



FACULTAD DE INGENIERIAS FISICOMECHANICAS.

ESCUELA DE INGENIERIA ELECTRICA, ELECTRONICA Y DE
TELECOMUNICACIONES.

GRUPO CONECTIVIDAD Y PROCESAMIENTO DE SEÑALES CPS

PREGRADO EN INGENIERIA ELECTRONICA

DMDL-PC-SH-UCI

TESIS DE PREGRADO

Author
David PEREZ

Supervisor
Dr. Carlos FAJARDO

Cosupervisor
Dr. Said Pertuz

ACADEMIC YEAR 2024
Add your graduation date here

David PEREZ

DMDL-PC-SH-UCI

Tesis de pregrado

Universidad Industrial de Santander

July 2024

“What we know is a drop, what we don’t know is an ocean.”

Isaac Newton

Contents

1	Acerca de	1
1.1	Introducción	1
1.2	Contenido	2
1.2.1	Objetivos	2
1.2.2	Metodologia	2
2	Infra-estructura de la solucion	3
2.1	Consideraciones generales	3
2.1.1	Formato de salida	5
2.1.2	Frameworks	5
2.1.3	Dependencias	6
2.2	Endpoints y servicios	6
2.2.1	Default	6
2.2.2	Inferencia	6
2.2.3	Creacion de usuario	8
2.2.4	Generador de sesion	9
2.3	Middleware y logs de ejecucion	10
2.3.1	El archivo <code>app.log</code>	10
2.4	Artefactos	11
2.4.1	El archivo <code>main.py</code>	11
2.4.2	El archivo <code>logger.py</code>	13
2.4.3	El archivo <code>predict.py</code>	15
2.4.4	El archivo <code>auth.py</code>	17
2.4.5	El archivo <code>database.py</code>	20
2.4.6	El archivo <code>models.py</code>	21
2.4.7	El archivo <code>middleware.py</code>	22
3	Despliegue de oepraciones	25
3.0.1	Ventajas de usar Docker	25
3.0.2	Contras de usar Docker	25
3.1	El archivo <code>Dockerfile</code>	26
3.2	El archivo <code>compose.yaml</code>	27

4 Casos de usuario	29
4.1 Servidor	30
4.1.1 Jerarquia de directorios	32
4.2 Desarrollo en vivo	33
4.3 Cliente	33
4.3.1 El archivo <code>client.py</code>	33
A Frequently Asked Questions	35
A.1 How do I change the colors of links?	35
Acknowledgements	37

Chapter 1

Acerca de

1.1 Introducción

El shock hemodinámico es una condición clínica severa caracterizada por una disminución sostenida en la perfusión de tejidos, lo que lleva a una entrega insuficiente de oxígeno y nutrientes a tejidos y órganos vitales. Varias causas subyacen a este fenómeno, lo que ha llevado a la necesidad de diversas formas de clasificación. Entre las más comunes se encuentran el shock hipovolémico, cardiogénico, obstructivo y distributivo[3]. En el contexto de la salud global, se estima que más personas mueren cada año por enfermedades cardiovasculares que por cualquier otra causa, según datos recientes de la Organización Mundial de la Salud (OMS) y la Organización Panamericana de la Salud (OPS). Esta imagen sombría incluye eventos relacionados con el shock hemodinámico y destaca la magnitud del desafío que este síndrome representa en la atención médica moderna. Por lo tanto, subraya la necesidad urgente de investigaciones exhaustivas y enfocadas en esta área para mejorar los resultados clínicos [2]. El diagnóstico temprano del shock hemodinámico es esencial para iniciar un tratamiento apropiado y mejorar el pronóstico del paciente. Sin embargo, el diagnóstico clínico temprano puede ser desafiante, especialmente en pacientes con síntomas poco comunes. A esto se suma la gestión médica simultánea de múltiples pacientes en unidades de cuidados intensivos, lo que representa un desafío para los médicos, especialmente en entornos con recursos limitados [3,5]. En los últimos años, la aplicación de algoritmos de aprendizaje automático ha revolucionado numerosos procesos en la industria, proporcionando una amplia gama de aplicaciones que van desde el reconocimiento de objetos mediante visión por computadora, reconocimiento de voz, comprensión del lenguaje natural, traducción automática, entre otros [6]. Uno de los campos que ha experimentado una transformación profunda gracias a esta tecnología es la medicina, donde se han desarrollado algoritmos que posteriormente se convierten en sistemas de apoyo a decisiones para profesionales médicos[7]. Estos algoritmos, entrenados mediante extensos conjuntos de datos clínicos, son capaces de identificar patrones que podrían facilitar

la detección temprana del shock hemodinámico, lo que puede agilizar la toma de decisiones terapéuticas y el diagnóstico en el contexto de las unidades de cuidados intensivos. El propósito de este estudio es desplegar un algoritmo de aprendizaje automático para predecir el shock hemodinámico, utilizando variables clínicas y hemodinámicas. Los hallazgos de esta investigación tienen el potencial de permitir el monitoreo continuo de los pacientes en la unidad de cuidados intensivos, permitiendo a los médicos tomar decisiones informadas basadas en la predicción y la condición del paciente, y actuar así antes de un evento de shock hemodinámico.

1.2 Contenido

La presente tesis tiene como objetivo principal abordar el despliegue de un modelo de inteligencia artificial en una máquina virtual Ubuntu utilizando contenedores Docker, mediante la implementación de FastAPI. Este enfoque proporciona un entorno flexible y escalable para la ejecución de modelos de IA, facilitando su integración en aplicaciones y sistemas existentes.

1.2.1 Objetivos

- Desarrollar un proceso de despliegue eficiente y replicable para modelos de inteligencia artificial en entornos basados en Ubuntu.
- Implementar FastAPI como interfaz de programación para la comunicación entre el modelo de IA y las aplicaciones cliente.
- Utilizar contenedores Docker para garantizar la portabilidad y el aislamiento del entorno de ejecución del modelo.

1.2.2 Metodología

Este documento busca exponer los objetivos planteados bajo las siguientes técnicas metodológicas:

1. Preparación del Entorno: Instalación y configuración de una máquina virtual Ubuntu con los requisitos necesarios para ejecutar contenedores Docker.
2. Desarrollo de una API utilizando FastAPI para exponer el modelo de IA y permitir su interacción con otras aplicaciones.
3. Creación de Contenedores Docker: Configuración de contenedores Docker para encapsular el modelo de IA y sus dependencias.
4. Despliegue y Pruebas: Implementación del modelo de IA en la máquina virtual Ubuntu y realización de pruebas para verificar su funcionamiento correcto.

Chapter 2

Infra-estructura de la solucion

2.1 Consideraciones generales

El modelo de Deep Learning expuesto funcionara tomando como paramteros de entrada la informacion historica hemodinamica de cada paciente por un rango de 1 hora con 48 minutos, esto quiere decir que para realizar una sesion de inferencia dado un paciente se necesitara la informacion hemodinamica del mismo por los ultimos 108 minutos. El grupo de investigacion en conectividad y procesamiento de señales CPS exporto los siguientes artefactos para establecer el acuerdo de servicio por el cual se define el formato de entrada al modelo de deep learning, proveer la version oficial del modelo de deep learning que sera desplegada y especificar los requerimientos tecnologicos y de ambiente que se requieren para la ejecucion del modelo en materia.

Los artefactos en cuestion:

- LSTM.h5: El modelo de deep learning en formato keras.
- predict.py: Artefacto para cargar y preprocesar datos, cargar el modelo y ejecutarlo.
- requirements.txt: archivo de texto plano especificando dependencias del ambiente.

Tomando en cuenta que la frecuencia de muestreo de las señales hemodinamicas en cada paciente es de 1 cada 5 minutos se prevee que entre cada sesion de inferencia se de este mismo periodo de tiempo. Asumiendo un flujo de pacientes alto es licito concluir que la solucion desplegada debera responder eficientemente a consultas multiples en el mismo o casi mismo instante de tiempo, es decir debera constituirse a si misma una aplicacion capaz de recibir, procesar y responder consultas multiples por uno o multiples usuarios.

Cualesquiera consultas realizadas por determinado cliente deberan incluir consigo la informacion requerida concepto de parametros de entrada para la sesion de inferencia. El formato de estos datos se constituye de la siguiente manera:

```

1 data_input = {'idAtencion':int,
2               'idSigno':int,
3               'nomSigno':str,
4               'valor':float,
5               'fecRegistro':str
6               }

```

El artefacto predict.py sugiere que la forma final en la que deberan ser ingestados los datos proviene de las columnas nomSigno que es la que contiene el nombre de las variables hemodinamicas y la columna valor que es la que contiene la informacion numerica dada la variable hemodinamica. El formato de los datos necesario es tal que estas variables ahora sean columnas independientes cada una con su respectivo nombre y valor cuando sea el caso y cuando no consignar un NaN para posterior filtrado tal que solo quedaran registros efectivos (no NaN) distribuidas las variables hemodinamicas en columnas.

```

1 async def predict(data):
2     # Crear DataFrame
3     df = pd.DataFrame(data)
4
5     # Reorganizar el DataFrame segun las especificaciones dadas,
6     ['idAtencion', 'fecRegistro', 'nombreSigno', 'valor']
7     df_resultante = df.pivot_table(index=['idAtencion', 'fecRegistro'],
8                                     columns='nomSigno', values='valor').reset_index()
9
10    # Mostrar el DataFrame resultante
11
12    filas_con_nan = df_resultante[df_resultante.isnull().all(axis=1)]
13
14    #Se filtran los datos con mayor numero de datos no validos para realizar
15    el procesamiento
16    columnas_deseadas = ['idAtencion', 'fecRegistro', 'MDC_BLD_PERF_INDEX',
17                        'MDC_ECG_HEART_RATE', 'MDC_ECG_V_P_C_RATE',
18                        'MDC_LEN_BODY_ACTUAL', 'MDC_MASS_BODY_ACTUAL',
19                        'MDC_PULS_OXIM_PULS_RATE',
20                        'MDC_PULS_OXIM_SAT_O2', 'MDC_TEMP',
21                        'MDC_TTHOR_RESP_RATE']
22
23    # Filtrar el DataFrame para incluir solo las columnas deseadas
24    df_resultante_filtrado = df_resultante[columnas_deseadas]
25
26    # Se usan los ultimos datos del paciente, es decir la ultima hora y
27    cuarenta minutos de datos registrados
28
29    ultimas_20_filas_por_id =
30        df_resultante_filtrado.groupby('idAtencion').tail(20)
31
32    df_resultante_lleno = ultimas_20_filas_por_id.ffill()
33
34    # Caracteristicas seleccionadas
35
36    selected_features = ['MDC_BLD_PERF_INDEX', 'MDC_ECG_HEART_RATE',
37                        'MDC_ECG_V_P_C_RATE',
38                        'MDC_LEN_BODY_ACTUAL', 'MDC_MASS_BODY_ACTUAL', 'MDC_PULS_OXIM_PULS_RATE',
39                        'MDC_PULS_OXIM_SAT_O2', 'MDC_TEMP', 'MDC_TTHOR_RESP_RATE']

```

```
1 features = df_resultante_lleno[selected_features]
```

El formato de entrada que comprende el modelo de inferencia contempla un arreglo de datos que a través de sus dimensiones segmenta los registros por ID de usuario, variables hemodinámicas y cantidad de registros requeridos por el batch de inferencia (30 registros) por cada variable.

La salida del modelo de inferencia asocia una ponderación de probabilidad de 0 a 1, 0 siendo la menor de las ponderaciones (caso menos probable) y 1 constituyéndose como la probabilidad más predominante. Cada sesión de inferencia ponderará cuatro diferentes casos probables relacionados con el estado rítmico del paciente, dichos estados son **Arresto Cardíaco**, **Bajo Gasto Cardíaco**, **Sano** y **Shock Cardiogénico**.

2.1.1 Formato de salida

El servicio expuesto para la sesión de inferencia tendrá un formato de salida tal que se pueda identificar el paciente asociado a dicha sesión de inferencia, las ponderaciones producidas por la sesión de inferencia y el estado con la mayor probabilidad asociada. Esta información será expuesta como una respuesta a una consulta HTTP mediante un objeto json.

Sírvase el siguiente como ejemplar del formato de salida:

```
1 response_object = {"idAtencion": 0,  
2                   "inferences": ["string"],  
3                   "State": "string"}
```

El formato de salida propone la segmentación de resultados a nivel de usuario por lo que se le podría solicitar al servicio de inferencia realizar la inferencia bien masivamente a un número múltiple de usuarios o una inferencia singular enfocada a un solo usuario. La respuesta del servicio de inferencia sería emitida en un documento json al cliente, el cual podría disponer del mismo como lo requiera.

2.1.2 Frameworks

La implementación del servidor se constituye mediante el framework FastAPI con el que se suplen requerimientos como tasas de consulta multitudinales y eficientes, compatibilidad con el entorno y amplia documentación disponible para el desarrollo de aplicaciones basados en este framework. Se prevé con esta REST API cubrir todos los servicios que se requieran y puedan requerirse, incluyendo el servicio de inferencia.

FastAPI se desarrolla en Python, por lo que la totalidad de la app se encontraría desarrollada en este lenguaje. Además de ser desarrollada en Python FastAPI supone facilidad al desarrollador permitiéndole interactuar dinámicamente con Swagger User Interface de tal forma que se facilita el hacer pruebas con esta herramienta en la etapa de desarrollo.

Para la exposición de el servicio de inferencia se implementa lo que se define como endpoint o url de consulta, que es en sí una ruta de accionamiento a una petición POST al sistema de la API. Este endpoint solo representará el servicio de inferencia por lo que su respuesta solo será la información relacionada con la salida del mismo. FastAPI es un framework que ofrece respuestas excepcionalmente rápidas, por lo que de existir el requerimiento de diversificar los servicios ofrecidos por la API, esta misma permitiría la

extension de los mismos solo con declarar nuevos endpoints y sus logicas de funcionamiento, sin embargo FastAPI no es un framework tan versatil en sus funcionalidades por lo que incurrir en expandir aristas o características de la aplicacion podria tornarse mas complejo que en otros frameworks donde la velocidad de respuesta no es tan crucial pero si la variedad de servicios.

Para la contenerizacion de la aplicacion se establece el uso del framework Docker, software que permite la encapsulacion de ambientes en imagenes con sistemas operativos incorporados para la ejecucion de la aplicacion sobre una maquina virtual preconfigurada. Docker supone facilidad en apartados como compatibilidad y escalabilidad, ya que al virtualizar una version propia de un sistema operativo no depende del sistema operativo de la maquina donde este se disponga a funcionar solucionando de entrada el alcance de la app al ser posible desplegarla en sistemas tanto Windows, MAC o Linux. No obstante, si es mandatorio para garantizar el exito de la aplicacion contar con una version del motor de Docker en la maquina host de la app.

2.1.3 Dependencias

Las dependencias y configuracion del ambiente se proveen cubiertas al contemplar el despliegue de la aplicacion como un contenedor Docker, sin embargo es importante aclarar que para el correcto funcionamiento de Docker se hace necesario contar con optimas cantidades de memoria RAM ya que este funciona basado en una maquina virtual. Tambien es mandatorio garantizar el acceso a este recurso mediante conexiones locales o de red local, en otras palabras, resolver la relacion cliente servidor entorno a identidades y metodo de autenticacion.

Como se ha mencionado previamente, otro de los requerimientos para el correcto funcionamiento es proveer al host con el motor Docker, por lo que se hace necesaria la instalacion del mismo antes de suponer cualquier despliegue.

2.2 Endpoints y servicios

2.2.1 Default

Este servicio informa al usuario con un reporte del estado de la app y de la version del modelo de inferencia alojado en ella. Esta definido como una peticion de tipo GET que debera estar compuesta por la url del server y la direccion del endpoint en cuestion. Asumiendo un despliegue en el puerto 0000 , el endpoint por defecto se compondria ***http://127.0.0.1:0000/*** y su funcionalidad principal radica en generar el log inicial de ejecucion detallando la version de la app en uso.

2.2.2 Inferencia

El servicio de inferencia aloja como tal el modelo de inferencia y se accede bajo una peticion POST indicando un payload o body request en la consulta, lo cual se conformaria como la informacion de entrada para el modelo de tal forma que se pueda llevar a cabo la sesion de inferencia. El endpoint asociado a este servicio se define como ***http://127.0.0.1:0000/predict*** y responde a la solicitud con el formato de salida de datos ya expuesto.

El endpoint ***/predict*** permite realizar inferencias sobre el estado probable de un paciente en el futuro con respecto a eventos de estados cardíacos.

Este endpoint recibe datos sobre signos vitales y otros parámetros médicos, y devuelve las probabilidades de diferentes estados de salud.

Método HTTP POST

Ruta /predict

Autenticación Este endpoint requiere autenticación JWT. Si el usuario no está autenticado, se devolverá un código de estado 401 con el mensaje *Authentication Failed*.

Parámetros de Entrada El cuerpo de la solicitud debe ser un objeto JSON con la siguiente estructura:

```
1 {  
2   "idAtencion": int,  
3   "idSigno": dict,  
4   "nomSigno": dict,  
5   "valor": dict,  
6   "fecRegistro": dict  
7 }
```

- **idAtencion:** Identificador de la atención del paciente.
- **idSigno:** Diccionario con los identificadores de los signos.
- **nomSigno:** Diccionario con los nombres de los signos.
- **valor:** Diccionario con los valores de los signos.
- **fecRegistro:** Diccionario con las fechas de registro de los signos.

Respuesta La respuesta será un objeto JSON con la siguiente estructura:

```
1 {  
2   "idAtencion": int,  
3   "inferences": {  
4     "Arresto_Cardiaco": float,  
5     "Bajo_Gasto_Cardiaco": float,  
6     "Sano": float,  
7     "Shock_Cardiogenico": float  
8   },  
9   "State": str  
10 }
```

- **idAtencion:** Identificador de la atención del paciente.
- **inferences:** Diccionario con las probabilidades de los diferentes estados de salud:
 - **Arresto_Cardiaco:** Probabilidad de arresto cardíaco.
 - **Bajo_Gasto_Cardiaco:** Probabilidad de bajo gasto cardíaco.
 - **Sano:** Probabilidad de estar sano.
 - **Shock_Cardiogenico:** Probabilidad de shock cardiogénico.
- **State:** Estado único inferido del paciente.

Códigos de Estado

- **200 OK:** Solicitud exitosa. Devuelve las inferencias sobre el estado de salud del paciente.
- **401 Unauthorized:** Autenticación fallida.

2.2.3 Creación de usuario

```
{@router.post("/create_user", status_code=status.HTTP_201_CREATED)}
```

Está diseñado para crear un nuevo usuario en la base de datos. A continuación se presenta una descripción detallada de su funcionalidad:

Funcionalidad del Endpoint

- **Ruta y Método:**
 - **Ruta:** `/create_user`
 - **Método HTTP:** POST
 - **Código de Estado:** 201 Created (indica que la solicitud ha tenido éxito y se ha creado un nuevo recurso)
- **Entrada:**
 - **Dependencia de la Base de Datos (db):** El endpoint recibe una dependencia de base de datos inyectada, la cual se utiliza para interactuar con la base de datos.
 - **Solicitud de Creación de Usuario (`create_user_request`):** Este es un objeto que contiene la información necesaria para crear un nuevo usuario. En particular, incluye:
 - * **username:** El nombre de usuario del nuevo usuario.
 - * **password:** La contraseña del nuevo usuario.
- **Proceso:**
 - **Hash de la Contraseña:** Se utiliza `bcrypt_context.hash` para convertir la contraseña proporcionada en un hash seguro antes de almacenarla en la base de datos.
 - **Creación del Modelo de Usuario:** Se crea una instancia del modelo de usuario (`Users`) con el `username` y el `hashed_password` generados.
 - **Agregar a la Base de Datos:** El modelo de usuario creado se agrega a la sesión de la base de datos mediante `db.add(create_user_model)`.
 - **Guardar los Cambios:** Se confirman los cambios en la base de datos mediante `db.commit()`.
- **Salida:**
 - El endpoint no devuelve ningún contenido en el cuerpo de la respuesta, pero el código de estado 201 indica que la creación del usuario fue exitosa.

Ejemplo de Solicitud

Una solicitud típica a este endpoint podría verse de la siguiente manera:

Solicitud HTTP POST:

```
POST /create_user
Content-Type: application/json
```

```
{
  "username": "nuevo_usuario",
  "password": "contraseña_segura"
}
```

Respuesta HTTP:

HTTP/1.1 201 Created

El endpoint `create_user` se encarga de recibir una solicitud para crear un nuevo usuario, hashear la contraseña de manera segura, crear un modelo de usuario con los datos proporcionados, agregar este modelo a la base de datos y confirmar los cambios, asegurando así que el nuevo usuario se almacene de manera segura y eficiente.

2.2.4 Generador de sesion

Descripción del Endpoint /token

```
1 @router.post("/token", response_model=Token)
2 async def login_for_access_token(form_data:
3     Annotated[OAuth2PasswordRequestForm, Depends()],
4     db: db_dependency):
5     user = authenticate_user(form_data.username, form_data.password, db)
6     if not user:
7         raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
8                             detail='Could not validate user.')
9     token = create_access_token(user.username, user.id, timedelta(minutes=20))
10    return {'access_token': token, 'token_type': 'bearer'}
```

Funcionalidad

Este endpoint proporciona un servicio de autenticación de usuarios mediante la generación de un token de acceso. A continuación, se describe detalladamente el proceso y funcionalidad de este servicio:

Método POST

Ruta /token

Parámetros de Entrada

- `form_data`: Este parámetro es una instancia de `OAuth2PasswordRequestForm`, el cual contiene el nombre de usuario y la contraseña proporcionados por el cliente.
- `db`: Dependencia de la base de datos necesaria para la autenticación del usuario.

Proceso

1. **Autenticación del Usuario:** La función `authenticate_user` se encarga de verificar las credenciales proporcionadas (nombre de usuario y contraseña) contra los datos almacenados en la base de datos.
 - Si la autenticación falla, se lanza una excepción `HTTPException` con el código de estado 401 `Unauthorized` y un mensaje de error indicando que no se pudo validar el usuario.

2. **Generación del Token de Acceso:** Si la autenticación es exitosa, se genera un token de acceso mediante la función `create_access_token`, el cual incluye el nombre de usuario, el ID del usuario y una validez de 20 minutos.
3. **Respuesta:** La respuesta del endpoint incluye el token de acceso generado y el tipo de token (`bearer`).

Parámetros de Salida

- `access_token`: Token de acceso generado para el usuario autenticado.
- `token_type`: Tipo de token, en este caso `bearer`.

Excepciones

- `HTTPException 401 Unauthorized`: Se lanza si las credenciales proporcionadas no son válidas.

2.3 Middleware y logs de ejecución

Middleware se constituye como un software capaz de ejecutar acciones como se le indique para cada ejecución de un endpoint. Dada esta funcionalidad, la implementación de los logs se hace ideal con este framework ya que permitiera de manera centralizada y estructurada elaborar un archivo que haga registro de los logs de ejecución de la app y se tomen métricas y atributos relevantes sobre el request.

2.3.1 El archivo `app.log`

El archivo `app.log` contiene los registros de actividad y eventos generados por una aplicación denominada FCV-MLAPI. A continuación, se describen los componentes principales y las funcionalidades relevantes registradas en este archivo.

Estructura de Datos

Cada línea del archivo `app.log` sigue una estructura específica que incluye varios campos clave:

- **Fecha y Hora:** El timestamp del evento en formato `AAAA-MM-DD HH:MM:SS,MS`.
- **Nivel de Log:** El nivel de severidad del log, en este caso, `INFO`.
- **Mensaje:** El contenido del mensaje de log, que describe la acción o evento registrado.

Descripción de Funcionalidades y Eventos

El archivo registra varios tipos de eventos, incluyendo el inicio de la aplicación, peticiones HTTP y la generación de respuestas. A continuación, se detallan algunos ejemplos de estos eventos:

Inicio de la Aplicación

2024-05-05 11:58:53,524 - INFO - Starting FCV-MLAPI....

Este tipo de registro indica que la aplicación FCV-MLAPI ha iniciado. La frase `Starting FCV-MLAPI...` aparece cada vez que la aplicación se inicia.

Peticiones HTTP El archivo también contiene registros de peticiones HTTP, incluyendo la URL solicitada, el método HTTP, la fecha y hora de la solicitud, y la duración de la misma. Ejemplos de estos registros son:

```
1 2024-05-05 13:09:08,139 - INFO - {'url': '/docs', 'method': 'GET', 'date':
2 '05/05/2024 13:09:08', 'duration': 0.0020029544830322266}
2 2024-06-13 06:32:08,181 , INFO , {'url': '/auth/token', 'method': 'POST',
' date': '13/06/2024 06:32:08', 'duration': 0.6640031337738037}
```

En estos registros, se observa:

- `url`: La URL solicitada por el cliente.
- `method`: El método HTTP utilizado (GET o POST).
- `date`: La fecha y hora en que se realizó la solicitud.
- `duration`: La duración de la solicitud en segundos.

Duración de las Peticiones La duración de las peticiones es un campo importante que permite evaluar el rendimiento de la aplicación. Por ejemplo:

```
1 2024-06-13 06:36:41,960 , INFO , {'url': '/predict', 'method': 'POST',
' date': '13/06/2024 06:36:41', 'duration': 1.4350006580352783}
```

En este caso, una petición a la URL `/predict` mediante el método POST tardó aproximadamente 1.435 segundos en completarse.

El archivo `app.log` es esencial para monitorear y depurar la aplicación FCV-MLAPI. Registra eventos cruciales como el inicio de la aplicación y las peticiones HTTP, incluyendo detalles sobre la URL solicitada, el método utilizado, la fecha y hora, y la duración de cada solicitud. Estos registros son valiosos para entender el comportamiento de la aplicación y mejorar su rendimiento.

2.4 Artefactos

2.4.1 El archivo `main.py`

La aplicación está construida utilizando FastAPI y proporciona servicios de autenticación, predicción y comprobación de estado del modelo.

Importaciones y Configuración Inicial

```
1 from fastapi import FastAPI, status, Depends, HTTPException
2 from pydantic import BaseModel
3 import pandas as pd
4 from predict import predict
5 from predict import __version__ as model_version
6 import json
7 from logger import logger
8 import asyncio
9 from middleware import middleware_log
10 from starlette.middleware.base import BaseHTTPMiddleware
11 import models
12 from database import engine, SessionLocal
13 from typing import Annotated
14 from sqlalchemy.orm import Session
15 import auth
16 from auth import get_current_user
```

El código importa los módulos necesarios, incluyendo **FastAPI**, **pydantic** para la validación de datos, **pandas** para el manejo de datos, y otros módulos personalizados para predicciones, autenticación, logging y middleware.

Inicialización de la Aplicación

```
app = FastAPI()
app.include_router(auth.router)
models.Base.metadata.create_all(bind=engine)
```

Se crea una instancia de **FastAPI** y se incluye el router de autenticación. Además, se inicializa la base de datos creando todas las tablas definidas en **models**.

Dependencias de la Base de Datos

```
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

db_dependency = Annotated[Session, Depends(get_db)]
user_dependency = Annotated[dict, Depends(get_current_user)]
```

Se define una función para obtener la sesión de la base de datos y cerrar la conexión después de su uso. También se crean dependencias anotadas para la base de datos y el usuario autenticado.

Middleware

```
app.add_middleware(BaseHTTPMiddleware, dispatch=middleware_log)
```

Se agrega un middleware personalizado para el logging de peticiones HTTP.

Definición de Modelos

```
class BatchIn(BaseModel):
    idAtencion: int
    idSigno: dict
    nomSigno: dict
    valor: dict
    fecRegistro: dict

class PredictionOut(BaseModel):
    idAtencion: int
    inferences: dict
    State: str
```

Se definen los modelos **BatchIn** y **PredictionOut** utilizando **pydantic**. **BatchIn** representa la entrada para las predicciones, mientras que **PredictionOut** representa la salida.

Rutas de la Aplicación

Ruta Principal

```

@app.get("/", status_code=status.HTTP_200_OK)
async def user(user: user_dependency, db: db_dependency):
    if user is None:
        raise HTTPException(status_code=401, detail='Authentication Failed.')
    return {'User': user}

```

Esta ruta verifica la autenticación del usuario y retorna la información del mismo si está autenticado.

Ruta de Comprobación de Estado

```

@app.get("/home")
async def home():
    result = await asyncio.sleep(0)
    return {"health_check": "OK", "model_version": model_version}

```

Esta ruta proporciona una comprobación de estado del modelo, retornando un mensaje de salud y la versión del modelo.

Ruta de Predicción

```

1 @app.post("/predict", response_model=PredictionOut)
2 async def generate_inference(payload: BatchIn, user: user_dependency, db:
3   db_dependency, status_code=status.HTTP_200_OK):
4     if user is None:
5         raise HTTPException(status_code=401, detail='Authentication Failed.')
6     else:
7         data = pd.DataFrame({"idAtencion": payload.idAtencion,
8                             "nomSigno": payload.nomSigno,
9                             "valor": payload.valor,
10                            "fecRegistro": payload.fecRegistro})
11         inferences = await predict(data)
12         for item in inferences.columns:
13             print(item)
14         np_inference = inferences.Inference.item()[0]
15         inference = [float(x) for x in np_inference]
16         UserId = inferences.idAtencion.unique().item()
17         UserId = int(UserId)
18         return {"idAtencion": UserId,
19               "inferences": {'Arresto_Cardiaco': inference[0],
20                             'Bajo_Gasto_Cardiaco': inference[1],
21                             'Sano': inference[2],
22                             'Shock_Cardiogenico': inference[3]},
23               "State": inferences.State.unique().item()}

```

Esta ruta recibe una entrada de tipo BatchIn, verifica la autenticación del usuario, y genera inferencias basadas en los datos proporcionados. Las inferencias se retornan en el formato definido por PredictionOut.

2.4.2 El archivo logger.py

Documentación del Sistema de Logging en Python

El siguiente código establece un sistema de logging en Python que permite registrar mensajes de log tanto en la consola (stdout) como en un archivo llamado app.log. A continuación, se describe la funcionalidad y el razonamiento detrás de cada parte del código.

```

1 import logging
2 import sys
3
4 logger = logging.getLogger(__name__)
5 formatter = logging.Formatter(fmt="%(asctime)s , %(levelname)s ,
6                               %(message)s")
7 stream_handler = logging.StreamHandler(sys.stdout)
8 file_handler = logging.FileHandler('app.log')
9 stream_handler.setFormatter(formatter)
10 file_handler.setFormatter(formatter)
11 logger.handlers = [stream_handler, file_handler]
12 logger.setLevel(logging.INFO)

```

Importación de Módulos

- `import logging`: Importa el módulo de logging de Python, que proporciona una interfaz flexible para emitir mensajes de log desde programas Python.
- `import sys`: Importa el módulo sys, que proporciona acceso a algunas variables utilizadas o mantenidas por el intérprete de Python y a funciones que interactúan fuertemente con el intérprete.

Creación del Logger

```
logger = logging.getLogger(__name__)
```

Esta línea crea un objeto logger. El nombre del logger se establece en `__name__`, lo que permite que el nombre del logger sea el nombre del módulo en el cual se encuentra este código. Esto es útil para identificar el origen de los mensajes de log en aplicaciones más grandes.

Definición del Formato del Log

```
formatter = logging.Formatter(fmt="%(asctime)s , %(levelname)s , %(message)s")
```

Se crea un objeto formatter que define el formato de los mensajes de log. El formato especificado incluye:

- `%(asctime)s`: La fecha y hora en que se generó el mensaje de log.
- `%(levelname)s`: El nivel de severidad del mensaje de log (por ejemplo, INFO, WARNING, ERROR).
- `%(message)s`: El contenido del mensaje de log.

Configuración de los Handlers

```
stream_handler = logging.StreamHandler(sys.stdout)
file_handler = logging.FileHandler('app.log')
```

Se crean dos handlers:

- `stream_handler` : *Envía los mensajes de log a la consola (stdout).*

Asignación del Formato a los Handlers

- `stream_handler.setFormatter(formatter)`
`file_handler.setFormatter(formatter)`

Se asigna el objeto formatter creado previamente a ambos handlers, asegurando que los mensajes de log tengan el mismo formato sin importar su destino.

Asignación de los Handlers al Logger

```
logger.handlers = [stream_handler, file_handler]
```

Se asignan los handlers configurados al logger. Esto permite que cualquier mensaje de log emitido por el logger se envíe tanto a la consola como al archivo de log.

Configuración del Nivel de Log

```
logger.setLevel(logging.INFO)
```

Se establece el nivel de log del logger en `INFO`. Esto significa que el logger emitirá mensajes de log con un nivel de severidad de `INFO` o superior (por ejemplo, `WARNING`, `ERROR`, `CRITICAL`).

2.4.3 El archivo `predict.py`

Documentación del Código para Predicción de Estados Cardiológicos

Esta sección describe las funcionalidades y el razonamiento detrás del código proporcionado, el cual realiza predicciones de estados cardiológicos basadas en datos de pacientes utilizando un modelo de red neuronal LSTM.

Dependencias:

El código utiliza varias bibliotecas y dependencias que se deben importar al inicio:

- `pandas`: Para la manipulación de datos.
- `numpy`: Para operaciones numéricas.
- `StandardScaler` de `sklearn.preprocessing`: Para la normalización de los datos.
- `load_model` de `tensorflow.keras.models`: Para cargar el modelo LSTM preentrenado.
- `asyncio`: Para manejar operaciones asincrónicas.
- `Path` de `pathlib`: Para manejar rutas de archivos.

Inicialización

El código inicializa el modelo LSTM y define un diccionario de clases:

```
1 model = load_model('LSTM--1.0.0.h5')
2 diccionario_clases = {
3     0: 'Arresto_cardiaco',
4     1: 'Bajo_gasto_cardiaco',
5     2: 'Sano',
6     3: 'Shock_cardiogenico'
7 }
```

Función predict

La función `predict` es una función asincrónica que toma los datos de entrada y realiza predicciones. A continuación, se describen los pasos principales de esta función:

1. Creación del DataFrame

Se crea un DataFrame a partir de los datos de entrada:

```
df = pd.DataFrame(data)
```

2. Reorganización del DataFrame

El DataFrame se reorganiza utilizando la función `pivot_table` para estructurarlo según las especificaciones dadas:

```
1 df_resultante = df.pivot_table(index=['idAtencion', 'fecRegistro'],
2                               columns='nomSigno', values='valor').reset_index()
```

3. Filtrado de Datos

Se filtran los datos para incluir solo las columnas deseadas y se utilizan los últimos datos registrados por cada paciente:

```
1 columnas_deseadas = ['idAtencion', 'fecRegistro', 'MDC_BLD_PERF_INDEX',
2                     'MDC_ECG_HEART_RATE', 'MDC_ECG_V_P_C_RATE',
3                     'MDC_LEN_BODY_ACTUAL', 'MDC_MASS_BODY_ACTUAL',
4                     'MDC_PULS_OXIM_PULS_RATE', 'MDC_PULS_OXIM_SAT_O2',
5                     'MDC_TEMP', 'MDC_TTHOR_RESP_RATE']
6 df_resultante_filtrado = df_resultante[columnas_deseadas]
7 ultimas_20_filas_por_id =
8     df_resultante_filtrado.groupby('idAtencion').tail(20)
df_resultante_lleno = ultimas_20_filas_por_id.ffill()
```

4. Selección de Características

Se seleccionan las características relevantes para la predicción:

```
1 selected_features = ['MDC_BLD_PERF_INDEX', 'MDC_ECG_HEART_RATE',
2                     'MDC_ECG_V_P_C_RATE',
3                     'MDC_LEN_BODY_ACTUAL', 'MDC_MASS_BODY_ACTUAL',
4                     'MDC_PULS_OXIM_PULS_RATE',
5                     'MDC_PULS_OXIM_SAT_O2', 'MDC_TEMP', 'MDC_TTHOR_RESP_RATE']
6 features = df_resultante_lleno[selected_features]
```

5. Creación de Secuencias

Se convierten los datos a secuencias por paciente:

```
1 sequences = []
2 patient_ids = df_resultante_lleno['idAtencion'].unique()
3 for patient_id in patient_ids:
4     patient_data = features[df_resultante_lleno['idAtencion'] == patient_id]
5     sequences.append(patient_data.values)
```

6. Normalización de los Datos

Se normalizan los datos utilizando `StandardScaler`:

```
1 scaler = StandardScaler()
2 X_test = [scaler.fit_transform(seq) for seq in sequences]
3 X_test = np.reshape(X_test, (len(X_test), X_test[0].shape[0],
4                               X_test[0].shape[1]))
```

7. Predicción

Se realizan las predicciones utilizando el modelo LSTM cargado:

```

1 results = pd.DataFrame()
2 for i in range(len(X_test)):
3     resultados = model.predict(X_test[i:])
4     max_index = np.argmax(resultados)
5     state = diccionario_clases[max_index]
6     temp = pd.DataFrame({'idAtencion': patient_ids[i], 'Inference':
7         [resultados], "State": state}, index=[0])
8     results = pd.concat([results, temp])

```

Se imprime la clase predicha para cada paciente:

```

1 for idx, resultado in enumerate(resultados, start=1):
2     clase_predicha = np.argmax(resultado)
3     nombre_clase_predicha = diccionario_clases[clase_predicha]
4     print(f"Prediccin {idx}: El paciente con id {patient_ids[i]} se
5         encuentra en estado: {nombre_clase_predicha}")

```

8. Retorno de Resultados

Finalmente, la función retorna los resultados obtenidos:

```
return results
```

2.4.4 El archivo auth.py

Esta seccion describe las funcionalidades y el razonamiento detrás de un conjunto de códigos implementados con FastAPI, SQLAlchemy y otras librerías de autenticación y encriptación. El objetivo principal del código es proporcionar un sistema de autenticación basado en tokens para una aplicación web.

Código Completo

```

1 import sqlalchemy
2 print(sqlalchemy.__version__)
3 print('test')
4 from datetime import timedelta, datetime
5 from typing import Annotated
6 from fastapi import APIRouter, Depends, HTTPException
7 from pydantic import BaseModel
8 from sqlalchemy.orm import Session
9 from starlette import status
10 from database import SessionLocal
11 from models import Users
12 from passlib.context import CryptContext
13 from fastapi.security import OAuth2PasswordRequestForm, OAuth2PasswordBearer
14 from jose import jwt, JWTError
15 router = APIRouter(
16     prefix='/auth',
17     tags=['auth']
18 )
19 SECRET_KEY =
20     'eyJhbGciOiJIUzI1NiJ9.eyJSc2x1IjojQWRtaW4iLCJpc3N1ZXIiOiJlc3N1ZXIiLCJp
21     VmVybWVtZSI6ImFkbmFJb1VzZSI6ImV4cCI6MTcxODA2Mjk0NCwiaWF0Ijox
22     NzE4MDYyOTQ0fQ.zN9eemsiMb7rGanbHVXumbU5wHJDnDBYg3jpb8WoRaAg'
23 ALGORITHM = 'HS256'
24 bcrypt_context = CryptContext(schemes=['bcrypt'], deprecated='auto')
25 oauth2_bearer = OAuth2PasswordBearer(tokenUrl='auth/token')
26
27 class CreateUserRequest(BaseModel):
28     username: str
29     password: str
30

```

[illegible]

Descripción de Funcionalidades:**Importaciones y Configuraciones Iniciales**

El código comienza con las importaciones necesarias de varias librerías:

- `sqlalchemy`: Para trabajar con la base de datos.
- `datetime`: Para manejar fechas y tiempos.
- `typing`: Para anotaciones de tipos.
- `fastapi`, `pydantic`, `starlette`: Para la creación de la API y manejo de solicitudes y respuestas.
- `passlib.context`: Para la gestión de la encriptación de contraseñas.
- `jose.jwt`: Para la creación y verificación de JSON Web Tokens (JWT).

Además, se configura el router de FastAPI y se definen constantes para la clave secreta y el algoritmo de encriptación.

Definición de Clases

- `CreateUserRequest`: Clase que define el esquema de solicitud para la creación de un usuario, con los campos `username` y `password`.
- `Token`: Clase que define el esquema de respuesta para el token de acceso, con los campos `access_token` y `token_type`.

Gestión de la Base de Datos

La función `get_db` se encarga de proporcionar una sesión de base de datos para las solicitudes entrantes. Utiliza un contexto de gestión para asegurar que la conexión se cierre correctamente después de cada uso.

Creación de Usuarios

El endpoint `create_user` permite crear un nuevo usuario. Toma un objeto `CreateUserRequest` y guarda el usuario en la base de datos con la contraseña encriptada.

Autenticación y Generación de Tokens

El endpoint `login_for_access_token` se utiliza para autenticar a los usuarios y generar un token de acceso. Si las credenciales son válidas, se genera un JWT con una expiración de 20 minutos.

Funciones de Ayuda

- `authenticate_user`: Verifica las credenciales del usuario contra la base de datos.
- `create_access_token`: Crea un JWT con el nombre de usuario y el ID del usuario, y una fecha de expiración.
- `get_current_user`: Decodifica el token JWT para obtener el usuario actual.

Razonamiento

El código sigue buenas prácticas de desarrollo de software, incluyendo:

- Uso de dependencias para gestionar la inyección de dependencias (`Depends`).
- Separación de responsabilidades mediante funciones y clases bien definidas.
- Manejo seguro de contraseñas con encriptación.
- Generación y verificación de tokens seguros para la autenticación.

2.4.5 El archivo `database.py`

Esta sección explica las funcionalidades y el razonamiento detrás del siguiente código en Python que utiliza SQLAlchemy para la gestión de una base de datos SQLite.

```
1 import sqlalchemy
2 from sqlalchemy import create_engine
3 from sqlalchemy.orm import sessionmaker
4 from sqlalchemy.ext.declarative import declarative_base
5
6 SQLALCHEMY_DATABASE_URL = 'sqlite:///./todosapp.db'
7
8 engine = create_engine(SQLALCHEMY_DATABASE_URL,
9                       connect_args={'check_same_thread': False})
10 SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
11
12 Base = declarative_base()
```

Importación de Módulos

- `import sqlalchemy`: Importa el paquete principal de SQLAlchemy, una biblioteca de SQL para Python.
- `from sqlalchemy import create_engine`: Importa la función `create_engine`, que se utiliza para configurar la conexión a la base de datos.
- `from sqlalchemy.orm import sessionmaker`: Importa `sessionmaker`, una función que se utiliza para crear nuevas sesiones de base de datos.
- `from sqlalchemy.ext.declarative import declarative_base`: Importa `declarative_base`, una fábrica que se utiliza para construir una clase base para modelos declarativos.

Configuración de la URL de la Base de Datos

```
SQLALCHEMY_DATABASE_URL = 'sqlite:///./todosapp.db'
```

Se define una URL para la base de datos SQLite. Esta URL indica que se utilizará una base de datos SQLite local llamada `todosapp.db` que se ubicará en el mismo directorio que el script Python.

Creación del Motor de la Base de Datos

```
1 engine = create_engine(SQLALCHEMY_DATABASE_URL,
2                       connect_args={'check_same_thread': False})
```

La función `create_engine` crea un motor de base de datos que se utilizará para interactuar con la base de datos SQLite. El argumento `connect_args` es específico para SQLite y permite que la base de datos sea accedida desde múltiples hilos.

Creación de la Sesión de la Base de Datos

```
1 SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

La función `sessionmaker` se utiliza para crear una clase de sesión. Esta clase se utilizará posteriormente para crear instancias de sesión de base de datos. Los argumentos `autocommit=False` y `autoflush=False` configuran la sesión para no realizar commits ni flush automáticos. El argumento `bind=engine` vincula la sesión al motor de la base de datos creado previamente.

Declaración de la Clase Base

```
Base = declarative_base()
```

La función `declarative_base` se utiliza para crear una clase base a partir de la cual se definirán todas las clases de modelo. Esta clase base contiene metadatos que se utilizarán para crear las tablas en la base de datos.

2.4.6 El archivo `models.py`

Descripción del Código: Definición del Modelo `Users`

El siguiente código define un modelo de base de datos utilizando SQLAlchemy, una biblioteca ORM (Object-Relational Mapping) para Python. Este modelo representa la tabla `users` en la base de datos.

```
1 from database import Base
2 from sqlalchemy import Column, Integer, String
3
4 class Users(Base):
5     __tablename__ = 'users'
6
7     id = Column(Integer, primary_key=True, index=True)
8     username = Column(String, unique=True)
9     hashed_password = Column(String)
```

Importaciones

- `from database import Base`: Importa la clase `Base` desde el módulo `database`. Esta clase es la base para todos los modelos ORM en SQLAlchemy.
- `from sqlalchemy import Column, Integer, String`: Importa las clases necesarias para definir las columnas de la tabla y sus tipos de datos.

Definición de la Clase `Users`

La clase `Users` hereda de `Base` y representa la tabla `users` en la base de datos.

Atributos de Clase

- `__tablename__ = 'users'`: Define el nombre de la tabla en la base de datos que este modelo representa.
- `id = Column(Integer, primary_key=True, index=True)`: Define la columna `id` como un entero, clave primaria, e indexada para optimizar las búsquedas.
- `username = Column(String, unique=True)`: Define la columna `username` como una cadena de texto que debe ser única, asegurando que no haya dos usuarios con el mismo nombre de usuario.
- `hashed_password = Column(String)`: Define la columna `hashed_password` como una cadena de texto que almacena la contraseña del usuario de forma encriptada.

Razonamiento del Código

- **Herencia de Base**: Heredar de `Base` permite que la clase `Users` se integre con SQLAlchemy, permitiendo la creación, consulta, actualización y eliminación de registros en la tabla `users`.

- **Uso de `__tablename__`:** Especificar `__tablename__` asegura que SQLAlchemy sepa a qué tabla en la base de datos se refiere este modelo.
- **Definición de Columnas:**
 - `id`: Utilizar una clave primaria es esencial para identificar de manera única cada registro en la tabla.
 - `username`: Hacer que esta columna sea única impide la duplicación de nombres de usuario, lo cual es crucial para la autenticación y gestión de usuarios.
 - `hashed_password`: Almacenar la contraseña de forma encriptada mejora la seguridad, protegiendo la información sensible de los usuarios.

2.4.7 El archivo `middleware.py`

Descripción del Script de Middleware para Logging

Esta sección describe la funcionalidad de un script de middleware en Python que utiliza FastAPI para registrar información sobre las solicitudes HTTP entrantes. El script hace uso de módulos como `asyncio`, `datetime` y `time`, así como un módulo personalizado de `logger` para llevar a cabo el registro de eventos.

Descripción del Script: A continuación, se presenta el código del script:

```

1 import asyncio
2 from fastapi import Request
3 from logger import logger
4 from datetime import datetime
5 import time
6
7 async def middleware_log(request: Request, call_next):
8     start = time.time()
9     response = await call_next(request)
10    duration = time.time() - start
11    log_dict = {
12        'url': request.url.path,
13        'method': request.method,
14        'date': datetime.now().strftime("%d/%m/%Y %H:%M:%S"),
15        'duration': duration
16    }
17    logger.info(log_dict, extra=log_dict)
18    return response;
```

Funcionalidad: El script proporciona la funcionalidad de middleware en una aplicación FastAPI, con el objetivo de registrar detalles de cada solicitud HTTP que llega al servidor. La funcionalidad del script se describe a continuación:

- **Importación de Módulos:** Se importan los módulos necesarios:
 - `asyncio` para manejar funciones asíncronas.
 - `fastapi.Request` para manejar las solicitudes HTTP.
 - `logger` desde un módulo personalizado para el registro de eventos.
 - `datetime` y `time` para manejar y registrar fechas y tiempos.

- **Función Asíncrona:** Se define una función asíncrona `middleware_log` que toma como parámetros una solicitud (`request`) y una función (`call_next`) que procesa la solicitud.
- **Medición del Tiempo de Ejecución:** Al inicio de la función, se registra el tiempo de inicio (`start`) utilizando la función `time.time()`.
- **Llamada a la Siguiente Función del Middleware:** Se llama a la función `call_next` para procesar la solicitud y se espera la respuesta (`response`).
- **Duración de la Solicitud:** Se calcula la duración de la solicitud restando el tiempo de inicio del tiempo actual después de procesar la solicitud.
- **Creación del Diccionario de Log:** Se crea un diccionario `log_dict` que contiene:
 - `url`: La ruta de la solicitud.
 - `method`: El método HTTP utilizado (GET, POST, etc.).
 - `date`: La fecha y hora actual en formato `dd/mm/AAAA HH:MM:SS`.
 - `duration`: La duración de la solicitud en segundos.
- **Registro de la Información:** Se utiliza el `logger` para registrar la información del diccionario de log.
- **Devolución de la Respuesta:** Finalmente, se devuelve la respuesta obtenida de la función `_next`.

Chapter 3

Despliegue de operaciones

Docker es una plataforma de contenedorización que permite a los desarrolladores empaquetar una aplicación y sus dependencias en un contenedor, garantizando que la aplicación se ejecute de manera consistente en cualquier entorno. Los contenedores son ligeros y portátiles, proporcionando un aislamiento eficiente de las aplicaciones y sus entornos. Docker se utiliza ampliamente para el desarrollo, prueba y despliegue de aplicaciones, facilitando la integración continua y la entrega continua (CI/CD).

3.0.1 Ventajas de usar Docker

- **Portabilidad:** Docker permite empaquetar la aplicación junto con todas sus dependencias en un contenedor, asegurando que funcione de manera consistente en cualquier entorno.
- **Aislamiento:** Cada contenedor es independiente, lo que evita conflictos de dependencias y facilita el mantenimiento y la actualización de componentes individuales.
- **Escalabilidad:** Docker facilita la escalabilidad horizontal, permitiendo desplegar múltiples instancias de la aplicación para manejar aumentos en la carga.
- **Eficiencia:** Los contenedores son más ligeros que las máquinas virtuales, lo que reduce el uso de recursos y mejora el rendimiento.
- **Rapidez en el despliegue:** La creación y despliegue de contenedores es rápida, lo que acelera los ciclos de desarrollo y despliegue.

3.0.2 Contras de usar Docker

- **Complejidad Inicial:** Configurar y gestionar contenedores puede tener una curva de aprendizaje pronunciada para aquellos que no están familiarizados con Docker.

- **Persistencia de Datos:** Manejar el almacenamiento persistente puede ser complicado, ya que los datos dentro de los contenedores no son persistentes por defecto.
- **Seguridad:** Aunque Docker proporciona aislamiento, compartir el núcleo del sistema operativo puede presentar riesgos de seguridad si no se configuran adecuadamente las políticas de seguridad.
- **Rendimiento:** En algunos casos, el overhead de la capa de abstracción de Docker puede impactar el rendimiento en comparación con la ejecución directa en el host.

3.1 El archivo Dockerfile

El `Dockerfile` es un script que contiene una serie de instrucciones para crear una imagen de Docker. A continuación se muestra y explica cada instrucción del `Dockerfile`.

```
1 FROM python:3.8-slim
2
3 WORKDIR /app
4
5 COPY requirements.txt requirements.txt
6 RUN pip install -r requirements.txt
7
8 COPY . .
9
10 CMD ["python", "app.py"]
```

LISTING 3.1: Dockerfile

- **FROM:** Esta instrucción especifica la imagen base desde la cual se creará nuestra nueva imagen. En este caso, se utiliza `python:3.8-slim`, una versión ligera de Python 3.8.
- **WORKDIR:** Establece el directorio de trabajo dentro del contenedor en `/app`.
- **COPY:** Copia el archivo `requirements.txt` desde el sistema host al directorio de trabajo del contenedor.
- **RUN:** Ejecuta un comando dentro del contenedor. En este caso, instala las dependencias listadas en `requirements.txt` utilizando `pip`.
- **COPY:** Copia todos los archivos del directorio actual del host al directorio de trabajo del contenedor.
- **CMD:** Especifica el comando por defecto que se ejecutará cuando se inicie un contenedor a partir de esta imagen. Aquí, se ejecuta el script `app.py` utilizando Python.

El archivo `.dockerignore` especifica los archivos y directorios que Docker debe ignorar al copiar archivos a la imagen. Esto puede ayudar a reducir el tamaño de la imagen y a mejorar la seguridad.


```
1 __pycache__
2 *.pyc
3 *.pyo
4 *.pyd
5 .env
6 .git
7 .vscode
```

LISTING 3.2: `.dockerignore`

- `__pycache__`: Carpeta donde se almacenan los archivos bytecode compilados de Python.
- `*.pyc`, `*.pyo`, `*.pyd`: Archivos bytecode de Python que no son necesarios para el contenedor.
- `.env`: Archivos de variables de entorno que pueden contener información sensible.
- `.git`: El directorio Git que contiene la historia del repositorio, generalmente no necesario dentro del contenedor.
- `.vscode`: Directorio de configuración del editor Visual Studio Code, no relevante para la ejecución en contenedor.

3.2 El archivo `compose.yaml`

El archivo `docker-compose.yml` se utiliza para definir y ejecutar aplicaciones Docker multi-contenedor. A continuación se presenta un ejemplo y su explicación.

```
1 version: '3.8'
2
3 services:
4   app:
5     build:
6       context: .
7       dockerfile: Dockerfile
8     ports:
9       - "5000:5000"
10    volumes:
11      - ./app
12    environment:
13      - FLASK_ENV=development
```

LISTING 3.3: `docker-compose.yml`

- **version**: Especifica la versión del formato de `docker-compose.yml` que se está utilizando.
- **services**: Define los servicios que se ejecutarán. En este caso, solo hay un servicio llamado `app`.

- **app/build/context**: Indica el contexto de construcción, que es el directorio actual (.).
- **app/build/dockerfile**: Especifica el archivo `Dockerfile` a utilizar para la construcción.
- **app/ports**: Mapea el puerto 8000 del contenedor al puerto 8000 del host.
- **app/volumes**: Monta el directorio actual del host en el directorio `/app` del contenedor, permitiendo el desarrollo en caliente.
- **app/environment**: Establece variables de entorno. En este caso, `FLASK_ENV` se establece en `development`.

Chapter 4

Casos de usuario

El caso de usuario se aborda desde la arquitectura propuesta y la infraestructura de software planteada. De acuerdo al diseño inicial propuesto, el entorno dedicado a produccion y operaciones debera poder recibir consultas y estar en la capacidad de comunicar sus resultados. Para ello se implementa el protocolo HTTP de modo que existira la relacion cliente - servidor lo cual hace necesario la declaracion de puertos y reglas del host.

Usando contenedores, la virtualizacion del ambiente es una de las condiciones de operacion, lo que conlleva la necesidad de asignar las relaciones pertinentes entre los puertos del ambiente virtualizado y la maquina local host, permitiendo asi la via libre de peticiones y respuestas entre los clientes de la aplicacion (otras aplicaciones con acceso a la red privada) y la maquina local host (maquina que contiene el contenedor en ejecucion).

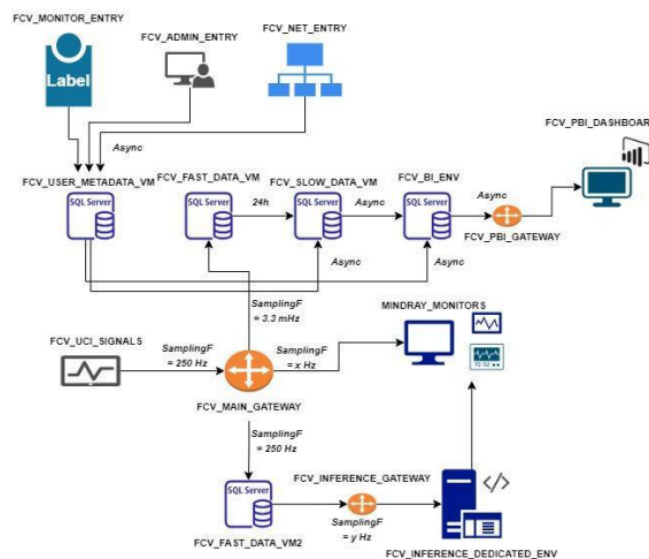


FIGURE 4.1: Infra-estructura de la solución

4.1 Servidor

Para coordinar esta ejecución, es necesario contar con el motor de Docker instalado en el ambiente de despliegue. Una vez activo el motor de Docker, instanciar la consola a la ruta raíz donde se alojan los archivos docker: Dockerfile, compose.yaml y .dockerignore. Una vez allí, ejecutar las líneas docker-compose up -d, esto iniciará la aplicación Docker como lo indica el archivo compose.yaml, permitiendo así la configuración de volúmenes y puertos pre establecida.

```

1 PS C:\Users\artur\OneDrive\Documentos\ML_FASTAPI_FCV\app>
   docker-compose up -d
2 [+] Running 1/2
3 - Network app_default   Created

   0.7s
4   Container app-server-1 Started

   0.7s
5 PS C:\Users\artur\OneDrive\Documentos\ML_FASTAPI_FCV\app>

```

Una vez en curso, para confirmar la ejecución y/o creación del contenedor se consulta mediante el comando docker ps -a, esto listará los contenedores presentes en la máquina local.

```

1 PS C:\Users\artur\OneDrive\Documentos\ML_FASTAPI_FCV\app>
   docker ps -a
2 CONTAINER ID   IMAGE          COMMAND                  CREATED
3              STATUS        PORTS
4 18655d182e83   app-server    "/bin/sh -c 'uvicorn"   About a
   minute ago   Up About a minute   80/tcp, 0.0.0.0:8000->8000/tcp
   app-server-1
5 PS C:\Users\artur\OneDrive\Documentos\ML_FASTAPI_FCV\app>

```

Para acceder a los últimos logs de ejecución generados por la aplicación, basta con ejecutar el comando docker-compose logs y este listará la información obtenida en la consola de ejecución de la aplicación contenerizada.

```

1 PS C:\Users\artur\OneDrive\Documentos\ML_FASTAPI_FCV\app>
   docker-compose logs
2 server-1 | INFO:    Will watch for changes in these
   directories: ['/app']
3 server-1 | INFO:    Uvicorn running on http://0.0.0.0:8000
   (Press CTRL+C to quit)
4 server-1 | INFO:    Started reloader process [7] using
   WatchFiles
5 server-1 | 2024-07-12 16:55:47.016356: I
   external/local_tsl/tsl/cuda/cudart_stub.cc:31] Could not
   find cuda drivers
6 on your machine, GPU will not be used.

```

```

7 server-1 | 2024-07-12 16:55:47.108020: E
  external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261]
  Unable to register cuDNN factory: Attempting to register
  factory for plugin cuDNN when one has already been
  registered
8 server-1 | 2024-07-12 16:55:47.108111: E
  external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607]
  Unable to register cuFFT factory: Attempting to register
  factory for plugin cuFFT when one has already been
  registered
9 server-1 | 2024-07-12 16:55:47.110305: E
  external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515]
  Unable to register cuBLAS factory: Attempting to register
  factory for plugin cuBLAS when one has already been
  registered
10 server-1 | 2024-07-12 16:55:47.126095: I
  external/local_tsl/tsl/cuda/cudart_stub.cc:31] Could not
  find cuda drivers
11 on your machine, GPU will not be used.
12 server-1 | 2024-07-12 16:55:47.126534: I
  tensorflow/core/platform/cpu_feature_guard.cc:182] This
  TensorFlow binary is optimized to use available CPU
  instructions in performance-critical operations.
13 server-1 | To enable the following instructions: AVX2 FMA, in
  other operations, rebuild TensorFlow with the appropriate
  compiler flags.
14 server-1 | 2024-07-12 16:55:50.532890: W
  tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38]
  TF-TRT Warning: Could not find TensorRT
15 server-1 | 2.0.30
16 server-1 | test
17 server-1 | 2024-07-12 16:55:55,043 , INFO , Starting
  FCV-MLAPI.....
18 server-1 | INFO:      Started server process [9]
19 server-1 | INFO:      Waiting for application startup.
20 server-1 | INFO:      Application startup complete.
21 PS C:\Users\artur\OneDrive\Documentos\ML_FASTAPI_FCV\app>

```

Los warnings obtenidos refieren a la ausencia de los drivers CUDA en la maquina local host, dado que no es de tipo data center ni es distribucion de tipo GPU , el software CUDA no es compatible ni se ejecutaria de ninguna forma en dicha maquina.

Segun los logs de ejecucion obtenidos por parte de la aplicacion, podria afirmarse que el servidor se construye y se ejecuta con exito. Para efectuar una prueba, consumir el endpoint de pruebas <http://127.0.0.1:8000/docs> y consultar nuevamente el log de ejecucion.

```

1 server-1 | INFO:      172.19.0.1:43496 - "GET /docs HTTP/1.1" 200
  OK
2 server-1 | 2024-07-12 17:06:15,561 , INFO , {'url':
  '/openapi.json', 'method': 'GET', 'date': '12/07/2024
  17:06:15', 'duration': 0.052373647689819336}

```

```

3 server-1 | INFO:      172.19.0.1:43496 - "GET /openapi.json
  HTTP/1.1" 200 OK
4 PS C:\Users\artur\OneDrive\Documentos\ML_FASTAPI_FCV\app>

```

Obtenida la respuesta HTTP 200, se concluye que el contenedor se ejecuta de manera correcta y relaciona correctamente los puertos de comunicacion respectivos de la maquina host y del contenedor.

Para apagar el servidor, solo es necesario ejecutar en la ruta raiz de la aplicacion: `docker-compose down`. Esto detendra la virtualizacion del ambiente y por lo tanto consigo el servidor.

```

1 PS C:\Users\artur\OneDrive\Documentos\ML_FASTAPI_FCV\app>
  docker-compose down
2 [+] Running 2/2
3   Container app-server-1 Removed

   10.5s
4   Network app_default   Removed

   0.3s
5 PS C:\Users\artur\OneDrive\Documentos\ML_FASTAPI_FCV\app>

```

4.1.1 Jerarquia de directorios

La aplicacion fue seccionada en modulos de trabajo, modelos de datos, bases de datos, requerimientos, logs de ejecucion y ejemplos de prueba. En el directorio `modules` se encuentran los artefactos que constituyen piezas funcionales de la aplicacion. Aqui se alojan artefactos como `auth.py`, `predict.py` y el modelo de inteligencia artificial desarrollado por CPS.














	<code>_pycache_</code>	11/07/2024 1:02 p. m.	Carpeta de archivos	
	<code>dbs</code>	11/07/2024 12:19 p. m.	Carpeta de archivos	
	<code>logs</code>	11/07/2024 12:18 p. m.	Carpeta de archivos	
	<code>models</code>	11/07/2024 4:40 p. m.	Carpeta de archivos	
	<code>modules</code>	12/07/2024 7:39 a. m.	Carpeta de archivos	
	<code>requirements</code>	11/07/2024 12:19 p. m.	Carpeta de archivos	
	<code>test</code>	11/07/2024 12:10 p. m.	Carpeta de archivos	
	<code>venv</code>	13/06/2024 5:44 a. m.	Carpeta de archivos	
	<code>.dockerignore</code>	5/07/2024 11:32 a. m.	Archivo DOCKERI...	1 KB
	<code>compose</code>	5/07/2024 4:18 p. m.	Archivo de origen ...	2 KB
	<code>Dockerfile</code>	11/07/2024 3:48 p. m.	Archivo	1 KB
	<code>main</code>	12/07/2024 11:15 a. m.	Python File	3 KB
	<code>README.Docker</code>	5/07/2024 9:16 a. m.	Archivo de origen ...	1 KB

FIGURE 4.2: Distribucion interna

El directorio `dbs` representa el espacio dedicado para alojar la base de datos de la aplicacion. Quedando expuesta como un volumen en el contenedor, los cambios efectuados en la misma seran reflejados en dicho volumen permitiendole a otros servicios de ser necesario consultar o incluso editar sobre la base de datos.

4.2 Desarrollo en vivo

Construir la ejecucion a partir del archivo `compose.yaml` permitira que se defina como volumen el directorio raiz de la aplicacion haciendo posible que el contenedor pueda escuchar en vivo a cualquier cambio que se realice dentro de este directorio y ajustarlo en el servidor sin la necesidad de generar una nueva imagen docker para otro contenedor, lo que a ade la funcionalidad de desarrollo en vivo, permitiendole al desarrollador ejecutar ediciones y validar estos cambios tomar efecto en el servidor.

```
server-1 | INFO: 172.19.0.1:43496 - "GET /docs HTTP/1.1" 200 OK
server-1 | 2024-07-12 17:06:15,561 | INFO | {'url': '/openapi.json', 'method': 'GET', 'date': '12/07/2024 17:06:15',
server-1 | 'duration': 0.052373647689819336}
server-1 | INFO: 172.19.0.1:43496 - "GET /openapi.json HTTP/1.1" 200 OK
server-1 | WARNING: WatchFiles detected changes in 'modules/predict.py'. Reloading...
server-1 | INFO: Shutting down
server-1 | INFO: Waiting for application shutdown.
server-1 | INFO: Application shutdown complete.
server-1 | INFO: Finished server process [9]
```

FIGURE 4.3: Logs de ejecucion

Una vez guardada la edicion hecha sobre el artefacto `predict.py` el servidor reinicia su operacion y resalta sobre los cambios registrados.

4.3 Cliente

Dado el protocolo usado de comunicacion HTTP, se sugiere el consumo de la aplicacion mediante un cliente python, pero no necesariamente tendria que ser un cliente python, podria ser cualquier cliente que pueda resolver peticiones AJAX, esto incluye clientes: js, C++, java, python, html, etc. Para el caso del ejemplo incluido, se construye un cliente python para el consumo de la aplicacion.

4.3.1 El archivo `client.py`

Construido en python y alojado en el directorio `test`, este cliente primero autentica su sesion para posteriormente realizar una consulta con el archivo `input_test.json` el cual contiene informacion dummy para evaluar el modelo de IA.

```
1 import requests
2 import json
3
4 # URL base de la aplicacin local
5 base_url = "http://127.0.0.1:8000"
6
7 # Cargar los datos de entrada para el endpoint de prediccin
8 with open('input_test.json') as f:
9     input_data = json.load(f)
10
11 # Credenciales de autenticacin
12 auth_data = {
13     "username": "CPS_DAVID_PEREZ_DEV",
14     "password": "UIS@123FCV"
15 }
16
17 def get_jwt_token(auth_url, auth_data):
18     response = requests.post(auth_url, json=auth_data)
19     if response.status_code == 200:
```

```

20     return response.json().get('access_token')
21 else:
22     raise Exception("Authentication failed")
23
24 # Endpoints
25 endpoints = {
26     "login": "/",
27     "predict": "/predict"
28 }
29
30 # Obtener el token JWT
31 token = get_jwt_token(base_url + "/auth/token", auth_data)
32 headers = {'Authorization': f'Bearer {token}'}
33 response = requests.post(base_url + "/predict", json=input_data,
34                           headers=headers)
35
36 print(f"Status Code: {response.status_code}")
37 print(f"Response: {response.json()}")
38 print("="*50)

```

Ejecutando el anterior script se debería obtener: **Servidor:**

```

server-1 | 2024-07-12 19:27:20,453 , INFO , {'url': '/auth/token', 'method': 'POST', 'date': '12/07/2024 19:27:20',
'duration': 0.8506975173950195}
server-1 | INFO: 172.20.0.1:35730 - "POST /auth/token HTTP/1.1" 200 OK
1/1 [=====] - 1s 642ms/step
server-1 | Predicción 1: El paciente con id 1408826 se encuentra en estado: Sano
server-1 | idAtencion
server-1 | Inference
server-1 | State
server-1 | <class 'list'>
server-1 | 2024-07-12 19:27:21,435 , INFO , {'url': '/predict', 'method': 'POST', 'date': '12/07/2024 19:27:21', 'du
ration': 0.9533584117889404}
server-1 | INFO: 172.20.0.1:35746 - "POST /predict HTTP/1.1" 200 OK
PS C:\Users\artur\OneDrive\Documentos\ML_FASTAPI_FCV\app>

```

FIGURE 4.4: Logs de ejecución - Canal del servidor

Cliente:

```

PS C:\Users\artur\OneDrive\Documentos\ML_FASTAPI_FCV\app\test> python client.py
Status Code: 200
Response: {'idAtencion': 1408826, 'Inferences': {'Arresto_Cardiac': 0.06022471934556961, 'Bajo_gasto_Cardiac': 0.93
5220397963349e-08, 'Sano': 0.9397750496864319, 'Shock_Cardlogenico': 7.849151728578363e-08}, 'State': 'Sano'}
=====
PS C:\Users\artur\OneDrive\Documentos\ML_FASTAPI_FCV\app\test>

```

FIGURE 4.5: Logs de ejecución - Canal del cliente

Guide written by —
Sunil Patel: www.sunilpatel.co.uk
Vel: LaTeXTemplates.com

Appendix A

Frequently Asked Questions

A.1 How do I change the colors of links?

The color of links can be changed to your liking using:

```
\hypersetup{urlcolor=red}, or  
\hypersetup{citecolor=green}, or  
\hypersetup{allcolor=blue}.
```

If you want to completely hide the links, you can use:

```
\hypersetup{allcolors=.}, or even better:  
\hypersetup{hidelinks}.
```

If you want to have obvious links in the PDF but not the printed text, use:

```
\hypersetup{colorlinks=false}.
```


Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor. . .