|  |
| --- |
| **Control del Documento** |

**Proyecto**

*Sistema web para el control de la calidad del agua*

**Título**

*Arquitectura y diseño del sistema – V1*

**Generado por**

*Martín Daniel Perez Puchuri*

***Aprobado por***

*Martín Daniel Perez Puchuri*

|  |
| --- |
| **Índice** |

**Control del Documento 1**

**Índice 2**

**Arquitectura y entrenamiento de la Red Neuronal 3**

**Arquitectura del Sistema Web 4**

**Arquitectura del Microservicio de Autentificación de Usuarios 4**

**Arquitectura del Microservicio de Autentificación de Lecturas 4**

**Arquitectura del Microservicio de la Red Neuronal 4**

1. **Arquitectura y entrenamiento de la Red Neuronal**
   1. *Diseño del modelo*

La arquitectura de esta red neuronal está dividida en una capa de entrada, dos capas ocultas y una capa de salida, las cuales serán descritas brevemente a continuación.

La capa de entrada tiene 8 neuronas, una por cada parámetro de calidad acuífera seleccionado para esta red neuronal artificial, los cuales son: pH, nivel oxígeno disuelto medido en mg/L, temperatura medida en grados Celsius, nivel de nitrógeno medido de manera porcentual a la saturación total, nivel dióxido de carbono medido en mg/L, nivel de sólidos suspendidos medidos en mg/L, nivel de sólidos disueltos medidos en mg/L y nivel de calcio medido en mg/L.

La primera capa oculta tiene 4 neuronas, ya que es una convención utilizar la mitad de las neuronas de la capa de entrada. La función de activación de esta capa es el rectificador linear (RELU).

La segunda capa oculta tiene 4 neuronas, siendo esta la última capa oculta. La función de activación de esta capa es el rectificador linear (RELU).

La capa de salida emplea la función sigmoide como función de activación ya que permite mostrar el nivel de contaminación acuífera de manera porcentual variando desde el cero (0) hasta el uno (1), mientras más cercano sea el valor a uno (1) este nos indica que el agua está en estado óptimo para la vida acuática.

Chart, diagram, bubble chart

Description automatically generated

Arquitectura de la Red Neuronal Artificial propuesta

* 1. *Dataset*

La data llegó en formato JSON y tuvo que ser transformada a CSV para poder ser empleada por la red neuronal, conteniendo un total de 10 mil lecturas.

Como podemos observar en la Tabla 3, la data contenía los parámetros fundamentales para determinar la calidad del agua: temperatura, nivel de pH, nivel de oxígeno disuelto, nivel de nitrógeno, nivel de dióxido de carbono, cantidad de solidos suspendidos, cantidad de solidos disueltos, nivel de calcio y finalmente el estado del agua según la lectura expresado con un 0 o 1, donde 1 representa un estado óptimo y 0 un mal estado.

PARÁMETROS PARA DETERMINAR LA CALIDAD DEL AGUA.

|  |  |
| --- | --- |
| **Parámetro** | **Unidad de medida** |
| Temperatura | C° |
| Potencial Hidrógeno | pH |
| Oxígeno Disuelto | mg/L |
| Sólidos Disueltos | mg / L |
| Sólidos Suspendidos | mg / L |
| CO2 | mg/L |
| Nitrógeno | % Saturación Total |
| Calcio | mg / L |

* 1. *Entrenamiento*

Para realizar el entrenamiento de los modelos primero se importaron las librerías de Python básicas para el tratamiento de data numpy, pandas y sklearn, luego se dividió el dataset para la data de entrenamiento y de testeo, dándole a esta última el 20% del total de lecturas.

Los siguientes pasos fueron realizar el escalamiento de las variables, ya que estás tenían valores que se encontraban en rangos numéricos diferentes, lo que dificulta a la red neuronal para darle peso a los mismos, posteriormente se importaron dos submódulos de la librería Keras, Sequential y Dense, para realizar la red neuronal.

Como siguiente paso se procedió a la creación de la red neuronal, definiendo la capa de entrada, la primera capa oculta y la segunda capa oculta, así como el optimizador y la función usada para la pérdida. Se definieron 100 épocas, analizando la data en lotes de 10 registros.

* 1. *Validación*

Para validar el modelo y la red neuronal se utilizaron las herramientas que nos brinda la librería de sklearn para las métricas, se usó accuracy\_score, que mide la exactitud de la red neuronal empleando la data real y de prueba, obteniendo un resultado de 93% de exactitud.

Posteriormente se realizó la matriz de confusión para nuestro modelo la cual se observa en la Figura 3, obteniendo valores que van acorde a la exactitud que se calculó anteriormente, en este caso podemos ver la cantidad de lecturas que fueron acertadas y los falseos positivos y negativos, los cuales están dentro del umbral de lo aceptable.

Text

Description automatically generated with low confidence

Matriz de confusión

Seguido de esto como podemos observar en la Figura 4, evaluamos la evolución de la precisión del modelo a través de las épocas, el modelo alcanza niveles aceptables de precisión a partir de la época 25, obteniendo un valor que oscila en el intervalo de 92% y 94%.

Chart, line chart

Description automatically generated

Evolución de la precisión del modelo

Posteriormente se realizó la gráfica para evaluar la pérdida del modelo como se muestra en la Figura 5, el modelo alcanza niveles estables a partir de la época 59, obteniendo valores muy cercanos a 0.15, lo cual es aceptable y denota que la red neuronal encontró una buena estabilidad.

Chart, histogram

Description automatically generated

Evolución de la pérdida del modelo

Finalmente se guardó el modelo y el escalador para poder ser reutilizados en el microservicio de la red neuronal.

1. **Arquitectura del Sistema Web**

Esta parte del sistema servirá de interfaz de usuario para el registro y logueo de nuevos usuarios, así como también para el uso de la herramienta de clasificación de nivel de contaminación acuífera en base a lecturas básicas que el usuario deberá ingresar.

El módulo del cliente cuenta con 5 submódulos que se encargan de partes específicas y bien definidas para mantener la alta cohesión y bajo acoplamiento.

Diagram

Description automatically generated

*Módulo del Cliente*

* 1. *Arquitectura del Sistema Web*
     1. **Módulo de Redux**

El módulo de Redux se encarga del manejo del estado de la aplicación, al ser una aplicación web, al refrescar el navegador se pierden los datos que en ese momento se estaban presentando en la pantalla, este módulo se encarga de mantener en el tiempo algunos datos importantes, en este caso el usuario y selecciones de la interfaz, sin embargo, no es su única función debido a que la librería de Redux ejecuta acciones síncronas esto facilita el testing y la depuración de la aplicación, lo cual es muy útil para el desarrollo.

Contiene dos archivos clave que se proceden a explicar:

* **Root-reducer.ts:** Este archivo registra todos los reducer que estamos empleando a lo largo de la aplicación, en este caso, tendremos dos reducers el reducer de lecturas y el reducer de usuarios.
* **Store.ts:** Este archivo es el principal, crea una instancia del del objeto “store” que se encarga de registrar y modificar el estado actual de la aplicación.

Este módulo cuenta con dos submódulos que son los siguientes:

* 1. **Lectures**

Este submódulo se encarga de registrar el estado de las lecturas en general. Cuenta con los siguientes archivos:

* **Lectures.reducer.ts:** Aquí se registra el estado por defecto que tendrá el reducer de Lecturas, así como las acciones que alterarán su estado al ejecutarse. Este archivo guarda referencia al estado actual de este reducer.
* **Lectures.selector.ts:** Aquí se registran los selectores sobre los campos de Lectures que serán empleados a lo largo de la aplicación para no hacer una llamada al método getState() del store de manera constante y evitar reescribir código.
* **Lectures.types.ts:** Aquí se registran las interfaces que tendrán como función definir los nombres y tipos de datos que pasarán a través de las acciones de Lecturas.
* **Interfaces:** Esta carpeta contiene la interface lecture.interface.ts que se empleará para generar objetos de tipo Lecture a partir de los JSON que llegan desde el servidor.
  1. **User**

Este submódulo se encarga de registrar el estado del usuario en general. Cuenta con los siguientes archivos:

* **User.reducer.ts:** Aquí se registra el estado por defecto que tendrá el reducer de Usuario, así como las acciones que alterarán su estado al ejecutarse. Este archivo guarda referencia al estado actual de este reducer.
* **User.selector.ts:** Aquí se registran los selectores sobre los campos de User que serán empleados a lo largo de la aplicación para no hacer una llamada al método getState() del store de manera constante y evitar reescribir código.
* **User.types.ts:** Aquí se registran las interfaces que tendrán como función definir los nombres y tipos de datos que pasarán a través de las acciones de User.
* **Interfaces:** Esta carpeta contiene la interface user.interface.ts que se empleará para generar objetos de tipo User a partir de los JSON que llegan desde el servidor.

Diagram

Description automatically generated

*Figura X: Módulo de Redux*

* 1. **Módulo de Lecturas**

El módulo de Lecturas se encarga de guardar los componentes y la lógica referente a las Lecutras, abarca las pantallas, los data transfer objects y las acciones que se ejecutan en cada página.

Este módulo cuenta con dos submódulos que son los siguientes:

* 1. **Consult**

Este submódulo se encarga de la lógica referente a la realización de consultas al microservicio de la red neuronal sobre el estado de contaminación acuífera. Está comprendido por los siguientes elementos:

* **ConsultPage.tsx:** Este archivo contiene la interfaz que el usuario empleará para realizar una consulta, tendrá botones y forms para recibir y mostrar información.
* **Consult.actions.ts:** Este archivo contiene las acciones que se realizan al interactuar con los forms y botones de la página de consultas, disparan acciones síncronas y asíncronas a la espera de la respuesta del microservicio de la red neuronal o para modificar la interfaz de la pantalla.
* **DTO:** Esta carpeta contiene el data transfer object **consult.dto.ts** que se empleará como modelo para enviar datos hacia el microservicio de la red neuronal encargado de generar la predicción en base a la data ingresada.
  1. **Records**

Este submódulo se encarga de la lógica referente a la lectura de consultas previas sobre el estado de contaminación acuífera. Está comprendido por los siguientes elementos:

* **RecordsPage.tsx:** Este archivo contiene la interfaz que el usuario empleará para leer las consultas que realizó previamente, tendrá botones y tablas para recibir y mostrar información.
* **Records.actions.ts:** Este archivo contiene las acciones que se realizan al interactuar con los botones de la página de records, disparan acciones síncronas y asíncronas a la espera de la respuesta del microservicio de autentificación o para modificar la interfaz de la pantalla.
* **DTO:** Esta carpeta contiene el data transfer object **lecture.dto.ts** que se empleará como modelo para enviar datos hacia el microservicio autentificación para recibir las lecturas que el usuario realizó previamente.

Diagram

Description automatically generated

*Figura X: Módulo de Lecturas*

* 1. **Módulo de Usuarios**

El módulo de Usuarios se encarga de guardar los componentes y la lógica referente a los Usuarios, abarca las pantallas, los data transfer objects y las acciones que se ejecutan en cada página.

Este módulo cuenta con dos submódulos que son los siguientes:

* 1. **Auth**

Este submódulo se encarga de la lógica referente a la seguridad y autentificación del usuario en el sistema. Está comprendido por los siguientes elementos:

* **AuthPage.tsx:** Componente raíz de la aplicación, se carga al comienzo y su función es contener las pantallas de login, registro y la ruta de la plataforma que se verá al loguearte.
* **ProtectedRoute.tsx:** Este componente se encarga de verificar si el usuario está logueado en la plataforma, de ser así muestra el componente hijo que posee, si no, avisa al usuario que no tiene permitido el acceso a dicho componente, es una manera de proteger las rutas del sistema.
* **DTO:** Esta carpeta contiene el data transfer object **lecture.dto.ts** que se empleará como modelo para enviar datos hacia el microservicio autentificación para recibir las lecturas que el usuario realizó previamente.

Este módulo cuenta con dos submódulos que son los siguientes:

* 1. **Login**

Contiene la lógica encargada del login en la plataforma en los siguientes archivos:

* **LoginPage.tsx:** Contiene la interfaz de la pantalla de Login de la aplicación comprendiendo botones y un form.
* **Login.actions.ts:** Contiene las acciones y métodos pertinentes a la pantalla de Login.
  1. **Register**

Contiene la lógica encargada del registro en la plataforma en los siguientes archivos:

* **RegisterPage.tsx:** Contiene la interfaz de la pantalla de Registro de la aplicación comprendiendo botones y un form.
* **Register.actions.ts:** Contiene las acciones y métodos pertinentes a la pantalla de Registro.
  1. **Settings**

Este submódulo se encarga de la lógica referente a la edición de los datos de usuario. Está comprendido por los siguientes elementos:

* **SettingsPage.tsx:** Interfaz de la página de opciones, desde aquí el usuario podrá editar sus datos personales como crea pertinente.
* **Settings.actions.ts:** Este archive contiene las acciones pertinentes a la interfaz de **SettingsPage**, emplea métodos síncronos y asíncronos.
* **Settings.dto.ts:** Es el data transfer object que se empleará como modelo para enviar datos hacia el microservicio autentificación para editar la información el usuario, desde su nombre, su email y contraseña.

Diagram

Description automatically generated

*Figura X: Módulo de Usuarios*

1. **Arquitectura del Microservicio de Autentificación de Usuarios**

Esta parte del sistema servirá para autenticar y guardar un registro de los usuarios de la plataforma y de las lecturas realizadas, este microservicio se conectará a una base de datos en la nube.

Contiene cuatro archivos clave que se proceden a explicar:

* **App.Module.ts:** Este archivo registra todos los módulos a emplear, así como también la conexión a la base de datos en la nube.
* **Main.ts:** Este archivo es el principal, se encarga de inicializar el microservicio de autentificación, habilitar los cors, habilitar los pipes para validación de data transfer objects y configurar el puerto a usar.
* **Utils.ts:** Este archivo contiene métodos que podrán ser accedidos de manera global, para realizar conversiones, validaciones, etc.
* **Enums.ts:** Este archivo contiene enums que serán empleados de manera global para evitar escribir strings de manera manual y facilitar la edición de variables.

Diagram

Description automatically generated

*Arquitectura del Microservicio de Autentificación*

Este módulo está comprendido por los siguientes submódulos:

* 1. **Módulo de Autentificación**

Módulo encargado de la autentificación de usuario en la plataforma mediante el empleo de Json Web Tokens, comprende los siguientes archivos:

* **Auth.module.ts:** Este archivo se encarga de registrar todas las partes del módulo de autentificación, incluyendo el controlador, el servicio y la strategy, así como también de la configuración de los Json Web Tokens.
* **Auth.controller.ts:** Este archivo es el enrutador del módulo, recibe las peticiones del cliente y deriva a los métodos de validación de usuario.
* **Auth.service.ts:** Este archivo es el servicio encargado de verificar la existencia de un usuario ingresado en la base de datos, sirve de nexo entre la base de datos y el controlador.
* **Jwt-auth.guard.ts:** Este archive se encarga de validar los tokens que llegan desde el cliente devolviendo una excepción si este es inválido.
* **Jwt-strategy.ts:** Este archivo sirve de estrategia local para validar a los usuarios, también se configuran los parámetros para la validez de un token.

Diagram

Description automatically generated

*Módulo de Autentificación*

* 1. **Módulo de Usuarios**

Este módulo se encarga de lo referente a la entidad de Usuarios, contiene los siguientes archivos:

* **Users.module.ts:** Este archivo se encarga de registrar todas las partes del módulo de usuarios, incluye el controlador, el servicio, el módulo de confirmación de usuario por mensajería y las colecciones que se emplearán en la base de datos.
* **Users.controller.ts:** Este archivo es el enrutador del módulo, recibe las peticiones del cliente y deriva a los métodos de validación de usuario. También cumple la función de validación de los datos que son recibidos mediante los data transfer objects y una librería para validación de datos.
* **Users.service.ts:** Este archivo es el servicio que hace de nexo entre la base de datos y el controlador, se encarga de recibir peticiones y ejecutar métodos de búsqueda, eliminación, edición y lectura, devolviéndole la información requerida al controlador.
* **DTO:** Este paquete contiene los distintos data transfer object que se emplean como modelo para recibir datos desde el cliente con la finalidad de validarlos y tener un mayor control de los datos que el usuario manda en las peticiones referentes a las lecturas. Entre ellos están:
  + **Login.dto.ts:** DTO empleado para el login del usuario en el sistema.
  + **Register.dto.ts:** DTO empleado para el registro del usuario en el sistema.
  + **Change-pass.dto.ts:** DTO empleado para cambiar la contraseña del usuario.
  + **Lost-pass.dto.ts:** DTO empleado para recuperar la contraseña del usuario.
  + **Edit-user.dto.ts:** DTO empleado para editar los datos del usuario.
* **User.schema.ts:** Este archivo contiene la forma de los documentos que serán almacenados en la colección de lecturas de la base de datos, es propio de Mongo DB.
* **User.model.ts:** Este archivo contiene el modelo de los objetos que serán extraídos de la base de datos y que podrán usarse en el código como entidades que interactúan a lo largo del sistema.
* **Errors.enum.ts:** Este archivo contiene el enum que se empleará para mostrar los mensajes de error a lo largo del módulo de usuarios, esto con la finalidad de evitar escribirlos de forma manual y facilitar la edición de las variables.

Diagram

Description automatically generated

*Figura X: Módulo de Usuarios*

1. **Arquitectura del Microservicio de Autentificación de Lecturas**

Esta parte del sistema servirá para autenticar y guardar un registro de los usuarios de la plataforma y de las lecturas realizadas, este microservicio se conectará a una base de datos en la nube.

Contiene cuatro archivos clave que se proceden a explicar:

* **App.Module.ts:** Este archivo registra todos los módulos a emplear, así como también la conexión a la base de datos en la nube.
* **Main.ts:** Este archivo es el principal, se encarga de inicializar el microservicio de autentificación, habilitar los cors, habilitar los pipes para validación de data transfer objects y configurar el puerto a usar.
* **Utils.ts:** Este archivo contiene métodos que podrán ser accedidos de manera global, para realizar conversiones, validaciones, etc.
* **Enums.ts:** Este archivo contiene enums que serán empleados de manera global para evitar escribir strings de manera manual y facilitar la edición de variables.

Diagram

Description automatically generated

*Arquitectura del Microservicio de Lecturas*

* 1. **Módulo de Lecturas**

Este módulo se encarga de lo referente a la entidad de Lecturas, contiene los siguientes archivos:

* **Lectures.module.ts:** Este archivo se encarga de registrar todas las partes del módulo de lecturas, incluye el controlador, el servicio y las colecciones que se emplearán en la base de datos.
* **Lectures.controller.ts:** Este archivo es el enrutador del módulo, recibe las peticiones del cliente y deriva a los métodos de validación de usuario. También cumple la función de validación de los datos que son recibidos mediante los data transfer objects y una librería para validación de datos.
* **Lectures.service.ts:** Este archivo es el servicio que hace de nexo entre la base de datos y el controlador, se encarga de recibir peticiones y ejecutar métodos de búsqueda, eliminación, edición y lectura, devolviéndole la información requerida al controlador.
* **Lecture.dto.ts:** Este archivo contiene el data transfer object que se emplea como modelo para recibir datos desde el cliente con la finalidad de validarlos y tener un mayor control de los datos que el usuario manda en las peticiones referentes a las lecturas.
* **Lecture.schema.ts:** Este archivo contiene la forma de los documentos que serán almacenados en la colección de lecturas de la base de datos, es propio de Mongo DB.
* **Lecture.model.ts:** Este archivo contiene el modelo de los objetos que serán extraídos de la base de datos y que podrán usarse en el código como entidades que interactúan a lo largo del sistema.
* **Errors.enum.ts:** Este archivo contiene el enum que se empleará para mostrar los mensajes de error a lo largo del módulo de lecturas, esto con la finalidad de evitar escribirlos de forma manual y facilitar la edición de las variables.

Diagram

Description automatically generated

*Módulo de Lecturas*

1. **Arquitectura del Microservicio de la Red Neuronal**

El microservicio de la Red Neuronal al estar escrito en el lenguaje de programación Python comprende un solo archivo para ejecutar toda la lógica de la red neuronal, sus elementos son los siguientes:

* **App.py:** Este archivo se encarga de arrancar el microservicio y cargar la red neuronal previamente entrenada. Hace las veces de servicio y enturador, ya que, las consultas desde el cliente llegan a este archivo este valida los datos de entrada y manda la lectura a la red neuronal, luego de realizar la predicción envía esta misma como respuesta al cliente en formato JSON.
* **RNA.pkl:** Este archivo contiene la red neuronal previamente entrenada y con los pesos fijados, es empleado por **App.py** para realizar predicciones.

Diagram

Description automatically generated

*Arquitectura del Microservicio de la Red Neuronal*

1. **Esquema de la base de datos**

Se eligió Mongo DB como motor de base de datos porque permite guardar los datos no solo en formato JSON sino también en CSV los cuales serán usados de manera posterior para seguir entrenando a la red neuronal.

La base de datos comprende dos colecciones, las cuales son:

* 1. **Lectures**

Esta colección se encarga de almacenar las lecturas realizadas por los usuarios y comprende los siguientes parámetros:

* **Id:** Identificador único del documento, es de tipo ObjectId.
* **Ph:** Nivel de pH en la lectura, es de tipo Number.
* **DissolvedOx:** Nivel de Oxígeno Disuelto en la lectura, es de tipo Number.
* **Temperature:** Temperatura en grados Celsius en la lectura, es de tipo Number.
* **Nitrogen:** Nivel de Nitrógeno en la lectura, es de tipo Number.
* **WaterTurb:** Nivel de Turbulencia del Agua, es de tipo Number.
* **Phosphorus:** Nivel de Fósforo en la lectura, es de tipo Number.
* **Date:** Fecha en la que se realizó la consulta, es de tipo String.
* **User:** Referencia al Id del usuario que realizó la lectura, es de tipo ObjectId.
  1. **Users**

Esta colección se encarga de almacenar las lecturas realizadas por los usuarios y comprende los siguientes parámetros:

* **Id:** Identificador único del documento, es de tipo ObjectId.
* **Email:** Email del usuario, es de tipo String.
* **Password:** Contraseña cifrada del usuario, es de tipo String.
* **FirstName:** Nombres del usuario, es de tipo String.
* **LastName:** Apellidos del usuario, es de tipo String.
* **Verified:** Campo que indica si el usuario verificó su email o no, es de tipo Boolean.

Table

Description automatically generated with low confidence

*Esquema de la Base de Datos*