Manual técnico - Interfaz gráfica ANNarchy

1. Introducción

Este manual técnico documenta el sistema desarrollado como una interfaz gráfica interactiva para el neurosimulador ANNarchy. Su propósito es proporcionar al personal encargado de usar y/o mantener el sistema la información necesaria para comprender, desplegar, utilizar y mantener la herramienta.

El sistema permite definir, construir, simular y analizar redes neuronales mediante una interfaz visual basada en web. Está especialmente diseñado para ser utilizado en redes locales, como laboratorios de investigación o salas de clase, facilitando el acceso a simulaciones neuronales avanzadas sin requerir conocimientos profundos en programación ni software específico.

2. Descripción General del Sistema

El sistema ofrece una interfaz visual para construir redes neuronales con ANNarchy, permitiendo:

- Definición de modelos de neuronas y sinapsis.
- Creación gráfica de redes neuronales.
- Configuración de monitores.
- Generación automática del código ANNarchy correspondiente.
- Ejecución de simulaciones.
- Visualización de resultados en gráficos interactivos.

Todo esto es posible mediante una arquitectura cliente-servidor desplegada con contenedores Docker.

3. Tecnologías Utilizadas

Frontend (Cliente)

- **React**: Implementa el editor visual de redes, paneles de configuración, vista de resultados y generación de gráficos.

Backend (Servidor)

- **Flask (Python)**: Implementa el backend para procesar las solicitudes de simulación del cliente.
- **ANNarchy**: Motor encargado de ejecutar las simulaciones.

Despliegue

- **Docker**: Proporciona un entorno de ejecución consistente y reproducible que incluye:
    - Sistema operativo Linux.
    - Versiones específicas de Python, ANNarchy y dependencias.
    - Contenedores separados para frontend y backend.

4. Arquitectura del Sistema

El sistema se organiza bajo una arquitectura **cliente-servidor**, donde el cliente corresponde a la interfaz gráfica accesible mediante navegador web, y el servidor ejecuta las simulaciones y gestiona los datos. Esta sección presenta tres diagramas que describen la estructura lógica, física y de datos del sistema: componentes, despliegue y clases.

4.1 Diagrama de Componentes

El diagrama de componentes representa la organización lógica de los principales módulos funcionales del sistema. Se estructura en torno a un **paquete principal** que agrupa los componentes del cliente, y se extiende hacia el backend como componente externo.

- **Interfaz de usuario**: Componente central del paquete, encargado de gestionar la construcción gráfica de redes neuronales, configuración de parámetros, simulación y visualización de resultados.

- **Generador de gráficos**: Componente consumido por la interfaz para representar los resultados de las simulaciones en formato visual.

- **Generador de código**: Componente que transforma la red construida en código ANNarchy válido; también es consumido por la interfaz.

- **Backend**: Componente externo al paquete, consumido por el generador de código para enviar solicitudes de simulación y recibir resultados.

- **Usuario**: Representado como consumidor directo de la interfaz de usuario, a través del navegador.

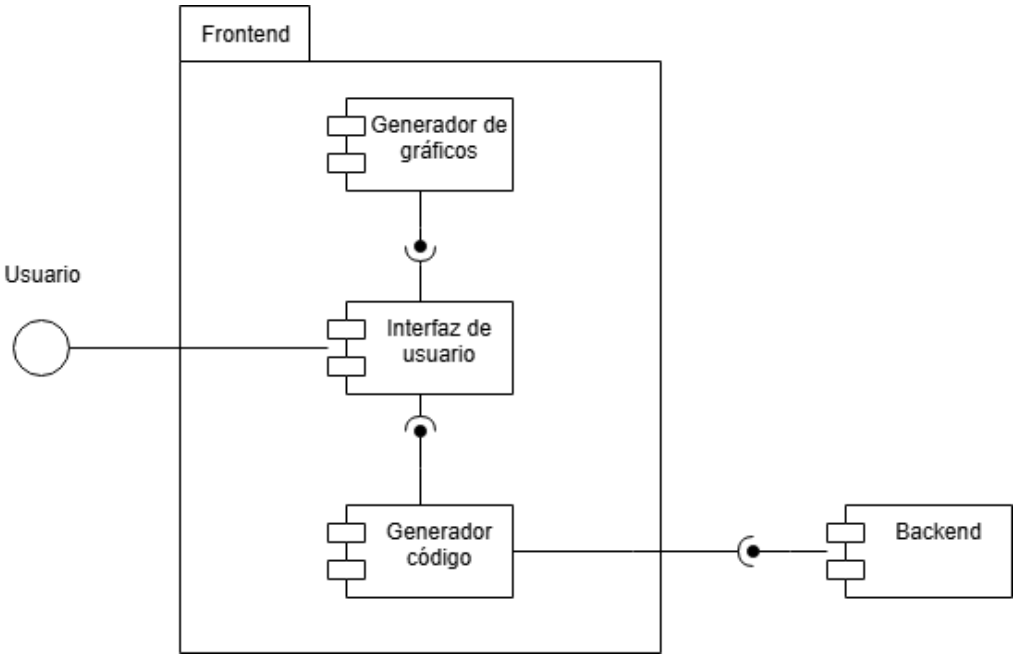


Diagrama de componentes

4.2 Diagrama de Despliegue

El diagrama de despliegue ilustra la distribución física del sistema en una red local, destacando los nodos involucrados y los artefactos desplegados en cada uno.

- **Nodo Servidor**: Contiene los contenedores Docker del **frontend** (interfaz cliente) y el **backend** (servidor Flask con ANNarchy).

- **Nodo Estudiante (cliente)**: Representado por un equipo que ejecuta un **navegador web**, el cual se conecta a la interfaz alojada en el servidor.

- **Comunicación**: Una flecha de asociación representa la interacción entre el navegador del estudiante y la interfaz en el servidor, estableciendo la conexión cliente-servidor sobre la red local.

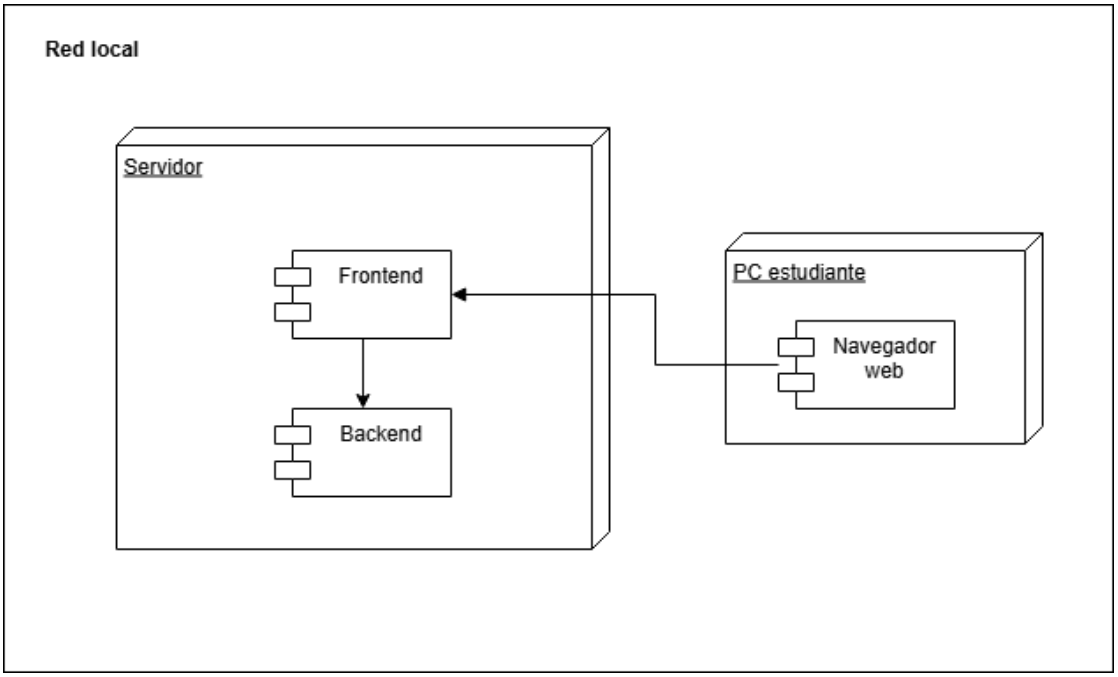


Diagrama de despliegue

4.3 Diagrama de Clases

El diagrama de clases modela las principales estructuras de datos empleadas internamente para representar las redes neuronales y sus componentes. Aunque el sistema no utiliza orientación a objetos de forma explícita, el diagrama describe las abstracciones implementadas.

- **Neurona**: Unidad base para la definición de poblaciones.
- **Población**: Requiere una neurona y puede tener asociado un **monitor** para registrar su actividad.
- **Monitor**: Elemento opcional vinculado a una población para obtener datos de simulación.
- **Proyección**: Puede estar asociada a una población; requiere una **conexión** y puede incluir una **sinapsis**.
- **Conexión**: Elemento obligatorio en toda proyección, define la dirección del flujo y los parámetros estructurales.

- **Sinapsis**: Componente opcional que define propiedades específicas del modelo sináptico utilizado.

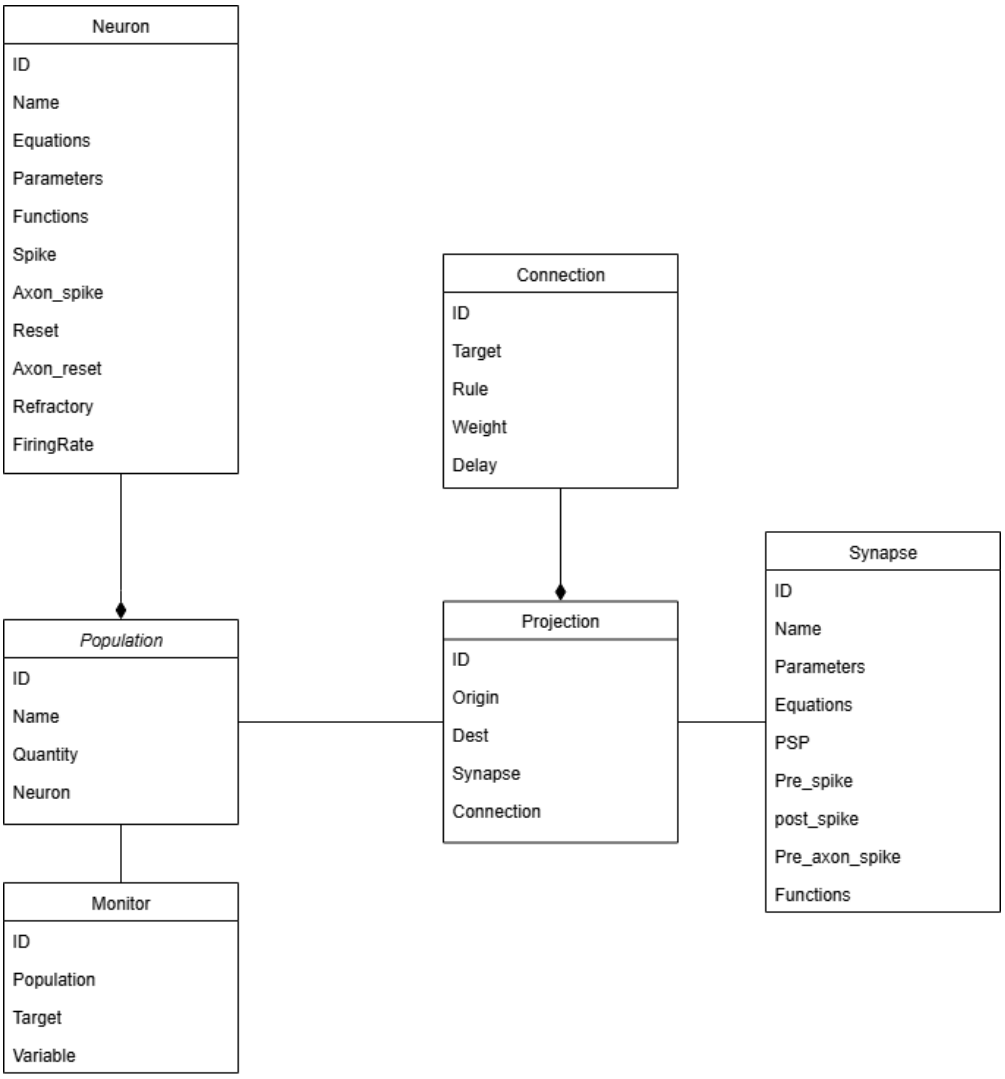


Diagrama de clases

5. Requisitos del Sistema

Este capítulo describe los requisitos técnicos necesarios para ejecutar correctamente el sistema, tanto en el entorno del cliente como del servidor.

5.1 Requisitos del cliente

El cliente accede al sistema a través de un navegador web. Para su correcto funcionamiento, se recomienda:

- **Navegador compatible**: Chrome, Firefox, Edge u Opera, en versiones recientes.
- **Conexión a red local**: Acceso estable a la red donde está alojado el servidor.

No se requiere instalación adicional en el equipo del usuario.

5.2 Requisitos del servidor

El servidor aloja el backend y el frontend del sistema dentro de contenedores Docker. Se recomienda cumplir con las siguientes especificaciones:

- **Sistema operativo**: Linux (Ubuntu 20.04 o superior).
- **Docker instalado**.
- **Recursos mínimos estimados**:
    - CPU: Procesador de 4 núcleos.
    - RAM: 8 GB.
    - Espacio en disco: 10 GB disponibles (Almacenar el sistema y las simulaciones).

Además, se debe habilitar el acceso a los puertos utilizados por los contenedores (por defecto, 5000 para backend y 3000 para frontend).

6. Despliegue del Sistema

Esta sección describe el procedimiento para instalar y desplegar el sistema en un entorno local utilizando Docker, asegurando que tanto el frontend como el backend estén correctamente configurados y accesibles.

6.1 Requisitos previos

- Tener instalado Docker en el servidor.
- Conexión a Internet para descargar las imágenes desde Docker Hub.

6.2 Descarga de imágenes

- Las imágenes se obtienen desde DockerHub utilizando los siguientes comandos:

```
docker pull nachowo/front
docker pull nachowo/back
```

6.3 Ejecución de contenedores

- Para ejecutar los contenedores de forma independiente, se pueden usar:

```
docker run -d -p 3000:3000 --name annarchy_front nachowo/front
docker run -d -p 5000:5000 --name annarchy_back nachowo/back
```

Si desea cambiar los puertos expuesto debe cambiar el primer puerto de la siguiente manera:

```
docker run -d -p XXXX:3000 --name annarchy_front nachowo/front
docker run -d -p XXXX:5000 --name annarchy_back nachowo/back
```

6.4 Configuración de red

- Asegurarse de que el servidor tenga configurada correctamente la red local para permitir conexiones entrantes a los puertos indicados expuestos.
- Configurar firewall o reglas de seguridad para permitir el tráfico en dichos puertos.

- En el caso de que el servidor esté detrás de un router puede ser necesario configurar el net forwarding para permitir el acceso desde redes externas.

6.4.1 Abrir puertos en Windows (Opcional)

Para abrir los puertos en Windows Firewall hay que seguir los siguientes pasos:

1. Abrir el **Panel de control** y navegar a **Sistema y seguridad > Firewall de Windows Defender > Configuración avanzada**.

2. En la ventana de **Firewall de Windows Defender con seguridad avanzada**, seleccionar **Reglas de entrada** en el panel izquierdo.

3. Hacer clic en **Nueva regla** en el panel derecho.

4. Seleccionar **Puerto** y hacer clic en **Siguiente**.

5. Elegir **TCP** y especificar los puertos locales de ambos servicios (3000 y 5000 predeterminado) separados por comas.

6. Seleccionar **Permitir la conexión** y continuar.

7. Elegir los perfiles a aplicar (Dominio, Privado, Público) según corresponda y continuar.

8. Asignar un nombre a la regla, por ejemplo, "Puertos ANNarchy" y finalizar.

6.4.2 Abrir puertos en Ubuntu (Opcional)

Para abrir los puertos en un servidor ubuntu es necesario ejecutar los siguientes comandos:

```
sudo ufw allow 3000/tcp
sudo ufw allow 5000/tcp
sudo ufw reload
```

6.5 Acceso al sistema

Desde un equipo cliente en la misma red, abrir un navegador y acceder a:

http://<IP_DEL_SERVIDOR>:<PUERTO_FRONTEND>

Para averiguar la ip del servidor hay que ejecutar el comando ipconfig en Windows o ip addr show en ubuntu.

7. Uso del Sistema

Esta sección describe de forma general el flujo funcional de la interfaz gráfica del neurosimulador ANNarchy.

- El sistema es accesible mediante navegador web desde equipos en la red local, ingresando la dirección IP y puerto del servidor donde está alojado el frontend.

- Los usuarios pueden crear redes neuronales utilizando un entorno gráfico que permite agregar y configurar elementos como neuronas, poblaciones, sinapsis y monitores.

- El sistema genera automáticamente código ANNarchy a partir del diseño realizado, que es enviado al backend para ejecutar simulaciones en paralelo.

- Los resultados de las simulaciones se reciben y presentan en la interfaz a través de gráficos interactivos para su análisis.

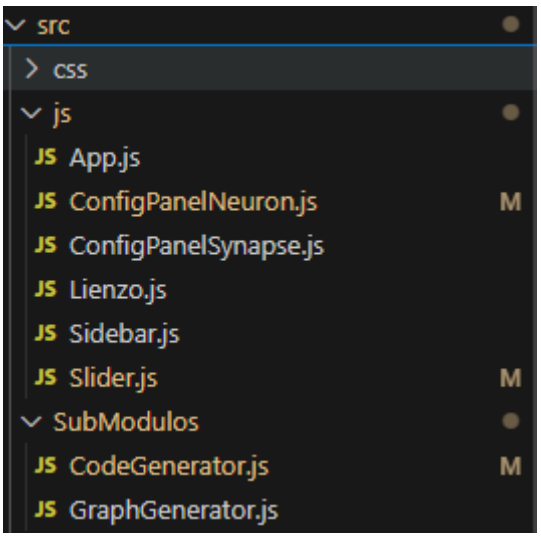
- La interfaz incluye funcionalidades para modificar parámetros, reiniciar simulaciones y exportar código generado.

8. Código fuente

El proyecto está dividido principalmente en dos grandes módulos: frontend y backend, cada uno con funciones y responsabilidades definidas, el código se encuentra en GitHub en el siguiente Repositorio.

8.1 Frontend

Desarrollado en React, se divide en 3 directorios que contienen los componentes de la interfaz gráfica, los estilos y los módulos de apoyo para el sistema.



Archivos frontend

8.1.1 JS

El primer directorio del frontend es la carpeta JS, esta carpeta contiene los componentes gráficos de la interfaz, estos son:

- **APP:** Componente principal, es el que invoca al resto. Tiene los estados para el tiempo de simulación y el paso de tiempo, también hace uso del módulo de **CodeGenerator** para hacer la solicitud de simulación.

- **Lienzo:** Este componente es el campo de trabajo del usuario, aquí es donde se posicionan las poblaciones y conexiones para crear la red neuronal, presenta las funciones para manejar la gestión de los elementos en este mismo, como el añadir, mover y eliminar. También presenta estados para todos los elementos que componen la red como son las poblaciones, sinapsis y monitores.

- **Sidebar:** Este componente es un menú lateral en donde se encuentran 2 pestañas, **Options** y **Code.** Dentro de la primera pestaña se encuentran los listados de neuronas y sinapsis, además de los botones para crear nuevos modelos para estas listas. La segunda pestaña hace uso del módulo **CodeGenerator** para mostrar el código de la simulación junto con un botón para descargarlo. Este archivo presenta la lógica de administración de los modelos y tiene los estados para los modelos predeterminados y personalizados.

- **ConfigPanelNeuron:** Este componentes es el panel usado para gestionar los parámetros de las neuronas y poblaciones, en el caso de las poblaciones incluye lo necesario para gestionar los monitores, hace uso de **GraphGenerator** para renderizar los gráficos.

- **ConfigPanelSynapse:** Este componente es el panel usado para el control de las sinapsis, se abre al hacer clic derecho sobre una conexión del lienzo, tiene 2 pestañas para gestionar la conexión y sinapsis.

8.1.2 SubModulos

Este directorio contiene los 2 submódulos de la interfaz :

- **CodeGenerator:** Este módulo es el encargado de crear el código de la simulación para la red descrita por el usuario en el lienzo. Además, presenta las funciones para hacer uso de los endpoints del servidor para solicitar las simulaciones y ver el estado de estas.

- **GraphGenerator:** Este módulo es el encargado de procesar la información resultante de las simulaciones para adaptar los datos y renderizar los gráficos de los monitores.

8.2 Backend

El backend consta de un único archivo, este es app.py. Su función es recibir, ejecutar y gestionar simulaciones enviadas por la interfaz. Este crea varios hilos para procesar las simulaciones mientras que el hilo original gestiona las solicitudes de simulación y estado del trabajo, para esto tiene 2 endpoints, **Simulate** y **Status**.

El primer endpoint extrae el código desde la solicitud, pone en una cola global el código a ejecutar y genera un identificador para este trabajo que retorna al cliente.

Luego un hilo trabajador toma el código, ejecuta la simulación con un subproceso y toma los resultados desde el standard output, estos resultados son guardados en una cola de trabajos completados. Luego, mientras el cliente hace polling al endpoint **Status** con el id del trabajo, se le responde con el estado de este, “En espera” significa que está en la cola esperando por ser ejecutado y “En ejecución” significa que está siendo procesado, en el caso de que la simulación haya terminado y los resultados estén listos, en lugar de entregarle un estado se le retornan estos resultados y se eliminan del backend.