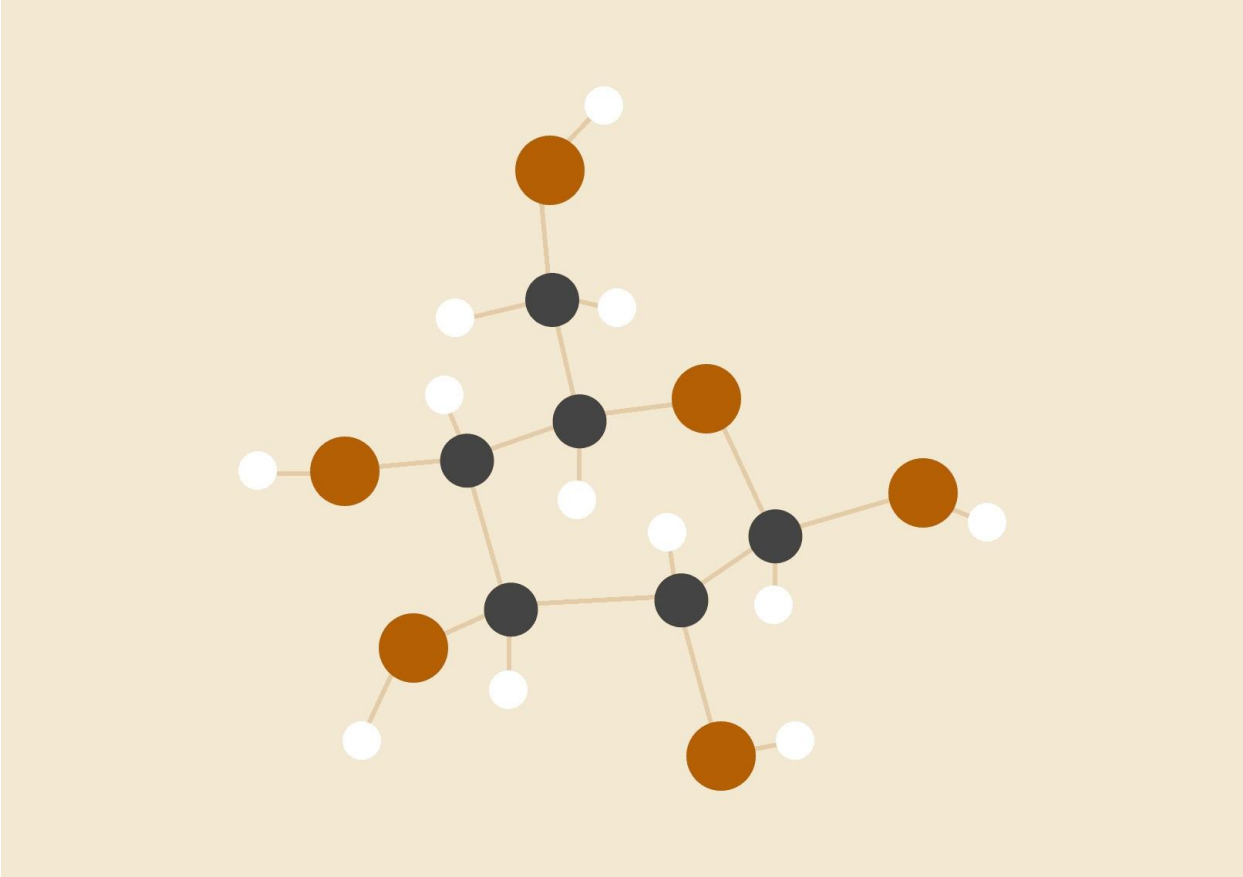
TRABAJO PRÁCTICO INTEGRADOR

IA3.5 Redes de Datos

Tecnicatura Universitaria en Inteligencia Artificial



Esteban Leon – Juan Manuel Saad – Federico Zimmer

26/06/2025

Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

INTRODUCCIÓN

El presente documento describe el diseño, desarrollo y funcionamiento de un sistema de software basado en una arquitectura de tres capas para la gestión de datos de los Premios Nobel. El sistema se compone de un cliente de consola, un servidor intermedio que actúa como puerta de enlace y un servidor final que gestiona la lógica del negocio y la persistencia de los datos.

La base de datos elegida es un archivo en formato **JSON** que actúa como un sistema de almacenamiento de documentos. Este archivo se inicializa a partir de la API pública oficial de los Premios Nobel y posteriormente es gestionado por el servidor final de nuestra aplicación.

El desarrollo y las pruebas se realizaron sobre un sistema operativo Windows, utilizando un entorno virtual de Python (venv) para aislar las dependencias del proyecto y garantizar la réplica del entorno de trabajo.

DESCRIPCIÓN DEL ENTORNO DE TRABAJO DEL SERVIDOR FINAL

Este servidor representa el núcleo de la aplicación, manejando la lógica del negocio y la interacción directa con la fuente de datos.

- Lenguaje y Framework: Se utilizó Python 3.10 junto con el framework FastAPI para la creación de las API's correspondientes.

- Descripción: Su responsabilidad principal es gestionar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre el archivo bd.json, que funciona como nuestra base de datos. Una característica clave es que, al iniciarse, el servidor verifica la existencia de este archivo. Si no lo encuentra, lo descarga y lo crea a partir de la API oficial de los Premios Nobel, asegurando que la aplicación siempre tenga un estado inicial de datos con el que trabajar.

- **Configuraciones**:

  - **Gestión de Datos**: Se emplean modelos de Pydantic (Prize, Laureate, etc.) para garantizar que todos los datos que entran y salen de la API tengan una estructura consistente y válida.

  - **Autenticación y Autorización**: Implementa un sistema de autenticación básica HTTP con dos roles definidos: lector (con permisos de solo lectura) y admin (con permisos completos para todas las operaciones).

  - **Persistencia**: Cualquier cambio realizado a través de los endpoints (POST, PUT, DELETE) se escribe y persiste directamente en el archivo bd.json.

  - **Control**: Se implementó un limitador de solicitudes, restringiendo a cada IP en un máximo de 5 solicitudes por segundo mediante un middleware. Con esto prevenimos abusos y mejoramos la estabilidad de la API.

- **Problemas Encontrados**:

  El principal desafío técnico fue diseñar una lógica sólida para la asignación de IDs únicos a los nuevos laureados (Laureate) que se agregaban. La solución fue implementar una función (get_max_laureate_id) que escanea todos los premios y laureados existentes para encontrar el ID más alto y asignar el siguiente valor correlativo, evitando así colisiones de identificadores.

Otro desafío fue comprender el funcionamiento de Pydantic para implementar modelos de datos consistentes y, de esta forma, evitar la redundancia en el código.

DESCRIPCIÓN DEL ENTORNO DE TRABAJO DEL SERVIDOR INTERMEDIO

Este servidor funciona como intermediario, un punto de entrada único y seguro para el cliente, abstrayendo la complejidad del backend.

- Lenguaje y Framework: Al igual que el servidor final, se desarrolló en Python 3.10 con FastAPI.

- Descripción: Su función principal es actuar como una capa de seguridad y enrutamiento. No contiene lógica de negocio; recibe las peticiones del cliente, autentica al usuario, verifica que su rol (admin o user) tenga los permisos necesarios para el endpoint solicitado y, en caso de ser necesario, reenvía la petición al servidor final utilizando las credenciales correspondientes.

- **Configuraciones**:

  - **Enrutamiento y Abstracción:** Mapea las rutas expuestas al cliente con las del servidor final, ocultando la dirección y estructura interna del backend. Esto permite, por ejemplo, cambiar la ubicación del servidor final sin necesidad de modificar el cliente.

  - **Seguridad Centralizada:** Gestiona de forma centralizada los usuarios y sus roles. Traduce los roles del cliente (admin, user) a los roles que entiende el servidor final (admin, lector), proveyendo un control de acceso unificado.

- **Control**: Se implementó un limitador de solicitudes, restringiendo a cada IP en un máximo de 5 solicitudes por segundo mediante un middleware. Con esto prevenimos abusos y mejoramos la estabilidad de la API.

- **Problemas Encontrados**:

Un punto crítico durante el desarrollo fue asegurar que las credenciales para la comunicación entre el servidor intermedio y el servidor final se gestionaran de forma segura. Se utilizó la librería requests junto con HTTPBasicAuth para garantizar que cada petición reenviada estuviera correctamente autenticada, evitando el acceso no autorizado al servidor de datos.

DESCRIPCIÓN DEL ENTORNO DE TRABAJO DEL CLIENTE

- **Lenguaje y Librerías**: Python 3.10, utilizando principalmente la librería requests para realizar las comunicaciones HTTP con el servidor intermedio.

- **Descripción**: Se trata de una aplicación de línea de comandos (CLI) que proporciona un menú interactivo y fácil de usar. A través de este menú, el usuario puede invocar todas las funcionalidades de la API de forma intuitiva.

- **Configuraciones**:

- **URL Base**: La dirección del servidor intermedio está definida como una constante para facilitar su modificación si fuera necesario.

- **Autenticación:** Emplea HTTPBasicAuth de la librería requests para adjuntar automáticamente las credenciales de admin o user en las cabeceras de las operaciones que requieren privilegios.

- **Interfaz de Usuario**: Presenta un menú numerado y funciones modulares para cada operación, solicitando al usuario los datos requeridos por consola.

- **Problemas Encontrados**:

El desafío principal fue la gestión de la entrada del usuario, especialmente para la creación de un premio con múltiples laureados. Esto se solucionó implementando un bucle while que permite al usuario agregar tantos laureados como desee de forma secuencial antes de enviar la petición final.

Otro desafío importante fue desarrollar una interfaz gráfica sencilla e intuitiva, que permitiera al usuario interactuar de forma eficaz con los distintos métodos de la API.

DESCRIPCIÓN DE LA BASE DE DATOS ELEGIDA

Se utiliza un único archivo, bd.json, como base de datos. El objeto raíz del archivo JSON es un diccionario con una única clave: "prizes".

- "prizes": Es una lista (array) de objetos, donde cada objeto representa un premio.

Estructura del Objeto Premio (Prize):

Propiedad	Tipo de Dato	Opcional	Descripción

year	integer	No	Año en que se otorgó el premio.
category	string	No	Categoría del premio (ej. "physics", "literature").
overallMotivation	string	Sí	Una motivación general para premios compartidos.
laureates	Lista de Objetos	No	Una lista que contiene a los laureados de ese premio.

Estructura del Objeto Laureado (Laureate):

Propiedad	**Tipo de Dato**	**Opcional**	**Descripción**
id	integer	Sí	Identificador único para cada laureado.
firstname	string	Sí	Nombre del laureado.

surname	string	Sí	Apellido del laureado.
motivation	string	Sí	La razón específica por la que se le otorgó el premio.
share	string	Sí	La fracción del premio que recibió (ej. "1", "2").

PROCEDIMIENTO

A continuación, se describen los endpoints disponibles y cómo interactuar con ellos.

Credenciales de ejemplo:

- **Admin**: usuario: admin, contraseña: admin123
- **Usuario normal**: usuario: user, contraseña: user123

1. Obtener todos los premios (acceso público)

- **Método**: GET
- **Endpoint**: /

- **Descripción**: Devuelve un listado de todos los premios, mostrando solo año y categoría. No requiere autenticación.

- **Ejemplo curl**:
  curl -X GET http://X.X.X.X:8000/

2. Buscar premios por año (requiere autenticación)

- **Método**: GET

- **Endpoint**: /prizes/{year}

- **Descripción**: Devuelve todos los premios de un año específico.

- **Ejemplo curl**:
  curl -X GET http://X.X.X.X:8000/prizes/2023 -u "user:user123"

3. Buscar premio por año y categoría (requiere autenticación)

- **Método**: GET

- **Endpoint**: /prizes/{year}/{category}

- **Descripción**: Devuelve el detalle completo de un premio, incluyendo sus laureados.

- **Ejemplo curl**:
  curl -X GET http://X.X.X.X:8000/prizes/2023/physics -u "user:user123"

4. Agregar un nuevo premio (solo admin)

- **Método**: POST

- **Endpoint**: /prize

- **Descripción**: Crea un nuevo premio. El cuerpo de la petición debe ser un JSON con la estructura del modelo Prize.

- **Ejemplo curl**:

```
curl -X POST http://X.X.X.X:8000/prize \
-u "admin:admin123" \
-H "Content-Type: application/json" \
-d '{
  "year": 2024,
  "category": "computer science",
  "laureates": [
    {
      "firstname": "John",
      "surname": "Doe",
      "motivation": "For creating amazing APIs.",
      "share": "1"
    }
  ]
}'
```

5. Modificar un premio/laureado (solo admin)

- **Método**: PUT

- **Endpoint**: /prizes/{year}/{category}

- **Descripción**: Actualiza la información de un premio o de sus laureados.

- **Ejemplo curl** (modificar el nombre de un laureado con ID 1025):

```
curl -X PUT http://X.X.X.X:8000/prizes/2023/chemistry \
-u "admin:admin123" \
-H "Content-Type: application/json" \
-d '{
   "laureates": [
      {
         "id": 1025,
         "firstname": "Jane"
      }
   ]
}'
```

6. Eliminar un premio (solo admin)

- **Método**: DELETE

- **Endpoint**: /prizes/{year}/{category}

- **Descripción**: Elimina un premio completo.

Ejemplo curl:

```
curl -X DELETE http://X.X.X.X:8000/prizes/2024/computer%20science -u
"admin:admin123"
```

INSTRUCCIONES DE USO DEL CLIENTE

1. Requisitos

Para la instalación y ejecución del software cliente, verifique los siguientes puntos:

- **Software:** Intérprete de Python, versión 3.10 o superior.
- **Red:** Acceso a la subred donde se localiza el servidor intermedio.

2. Instalación

El proceso de instalación se centra en la configuración del entorno y la instalación de las bibliotecas necesarias.

1. **Entorno Virtual:** Es una buena práctica crear un entorno virtual para aislar el proyecto.

   Script: python3 -m venv venv

   Ejecución: - Windows: .\\venv\\Scripts\\activate – Linux source venv/bin/activate

2. **Instalar Dependencias:** Desde una terminal y dentro del entorno virtual, ejecute el comando que instalará todos los paquetes requeridos definidos en requirements.txt: ***pip install -r requirements.txt***

3. Configuración

La comunicación con el servidor requiere una configuración previa:

1. Localice los archivos de configuración del proyecto.

2. Defina en ellos la dirección IP y el número de puerto asignados al servidor intermedio.

4. Ejecución e Interacción

1. **Inicio:** Lance la aplicación ejecutando el script principal desde la terminal: ***python cliente.py***

5. Uso del Menú Interactivo

Al ejecutar el cliente, se presentará el siguiente menú:

--- MENÚ API PREMIOS ---
1. Ver todos los premios
2. Buscar por año
3. Buscar por año y categoría
4. Agregar nuevo premio
5. Modificar laureado
6. Eliminar premio
0. Salir

- **Opción 1**: Muestra una lista simple de todos los premios. No necesita credenciales.

- **Opción 2**: Pide un año y muestra los premios correspondientes. Necesita credenciales de user o admin. El cliente usa admin por defecto.

- **Opción 3**: Pide un año y categoría para mostrar el detalle completo. Necesita credenciales de user o admin.

- **Opción 4**: Guía al usuario para crear un nuevo premio, pidiendo año, categoría y los datos de uno o más laureados. **Requiere permisos de administrador**.

- **Opción 5**: Permite modificar los datos de un laureado existente. Pide el año y categoría del premio, y el ID del laureado a cambiar. **Requiere permisos de administrador**.

- **Opción 6**: Pide un año y categoría para eliminar el premio correspondiente. **Requiere permisos de administrador**.

- **Opción 0**: Termina la ejecución del cliente.

REFERENCIAS

1. FastAPI: https://fastapi.tiangolo.com/
2. Pydantic: https://docs.pydantic.dev/2.11/