremos trabajando en darle más seguridad a estas verificaciones.

Para verificar que las imágenes se estaban guardando correctamente en la base de datos se agregó también el endpoint **/Pruebas/imágen/(id)** para poder visualizar la imagen de un diagnóstico en el mismo swagger de flask. Da una respuesta de tipo **mimetype** que trae la imagen guardada en binario y la muestra según el formato, sea png, jpg, jpeg



9. **Gestión de Admin - Modificación y Baja de usuario**

    - **Alta de usuario:**

      Se agregó el endpoint **POST /admin/alta** que recibe de un formulario los datos del usuario a ingresar y lo agrega a la base de datos. Esta api recibirá los datos necesarios para ingresar un usuario nuevo recibiendo su nombre, dni, contraseña, email y establecimiento al cual pertenece.

    - **Baja usuario:**

      Este endpoint **DELETE /usuarios/<string:dni>,** lo que hará será tomar el usuario de la base de datos con el DNI que se le administra para así poder buscarlo y eliminar toda su info.

    - **Modificación de usuarios:**

      Se agregó el endpoint **PATCH '/update-user-informacion',** que buscará un usuario por DNI (deducimos que era la mejor manera de buscar un usuario sin saber su ID) y luego se le podrá modificar la información necesaria de estos.

10. **Gestión de Profesional de la Salud**

    - **Cargar imagen e informe:**

      Agregamos el endpoint **/predecir/cerebro** que recibe una imagen y datos relevantes para el modelo de clasificación mediante una solicitud POST. Este endpoint se conecta con la API del equipo 0, a la cual le enviamos la imagen y nos devuelve la respuesta del modelo con los distintos tumores de cerebros y la probabilidad de tenerlo.

      Además, se conecta con la base de datos para guardar el diagnóstico una vez obtenido el response de la API del equipo 0.

11. **Gestión de Médico**

    - **Diagnosticos/{diagnostico_Id}:**

      Se prepara para verificar si los médicos existen o no en la tabla según sus roles. Si es médico devuelve el resultado, si es auditor no podrá acceder a los resultados, si es profesional administrador devuelve “rol no válido”.

    - **GET Diagnostico/historial:**

      Se configuró el response para agregar o no el detalle del resultado según los permisos de médico o auditor, para evitar que auditor reciba datos que no debe poder consultar.

12. **Alta de establecimientos médicos**

    Carga de establecimientos médicos a partir de datasets públicos. Se añadieron 67 establecimientos médicos, principalmente de San Miguel, y algunos del resto del país.

13. **Periodicidad de 60 días para contraseñas**

- Las contraseñas deberán ser cambiadas cada 60 días por seguridad, por lo tanto, implementamos una función que permite verificar si es necesario cambiar una contraseña en función de cuándo fue la última vez que se cambió.
- Expresión regular utilizada para cumplir con las políticas de creación de contraseñas:

  '^(?!.*(.).*\1)(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.*[~!@#$%^&*()_+\-=\{\}\|\[\]\\:";\'<>?])[A-Za-z0-9~!@#$%^&*()_+\-=\{\}\|\[\]\\:";\'<>?]{8,}$'

  - **^:** Indica el inicio de la cadena.
  - **(?!.*(.).*\1)**: Utiliza un negative lookahead para asegurarse de que no haya caracteres repetidos en la contraseña. **(.)** captura cualquier carácter y **.*\1** asegura que no haya repeticiones.
  - **[A-Za-z0-9~!@#$%^&*()_+\-=\{\}\|\[\]\\:";'<>?]{8,}:** Define un conjunto de caracteres permitidos (letras mayúsculas y minúsculas, dígitos y símbolos especiales) y especifica que debe haber al menos 8 de estos caracteres.
  - **$:** Indica el final de la cadena.

14. Implementación de Cifrado con AES

AES-128 es suficiente para la mayoría de los casos y ofrece un buen equilibrio entre seguridad y rendimiento. AES-256 se elige cuando se necesita un nivel extremadamente alto de seguridad y se está dispuesto a aceptar una posible disminución del rendimiento. Fue por esto que decidimos implementar AES 128.

Implementación en la capa base de datos

Se utilizó la funcionalidad de variables de entorno de Render para almacenar la clave de cifrado de forma segura.

La clave para el cifrado en la capa de base de datos cumple con políticas de:

- **Longitud de clave:** Utiliza claves largas, preferiblemente de al menos 12 caracteres. Cuanto más larga sea la clave, más difícil será de adivinar mediante fuerza bruta.
- **Combinación de caracteres:** Incluye una combinación de letras mayúsculas, letras minúsculas, números y caracteres especiales en tu clave. Esto hace que sea más difícil de adivinar mediante ataques de diccionario.

El cifrado está ubicado en el POST /usuario/admin/alta, el cual se modificó para que el correo se guarde en la base de datos cifrado.

Desde la base de datos podemos implementar la función:

- **Cifrado:** pgp_sym_encrypt('mail', 'clave','compress-algo=0,cipher-algo=AES128') para encriptar el email con una clave o llave.
- **Descifrado:** pgp_sym_decrypt(email::bytea, 'clave','compress-algo=0,cipher-algo=AES128') desencriptar con la misma llave o clave, esto lo implementamos desde python sin usar librerías.
- **Obtener clave almacenada:** def obtener_clave_desde_Medico() -> clave = os.environ.get('claveMedico')

Query final:

"INSERT INTO public.usuario (nombre, dni, email, password, rol_id, establecimiento_id, fecha_ultima_password, especialidad) VALUES (%s, %s,**pgp_sym_encrypt(%s, %s, %s)**, %s, %s, %s, %s, %s) RETURNING id;"

cursor.execute(consulta, (nombre, dni, email, clave,arg2, password, rol_id, establecimiento_id, fecha_ultima_password, especialidad))

- **GET /Usuarios/{id}**

  Este endpoint sirve para ver el correo de forma no cifrada con la clave.

  En cuanto a ver si se guardó de forma correcta podemos ver en la misma base de datos con la función **pgp_sym_decrypt(email::bytea, 'clave','compress-algo=0,cipher-algo=AES128'**)

  O también en desde el mismo lenguaje python con la query armada, de la siguiente manera
  select_query = """ SELECT id, nombre, dni, **pgp_sym_decrypt(email::bytea, %s, 'compress-algo=0,cipher-algo=AES128')** AS email_descifrado, password, rol_id, establecimiento_id, fecha_ultima_password, especialidad

  FROM usuario WHERE id = %s """

  cursor.execute(select_query, (clave_maestra, medico_id))

La funcionalidad del cifrado antes estaba implementada con la librería fernet, esta se veía en los siguientes endpoints:

- **Post /Medicos -> c**reaba el usuario con todos los datos con entrada json y cifraba el correo, agregaba el datetime también.
- **Get /Medicos/{Id} ->** llamaba la query de toda la fila y desencriptar el correo con la llave para ver que se guardó

15. Test de API REST

En cuanto a las pruebas de la API se probaron las siguientes funcionalidades:

Diagnósticos:

- Test de POST Pulmones
- Test de GET Historial
- Test de POST predecir cerebro
- Test de GET Diagnósticos

Usuarios:

- Test de GET usuarios Médicos
- Test de POST AdminAlta
- Test de PATCH update-user-informacion
- Test de GET Usuario ID

Todas de ellas pasaron las pruebas en donde se probaron Validación de datos de las respuestas y Validación de los usuarios dentro de la base de datos.

Adjunto se encuentra el documento con los test case y los resultados de las pruebas.

16. **Prueba de penetración**

    Las pruebas de penetración resultaron exitosas en términos generales, si bien se presentan alertas de riesgo medio a bajo, estas no recaen en el equipo ya que tienen que ver con el servidor donde está la API, de nuestra parte, nos enfocaremos a mantener las alertas de riesgo “Alto” en 0, ya que estas serán las que más efecto tendrán en nuestra herramienta.

    Adjunto se encuentra el reporte generado por ZAP.

17. **Prueba de integración para conexión con modelos**

    Prueba que llevamos a cabo para verificar la efectividad de la comunicación entre APIs, definición mencionada y explicada en la actividad 3.

18. **Gestión del Proyecto**

    Conjunto de actividades que se llevaron a cabo sobre la gestión del proyecto durante este Sprint:

    - Actualización del plan de proyecto:
        - Requerimientos funcionales
        - WBS
        - Diccionario
        - Calendario (Diagrama de Gantt)
        - Estimaciones para próximos Sprints.
        - Matriz de riesgos.
    - Documentación del informe de avance.
    - Armado de la presentación visual para la reunión formal 3.

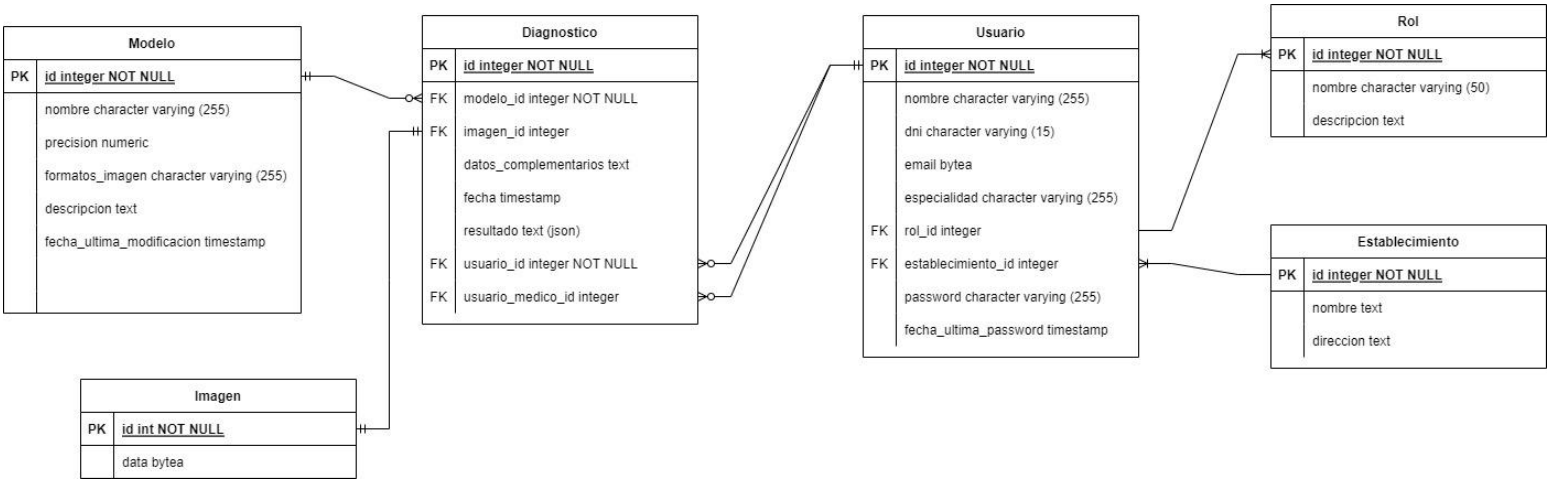
Problemas o inconvenientes detectados

- Problemas de acceso a la BBDD.
- Modificación de endpoints.
- Modificación de Backlog.

Cambios

- **Políticas para contraseñas:** En un principio, las contraseñas de los usuarios serían creadas por nuestro equipo. Por temas de seguridad, entendimos que el usuario es el único que debe tener acceso a su contraseña y perfil, por lo cual es el mismo quien debe crearla.
- **Feedback a modelos:** El modelo se retroalimentará, por lo cual se tendrá en cuenta el feedback de los médicos y se agregó la tarea “Feedback a modelos” en el Sprint 4.
- **Cambio de arquitectura:** Al principio del Sprint se planteó un cambio de arquitectura con los demás equipos para evitar problemas de Performance. Finalmente, este cambio se descartó al agregarse funcionalidades que requerían la primera versión de arquitectura. A su vez, no hubo problemas de Performance en las pruebas implementadas.

- **Cifrado desde la capa de aplicación a la base de datos:** Surgieron varias cuestiones con respecto a la implementación del cifrado AES. Por sugerencia del PO, pasamos de tener el cifrado implementado en la api con librerías, a la base de datos, en donde se aplicó utilizando una clave que almacenamos en las variables de entorno de render, para más seguridad.
- **Modificaciones de endpoints:** Luego de las entrevistas con los expertos, los datos requeridos de los pacientes cambiaron. Esto llevó a modificar las tablas de la BBDD.
- **Modificación DER:** Modificaciones implementadas en el modelo DER de la BBDD. Debido al requerimiento de feedback a modelos, se acordó con el equipo 0 que ellos almacenarán las imágenes que le lleguen para hacer predicciones a sus modelos, además de estar guardadas en nuestra base. Para poder identificar las imágenes tendrán el mismo id en nuestra base de datos como en la de ellos. Por esta razón, tuvimos que agregar una tabla “Imagen” con una columna id a la que se referencia en “diagnostico”, en lugar de guardar el contenido de la imagen directamente en la tabla diagnostico.



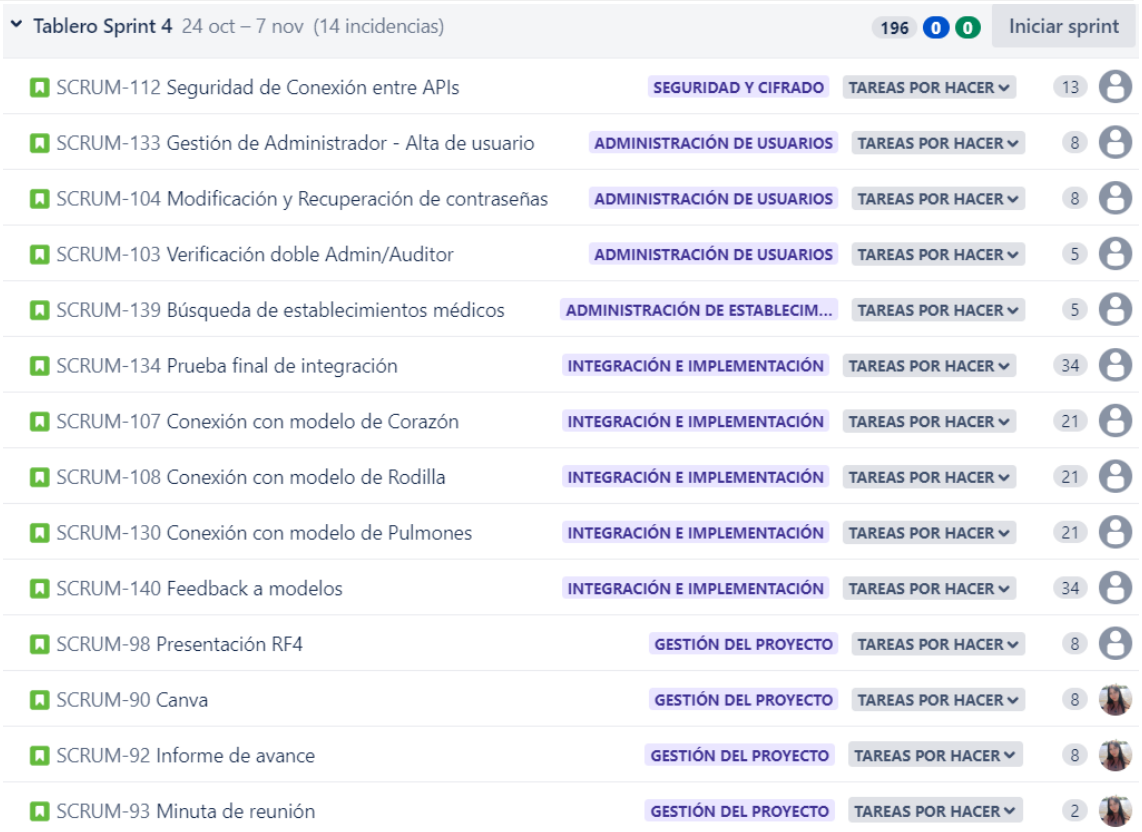
Matriz de Riesgos

	Descripción	Prob. de ocurrencia	Impacto del riesgo	Exposición al riesgo	Acciones para mitigarlo	Contingencia	Responsable
1	Problemas de acceso a la BBDD	3	3	9	Acceder con frecuencia a la BBDD para comprobar que no hayan problemas de conexión.	Gestionar las tareas entre otros miembros del equipo mientras se resuelve el problema.	Líder Técnico
2	Modificación de endpoints	3	3	9	Mantener comunicación constante con los demás equipos para minimizar cambios.	Estimar las tareas de endpoints teniendo en cuenta las posibles modificaciones.	Equipo
3	Modificación de Backlog	3	3	9	Mantener comunicación constante con los demás equipos para minimizar cambios.	Asignar y estimar rápidamente las tareas para llegar a cumplir los plazos establecidos.	Scrum Master Líder Técnico

4	Modificación de requerimientos	3	3	9	Mantener comunicación constante con todos los equipos para minimizar cambios.	Gestionar los cambios para mantener el control de las tareas.	Scrum Master Líder Técnico
5	Caída de BBDD	2	3	6	Mantenimiento del equipo utilizado para la BBDD.	Utilizar la alternativa definida para volver a levantar la BBDD.	Equipo
6	Plazos incumplidos	2	3	6	Estimar las tareas teniendo en cuenta la disponibilidad del responsable.	Re-planificar el siguiente Sprint estimando las tareas incompletas.	Equipo
7	Fallos en la integración con modelos	2	3	6	Realizar pruebas de integración.	Identificar y plantear una solución al fallo con los equipos de modelos.	Equipo
8	Falta de pruebas adecuadas	2	2	6	Diseñar un plan de pruebas y realizar la cantidad de tests necesarios.	Re-planificar el siguiente Sprint, priorizando las pruebas.	Tester
9	Falta de conocimiento y experiencia sobre las tareas asignadas	2	2	4	Capacitación de las herramientas que se utilizarán a futuro.	Estimar el proceso de capacitación del integrante asignado a dicha tarea.	Equipo
10	Problemas de comunicación con los demás equipos	1	3	3	Mantener comunicación regular entre los miembros de los equipos.	Designar un responsable para gestionar la comunicación.	Scrum Master
11	Pérdida de datos maestros	1	3	3	Realizar copias de seguridad semanalmente.	Restauración de la copia de seguridad.	Equipo
12	Problemas en la API	1	3	3	Deployar una API auxiliar que funcione igual a la principal.	Utilizar API auxiliar.	Equipo

Propuesta próximo ciclo

Nuestro siguiente Sprint consta de las siguientes tareas, con sus correspondientes story points estimados:



En principio, para aplicar seguridad en las APIs de todos los equipos, estaremos utilizando las herramientas de JSON Web Token y/o Let's Encrypt. Además, terminaremos la gestión de administrador, pues del ABM quedará pendiente el alta y el registro del usuario.
Por otro lado, vamos a seguir implementando las políticas de contraseñas, aplicando una verificación doble necesaria solamente para el rol de Admin / Auditor, pues es el que tiene acceso a los datos encriptados, por lo que sus usuarios deben poseer más seguridad.
A su vez, también vamos a gestionar la modificación y recuperación de contraseñas.
Continuaremos haciendo las conexiones con los modelos de corazón, rodilla y pulmones.
Por último, haremos una prueba de integración final con los modelos, para asegurarnos de que el rendimiento del sistema es el esperado.

Herramientas y Repositorios

Repositorio Datacript | GitHub
Repositorio API | GitHub
API | Render