- prediction_service module
  - class prediction_service.DataRequest(*, angulo: list, par: list, reset: bool, identificador: str)
    - angulo : list
    - identificador : str
    - model_computed_fields : ClassVar[dict[str, ComputedFieldInfo]] = {}
    - model_config : ClassVar[ConfigDict] = {}
    - model_fields : ClassVar[dict[str, FieldInfo]] = {'angulo': FieldInfo(annotation=list, required=True), 'identificador': FieldInfo(annotation=str, required=True), 'par': FieldInfo(annotation=list, required=True), 'reset': FieldInfo(annotation=bool, required=True)}
    - par : list
    - reset : bool
  - class prediction_service.ModelUpdateRequest(*, model_folder: str, model_name: str, window_size: int)
    - model_computed_fields : ClassVar[dict[str, ComputedFieldInfo]] = {}
    - model_config : ClassVar[ConfigDict] = {'protected_namespaces': ()}
    - model_fields : ClassVar[dict[str, FieldInfo]] = {'model_folder': FieldInfo(annotation=str, required=True), 'model_name': FieldInfo(annotation=str, required=True), 'window_size': FieldInfo(annotation=int, required=True)}
    - model_folder : str
    - model_name : str
    - window_size : int
  - async prediction_service.get_data()
  - async prediction_service.health_check()
  - async prediction_service.receive_data(request: DataRequest)
  - prediction_service.start_service()
  - async prediction_service.update_model(request: ModelUpdateRequest)

prediction_service module

Servicio de Datos con FastAPI

Este script implementa un servicio de datos utilizando FastAPI. Se encarga de recibir, procesar y enviar datos para la predicción.

Imports: : - fastapi: Framework para construir APIs.

- pydantic: Librería para validación de datos.
- requests: Librería para realizar solicitudes HTTP.

Funciones: : - update_model: Actualiza la información del modelo.

- receive_data: Recibe y procesa los datos.
- get_data: Envía los datos almacenados.
- health_check: Verifica el estado del servidor.
- start_service: Inicia el servidor de FastAPI.

class prediction_service.DataRequest(*, angulo: list, par: list, reset: bool, identificador: str)

Bases: `BaseModel`

Modelo de solicitud de datos.

Atributos: : angulo (list): Lista de valores de ángulo. par (list): Lista de valores de par. reset (bool): Indicador de reinicio de datos. identificador (str): Identificador del coche.

angulo ***: list***

identificador ***: str***

model_computed_fields ***: ClassVar[dict[str, ComputedFieldInfo]] = {}***

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_config ***: ClassVar[ConfigDict] = {}***

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

model_fields ***: ClassVar[dict[str, FieldInfo]] = {'angulo': FieldInfo(annotation=list, required=True), 'identificador': FieldInfo(annotation=str, required=True), 'par':***

FieldInfo(annotation=list, required=True), 'reset': FieldInfo(annotation=bool, required=True)}

Metadata about the fields defined on the model, mapping of field names to [FieldInfo] [pydantic.fields.FieldInfo].

This replaces Model._*fields*_ from Pydantic V1.

par ***: list***

reset ***: bool***

class prediction_service.ModelUpdateRequest(*, model_folder: str, model_name: str, window_size: int)

Bases: `BaseModel`

Modelo de solicitud de actualización del modelo.

Atributos: : model_folder (str): Carpeta del modelo. model_name (str): Nombre del modelo. window_size (int): Tamaño de la ventana de datos.

model_computed_fields ***: ClassVar[dict[str, ComputedFieldInfo]] = {}***

A dictionary of computed field names and their corresponding ComputedFieldInfo objects.

model_config ***: ClassVar[ConfigDict] = {'protected_namespaces': ()}***

Configuration for the model, should be a dictionary conforming to [ConfigDict] [pydantic.config.ConfigDict].

model_fields ***: ClassVar[dict[str, FieldInfo]] = {'model_folder': FieldInfo(annotation=str, required=True), 'model_name': FieldInfo(annotation=str, required=True), 'window_size': FieldInfo(annotation=int, required=True)}***

Metadata about the fields defined on the model, mapping of field names to [FieldInfo] [pydantic.fields.FieldInfo].

This replaces Model._*fields*_ from Pydantic V1.

model_folder ***: str***

model_name ***: str***

window_size ***: int***

async prediction_service.get_data()

Envía los datos almacenados.

Returns: : dict: Datos almacenados.

async prediction_service.health_check()

Verifica el estado del servidor.

Returns: : dict: Estado del servidor.

async prediction_service.receive_data(request: DataRequest)

Recibe y procesa los datos.

Args: : request (DataRequest): Solicitud de datos con los valores y configuración.

Returns: : dict: Mensaje de confirmación.

prediction_service.start_service()

Inicia el servidor de FastAPI.

async prediction_service.update_model(request: ModelUpdateRequest)

Actualiza la información del modelo.

Args: : request (ModelUpdateRequest): Solicitud de actualización del modelo.

Returns: : dict: Mensaje de confirmación.