MANUAL TÉCNICO

EDU-NeuroRepVIZ

Tabla de Contenidos

Acerca de

El presente documento tiene la finalidad de brindar información relevante para la implementación de NeuroRepVIZ en el largo plazo. Igualmente, presenta las posibilidades de expansión y mejora que se pueden realizar en la herramienta web.

Grupo Objetivo

El contenido de este manual da por hecho que el usuario tiene conocimientos básicos de informática.
Igualmente, que posee conocimientos alrededor del manejo de herramientas web y de bases de datos.

Nota legal

La información presentada en el presente manual hace parte de un trabajo colaborativo realizado entre el grupo IMAGINE y el Hospital Militar Central, como parte del desarrollo de una herramienta web interactiva, basada en un repositorio de neuroimágenes existente, que apoya el proceso de entrenamiento de residentes de radiología mediante la resolución de retos, análisis de casos y la realimentación personalizada.

Toda la información en este manual ha sido preparada con el mejor conocimiento y de acuerdo con el estado del arte. Sin embargo, esto no exime de la posible existencia de errores o inexactitudes. No asumimos ninguna responsabilidad por la exactitud y completitud de esta información.

Sin embargo, estos objetivos conllevan a la creación de ciertas restricciones que aseguren el funcionamiento adecuado de la solución. Por lo tanto, se debe asegurar que el acceso al sistema se de por medio de una lista de control de acceso, mediante la que únicamente usuarios registrados pueden hacer uso de la herramienta. Igualmente, asegurar la anonimización de la información de los pacientes para proteger su identidad y privacidad.

1 Arquitectura del sistema

La arquitectura de la solución esta basada en una solución *full-stack* mediante la cual se lleva a cabo el despliegue de un servidor *front-end*, un sevidor *back-end* y una base de datos. Estas herramientas, en conjunto con una aplicación en *Python*, que permite el manejo de la base de datos, conforman la arquitectura general del sistema.

Como se puede observar en la Figura.1.1, la arquitectura del sistema corresponde a un bloque grande que corresponde a un servidor que hace las veces de *front-end* y *back-end*. Dentro de este, se cuenta con un bloque que es la parte de visualización del sistema RepNeuroVIZ, el cual hace uso de *NAVIO* para el manejo de datos clínicos y previsualización de los datos. Junto a este se encuentra el servidor de base de datos que permite el almacenamiento y visualización de los estudios de radiología realizados y seleccionados por los doctores dado que son considerados importantes para su estudio.

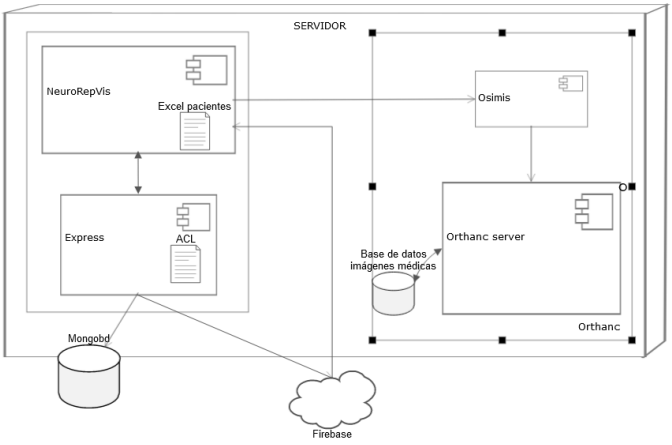


Figura 1.1: Arquitectura NeuroRepVIZ

2 Front-end

La implementación del *front-end* de la aplicación se describirá a continuación, en esta se hará referencia a los paquetes utilizados y la estructura general del proyecto, los cuales se encuentran en una carpeta llamada *RADIOLOGYREP*

2.1. Paquetes

2.1.1. React-multi-select

Es un componente de *JavaScript*, especificamente **JSX** ue permite hacer la selección de varios elementos y puede ser modificado para un uso particular.

2.1.2. Axios

Esta es una libreria que permite realizar peticiones al *back-end*, permitiendo tener un cliente HTTP.

2.1.3. D3

Librería para la generación de visualizaciones de datos, para hacer la lectura de archivos "*.csv*"que permite trabajar en conjunto con NAVIO.

2.1.4. Firebase

Librería que se utiliza para aprovechar las características de la plataforma del mismo nombre, la cual facilita los procesos de autenticación y validación de usuarios, entre otras caracteristicas.

2.1.5. Materialize CSS

Librería que permite hacer uso de diferentes estilos de visualización de la aplicación.

2.1.6. Moment

Librería que permite la manipulación de objetos de tipo f̈echa ën JavaScript, permitiendo realizar comparaciones y generación de marcas de tiempo UNIX.

2.1.7. Navio

Componente de visualización de datos que permite la selección de muestras.

2.1.8. React

Librería que permite la creación de interfaces gráficas, facilitando la renderización de paginas web (HTML). En conjunto con *React datepicker*, *React router dom* y *React toastify*, permiten hacer la creación de elementos, provisionar rutas y mostrar notificaciones en la aplicación web.

2.2. Estructura

La estructura que se puede apreciar en la Figura.2.1 de esta contiene dos carpetas importantes: **public** y **src**, las cuales permiten la visualización de los datos dentro de la herramienta.

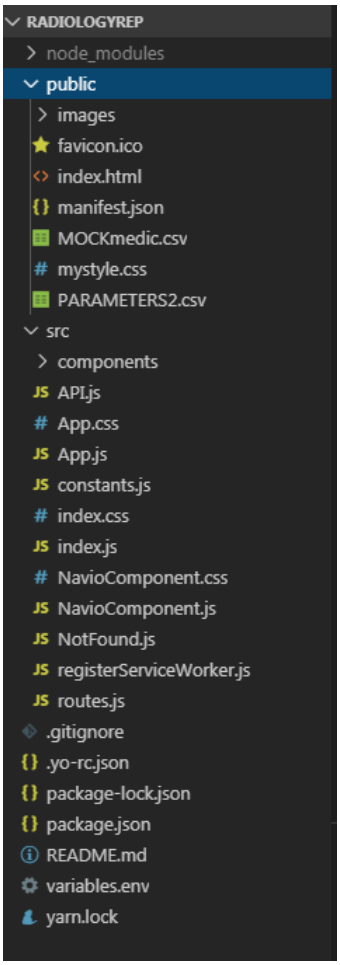


Figura 2.1: Organización *Front-end*

En esta organización adicionalmente se debe tener en cuenta que es un proyecto elaborado con *React*, y por tanto los componentes que se encuentran en la carpeta *src*,

funcionan como etiquetas de contenido en HTML. Por tanto en la Figura. 2.2, se puede apreciar la disposición de estos.

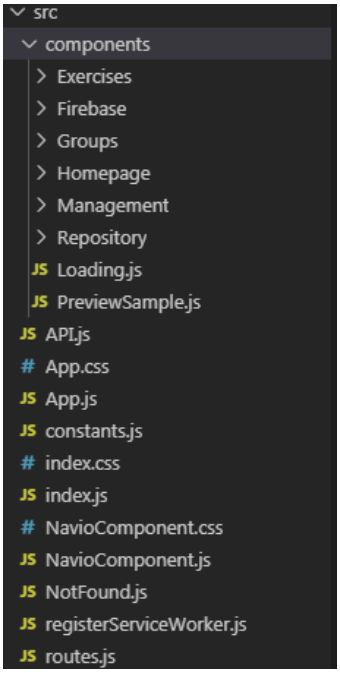


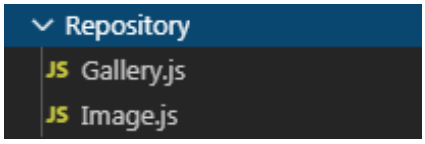
Figura 2.2: Organización ***src***

Donde se tiene que los archivos cumplen las siguientes funciones:

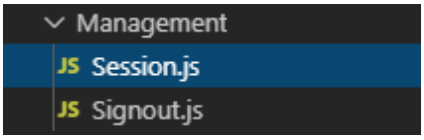
- **Routes.js**: Describe las rutas a utilizar en la aplicación. Se tiene por ahora la ruta inicial '/' la cual sería la página para iniciar sesión/registrarse y '/session' para acceder a la aplicación de por sí.
- **NotFound.js**: Componente que muestra una página donde se le avisa al usuario que la ruta no existe.
- **NavioComponent.js / css:** Componente que muestra el elemento navio. Toda la inicialización para mostrar los datos se hace por aquí. Adicionalmente, el archivo css serían los estilos utilizados.
- **Index.js/css:** Punto de entrada de la aplicación con sus respectivos estilos.
- **Constants.js:** Lista de rutas como variables constantes.
- **App.js /css:** Componente que muestra la página inicial con sus respectivos estilos cuando se ingresa a la aplicación con usuario autenticado.
- **API.js:** constante que apunta a la dirección URL del backend.

- **Loading.js:** Elemento que muestra un '*spinner*' de carga, al momento de cambiar de página, o hacer consultas de datos.
- **PreviewSample.js:** Componente para mostrar información asociada a una submuestra: Navio y Galeria de imágenes.

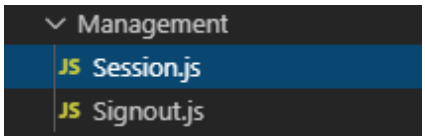
De igual forma, cada una de estas carpetas contiene una serie de archivos que referencian componentes utilizados a lo largo de la herramienta.



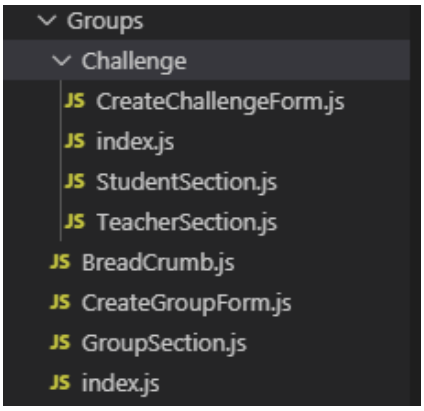
(a) Organización ***Repository***



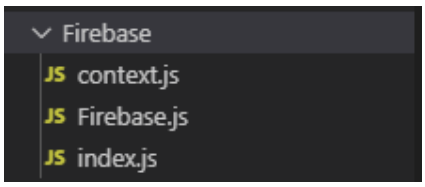
(b) Organización ***Management***



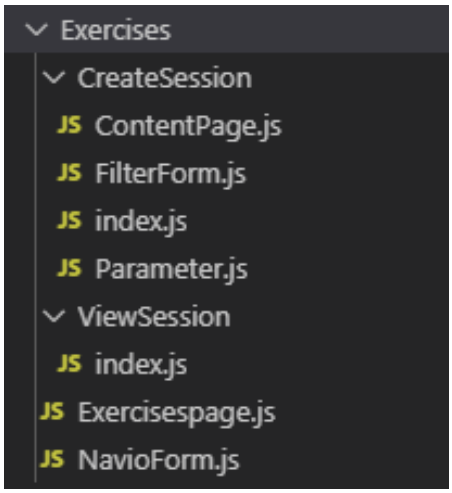
(c) Organización ***Homepage***



(d) Organización ***Groups***



(e) Organización ***Firebase***



(f) Organización ***Exercises***

Figura 2.3: Organización carpetas en ***component***s

Las cuales cumplen con las siguientes funciones:

- Gallery.js: Galería de imágenes de vista previa sobre los estudios de radiología.
- Image.js: Componente que renderiza una sola imagen, es utilizado para que sea renderizado varias veces en Gallery.js
- Session.js: Componente que renderiza la página y diferentes sub-componentes dependiendo del estado actual. Por ejemplo: En caso de estar en página de muestras, este componente renderiza el componente asociado a muestras, lo mismo para grupos.
- Signout.js: Componente que renderiza un botón para cerrar sesión.
- Homepage.js: Componente para mostrar el formulario para iniciar sesión o registrarse. Dependiendo del estado, renderiza Login.js ó SignUp.js.
- Login.js: Componente que muestra solamente el formulario para iniciar sesión.
- SignUp.js: Componente que muestra solamente el formulario para registrarse.
- Index.js: Componente para mostrar y agregar grupos.
- GroupSection.js: Componente que se muestra cuando se hunde click en un grupo específico. Muestra los estudiantes, profesores y retos asociados al grupo.
- CreateGroupForm.js: Formulario para crear un grupo.
- BreadCrumb.js: Componente que muestra las migas de pan o 'breadcrumbs' para facilitar la navegación en la página.
- Challenge/CreateChallengeForm.js: Formulario para crear un reto.
- Challenge/index.js: Componente que renderiza la vista en caso de hundir en un reto. En el caso de que el usuario ingresado sea un profesor renderiza TeacherSection.js, caso contrario renderiza StudentSection.js.
- Challenge/TeacherSection.js: Componente que muestra las respuestas de los estudiantes sobre un reto.
- Challenge/StudentSection.js: Formulario para enviar una respuesta sobre un reto, permite realizar varias entregas.
- Context.js: Componente que se encarga de mantener el estado de autenticación de firebase.
- Firebase.js: Componente que provee todas las funcionalidades requeridas para utilizar la autenticación de firebase.
- Index.js: Componente que utilizando los dos anteriores componentes, permite conectarse a firebase y utilizar las funciones asociadas a la autenticación.

- ViewSession/index.js: Componente que renderiza la muestra utilizando PreviewSample.js por detrás.
- CreateSession/ContentPage.js: Componente que renderiza y maneja la lógica detrás de la creación de una sub muestra, desde tipo de filtro (FilterForm.js ó NavioForm.js) hasta los datos finales que serán enviados al backend.
- CreateSession/FilterForm.js: Formulario para crear una sub muestra utilizando parámetros.
- CreateSession/index.js: Componente que renderiza ContentPage y hace el envío final de los datos al backend.
- CreateSession/Parameter.js: Componente reutilizado en FilterForm.js para renderizar un solo parámetro que haya seleccionado el usuario.
- Exercisespage.js: Componente que renderiza todas las submuestras creadas por el profesor y ademas el boton para ingresar en el proceso de crear muestra (CreateSession/index.js).
- NavioForm.js: Formulario para crear submuestra utilizando el componente de Navio.

3 Back-end

Para la implementación del *back-end* de la aplicación se describirá a continuación, en esta se hará referencia a los paquetes utilizados y la estructura general de la API, los cuales se encuentran en una carpeta llamada *RADIOLOGYREP-BACKEND*.

3.1. Paquetes

- **Body-parses:** Este paquete es utilizado para manejar los requests que entran y salen del servidor, esto quiere decir modificar el cuerpo o leer los requests con el tipo de dato apropiado.
- **CORS:** Este paquete es utilizado para poder hacer cross-origin requests, para poder prototipar rápidamente y no tener que preocuparnos por las configuraciones de la red (no es lo importante de nuestro sistema).
- **Express, Express-formidable:** Este par de paquetes nos permite desplegar nuestro código como un servidor que escucha en un puerto los requests de los usuarios, y también realiza las conexiones necesarias con la base de datos.
- **Fast-csv:** Este paquete lo utilizamos para acceder al ACL (Lista de Control de Acceso), archivo que está en formato csv. Nos permite leer rápidamente el archivo y aplicar las reglas de negocio.
- **Gridfs-stream, multer, multer-gridfs, storage, mongoose:** Estos 5 paquetes son utilizados para acceder a la base de datos, y también poder almacenar archivos en la base de datos.
- **Nodemon:** Este paquete es utilizado únicamente en desarrollo. Es utilizado para correr el código "en caliente". Nos permite actualizar el código del servidor, sin tener que reiniciarlo cada vez que se realizan cambios en el mismo.

3.2. Estructura

Como se puede observar en la Figura.3.1 tenemos 3 módulos importantes: ***API***, *controladores* y *esquemas*. Junto con ellos, contamos con un archivo de configuración y un archivo "**lista.csv**", que es la lista de control de acceso. El archivo de configuración contiene la **URL** de acceso a la base de datos de *mongodb*. La cual en caso de requerir cambiarse, únicamente se cambia la URL de acceso y un usuario permitido en la base de datos.

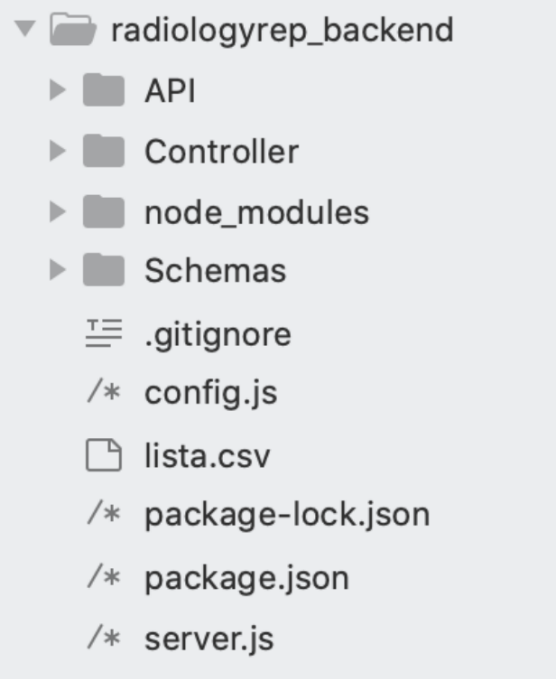


Figura 3.1: Estructura de directorio *back-end*

3.3. Controladores

En la parte de controladores, disponemos de un controlador para cada uno de los recursos a utilizar. Como se puede observar en la Figura 3.2.

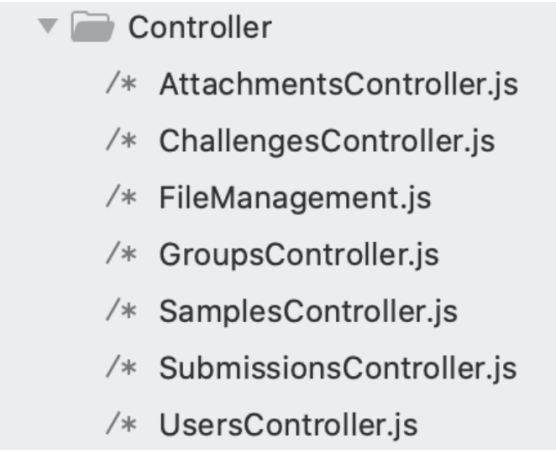


Figura 3.2: Estructura de directorio controladores *back-end*

Cada uno de estos controladores contiene los métodos que aplican las reglas de negocio y además hacen el manejo del recurso. Es decir, como se crean, como se acceden a estos y como se almacena la información de estos en la base de datos.

- **AttachmentsController:** Este controlador tiene 6 métodos: Crear, actualizar archivo de entrega, borrar, obtener recurso por Id, obtener recurso por Id del dueño y obtener todos los recursos en la colección.
- **ChallengesController:** Este controlador tiene 5 métodos: Crear reto, actualizar estado del reto, borrar, obtener reto por Id y obtener todos los retos en la colección.
- **FileManagement:** Este controlador tiene 3 métodos: Obtener todos los archivos subidos, borrar un archivo y descargar un archivo específico.
- **GroupsController:** Este controlador tiene 9 métodos: Crear, actualizar, borrar, obtener grupo por Id, obtener todos los grupos en la colección, agregar estudiante a grupo, quitar estudiante de grupo, agregar profesor a grupo, quitar profesor de grupo.
- **SamplesController:** Este controlador tiene 9 métodos: Crear muestra, actualizar muestra, borrar, obtener muestra por Id y obtener todas las muestras en la colección.
- **SubmissionsController:** Este controlador tiene 5 métodos: Crear entrega, actualizar entrega, borrar, obtener entrega por Id y obtener todas las entregas en la colección.
- **UsersController:** Este controlador tiene 6 métodos: Crear, actualizar, borrar, obtener usuario por id, obtener usuario por email y obtener todos los usuarios en el sistema.

3.4. API

En la **API**, existe una interfaz de acceso por cada recurso existente, como se puede observar en la Figura. 3.3.

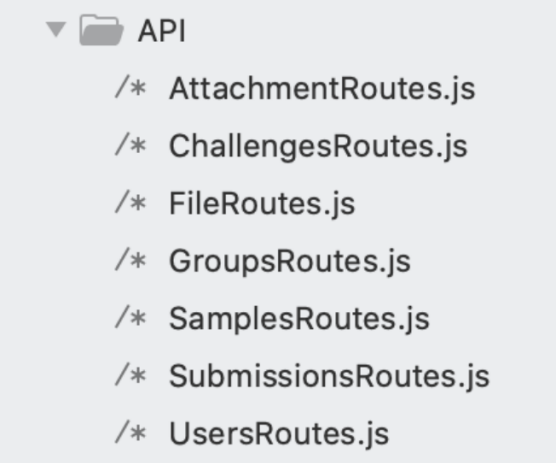


Figura 3.3: Estructura directorio API *back-end*

Cada una de estas interfaces de acceso se tiene las ruta para manipular y acceder a los recursos del *back-end* haciendo uso de los controladores previamente descritos.

3.5. Esquemas

Para cada recurso utilizado en la aplicación tenemos un esquema asociado. Este esquema describe la estructura de información (objeto) que se va a guardar dentro de la base de datos en una colección. Estos se ecuentran en la Figura.3.4

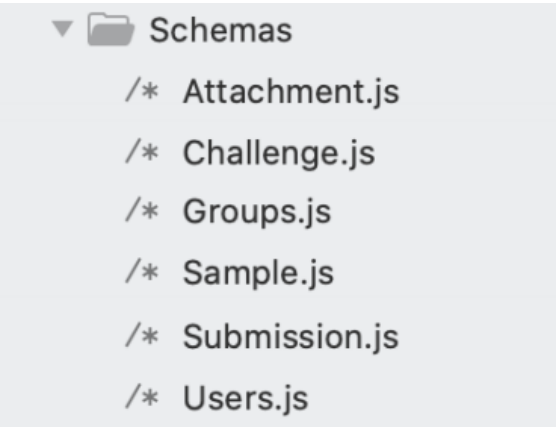


Figura 3.4: Estructura directorio modelo de datos *back-end*

Y por tanto,se puede ver como interactúan estos esquemas (Figura.3.4) considerando

que puede contener cada uno de los objetos creados.

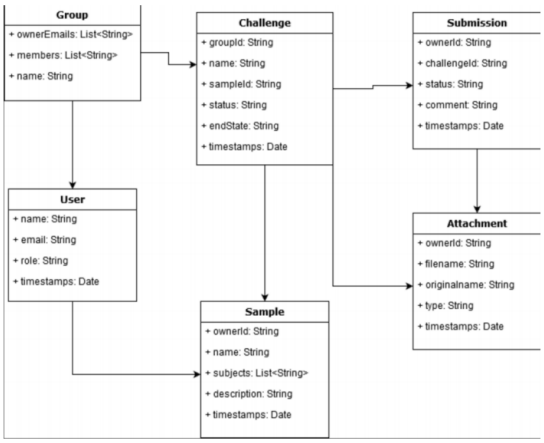


Figura 3.5: Diagrama UML de datos

- **Attachment:** Describe la estructura de información que se va a guardar por cada archivo.
- **Challenge:** Describe la estructura de información que se va a guardar por cada reto creado.
- **Groups:** Describe la estructura de los grupos de clase, para asignar retos y enviar retroalimentaciones a estudiantes.
- **Sample:** Describe la estructura que tiene cada muestra, este objeto almacena la información de cuales sujetos se tienen que mostrar en la visualización de NAVIO, en cada uno de los retos.
- **Submission:** Describe la estructura que tiene cada entrega de un estudiante, y además contiene la retroalimentación hecha por el profesor.
- **Users:** Guarda la información del usuario y un identificador asociado que se utiliza para relacionar el usuario con las muestras, los grupos y las entregas hechas.

4 Manejo base de datos de imágenes

El manejo de la base de datos del sistema, se realiza mediante la conexión al PACS, el cual facilita la posterior visualización de datos. Este manejo se hace por medio de un programa desarrollado en *Python*, el cual permite una conexión sencilla a la plataforma. Esta conexión se da por una serie de *scripts*, que permiten la conexión. Estos se encuentran en una carpeta llamada *PYTHONSCRIPTS*.

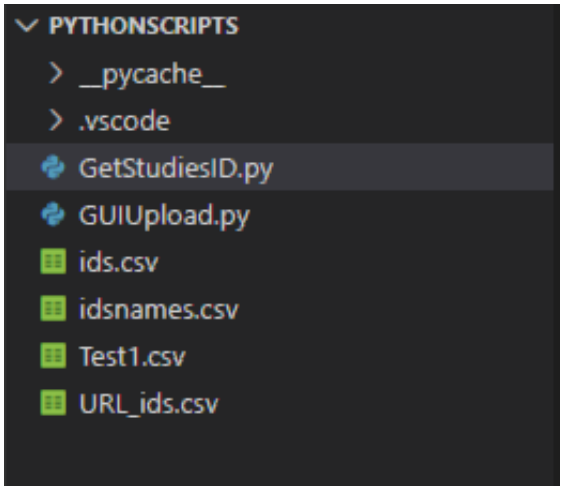


Figura 4.1: Carpeta con *scripts* de Python

Como se puede observar en la Figura. 4.1, esta contiene dos archivos "*.py*"los cuales hacen referencia a una interfaz gráfica que permite el manejo de las funciones de manera sencilla ("*GUIUpload.py*") y otro que contiene las funciones a ejecutar ("*GetStudiesID.py*"). Dentro de este último se encuentran las funciones que se pueden observar en la Figura. 4.2, las cuales permiten acceder, recuperar, agregar y anonimizar los estudios existentes dentro de la base de datos de Orthanc.

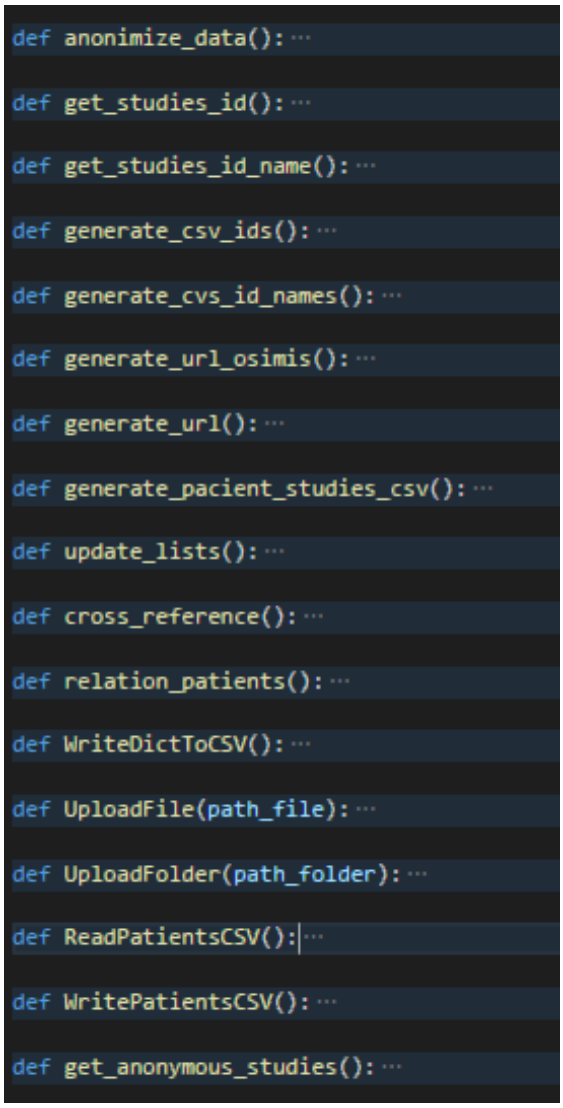


Figura 4.2: Funciones implementadas en *GetStudiesID.py*

5 Despliegue

La versión 1.0 se encuentra desplegada actualmente en un servidor en la Universidad de los Andes, dentro de la red privada del laboratorio COLIVRI. Este despliegue puede ser trasladado cuando sea requerido al Hospital Militar teniendo en cuenta las restricciones necesarias para el funcionamiento del servidor.

En este despliegue se cuenta con acceso a la plataforma final de la versión 1.0 del repositorio, la cual luce como se puede apreciar en las Figuras. 5.1,5.2. Igualmente, se tiene la oportunidad de hacer uso de las funciones presentadas en el informe anterior de **Pruebas Funcionales y No Funcionales** donde se encuentran detalladas, junto con la utilización del la interfaz de usuario para el manejo de imágenes en Orthanc.

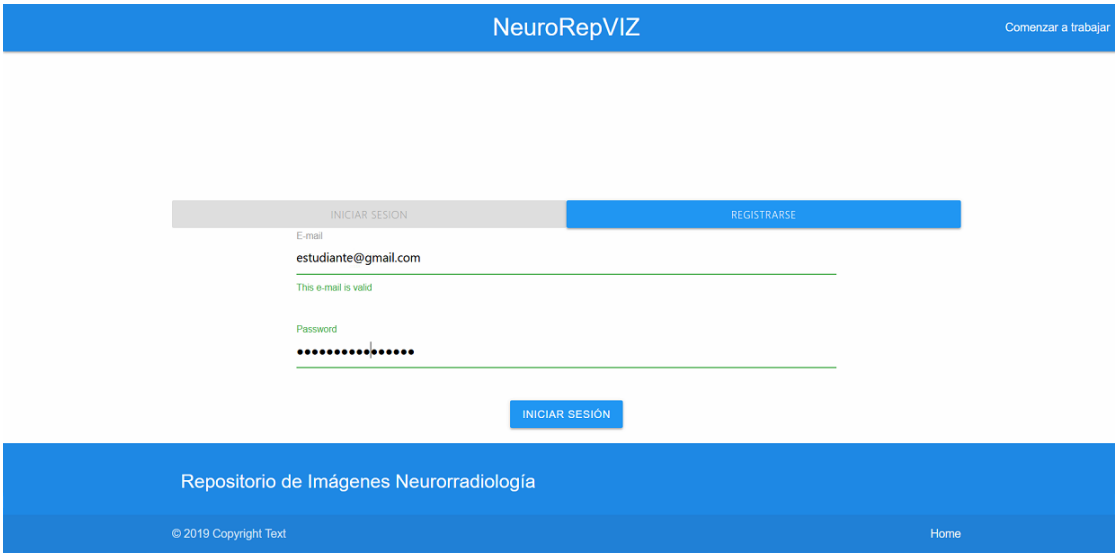


Figura 5.1: Página de inicio del repositorio

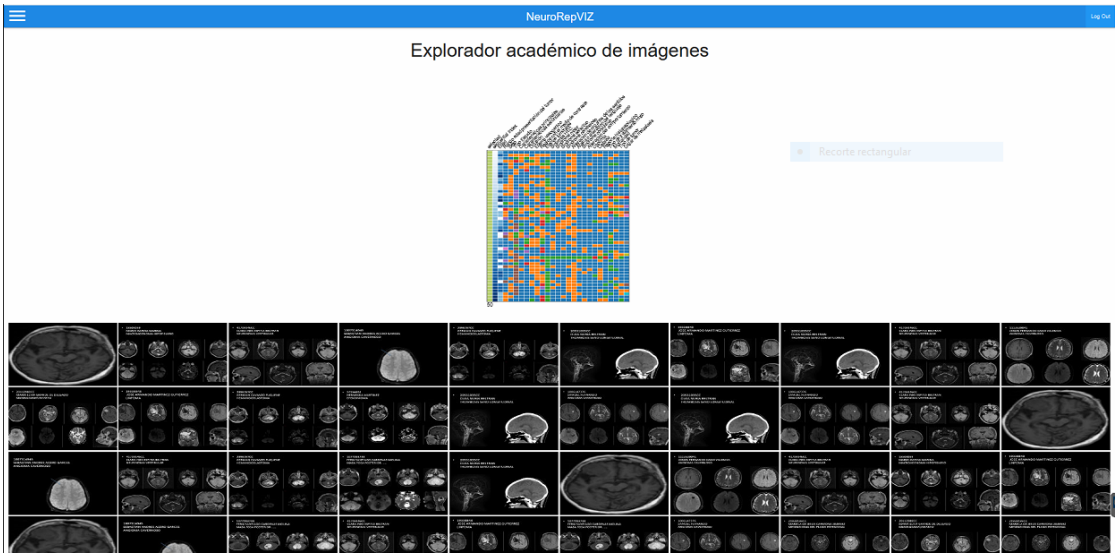


Figura 5.2: Explorador de Imágenes diagnosticas

Revisiones

Versión	Creado por:	Revisado por:	Fecha (dd-mm-aaaa)
0.0	Francisco Durango	José Tiberio Hernandez	22-12-2020