Projet 5 - Documentation technique

Version 1.0.0

S-S-Zheng

09 janvier 2026

Contents

CHAPTER 1

Routes et Schémas API

Module de définition du router pour les prédictions d'attrition.

Ce module constitue le point d'entrée principal de l'intelligence artificielle. Il orchestre le flux de données complet : réception de la requête, journalisation initiale, vérification de l'existence d'un cache en base de données, exécution de l'inférence par le modèle CatBoost, persistance du résultat et retour de la réponse à l'utilisateur.

app.api.routes.predict.**predict**(*request*, *payload*, *db=Depends(dependency=<function get_db_generator>*, *use_cache=True*, *scope=None)*)

Réalise une prédiction pour un employé donné.

Cette méthode suit la pipeline suivante :

1. **Logging** : Initialisation d'un enregistrement dans "request_logs".
2. **Mise en cache** : Vérification si les caractéristiques ont déjà été traitées (via un hash SHA-256) pour retourner un résultat instantané.
3. **Inférence** : Si nouveau, appel de la méthode predict du modèle chargé.
4. **Persistance** : Sauvegarde des entrées et de la sortie dans "predictions".
5. **Finalisation** : Calcul du temps de réponse et mise à jour du log.

Paramètres

- **request** (*Request*) – Objet requête FastAPI pour accéder au modèle global.
- **payload** (PredictionInput) – Dictionnaire validé contenant les 21 features.
- **db** (*Session*) – Session de base de données injectée par dépendance.

Renvoie

Résultat comprenant la prédiction (0/1), le score de confiance et le nom de la classe.

Type renvoyé

PredictionOutput

Lève

- **HTTPException** – 503 si le modèle n'est pas chargé.
- **HTTPException** – 422 en cas d'erreur de valeur lors de l'inférence.
- **HTTPException** – 500 pour les erreurs serveur imprévues.

Module de définition du router pour l'analyse de l'importance des variables.

Ce module expose un endpoint permettant de comprendre la logique interne du modèle en identifiant les caractéristiques (features) qui ont le plus d'influence sur les prédictions d'attrition. Il fait le pont entre les requêtes HTTP et la méthode de calcul d'importance du modèle CatBoost.

app.api.routes.feature_importance.**feature_importance**(*request*, *top_n=Query(5)*)

Récupère les variables les plus influentes du modèle de prédiction.

Cet endpoint interroge l'instance du modèle chargée en mémoire pour extraire les "n" caractéristiques ayant le poids le plus élevé dans le processus de décision. Cela permet une explicabilité globale du modèle (Global Feature Importance).

Paramètres

- **request** (*Request*) – Objet requête FastAPI utilisé pour accéder au "state" de l'application.
- **top_n** (*int*) – Nombre de variables à retourner, triées par importance décroissante. Doit être supérieur ou égal à 1. Par défaut à 5.

Renvoie

Un objet contenant la liste des tuples (nom_variable, score).

Type renvoyé

FeatureImportanceOutput

Lève

- **HTTPException (503)** – Si l'instance du modèle n'est pas trouvée dans l'état de l'application.
- **HTTPException (422)** – Si les paramètres de requête sont invalides pour le modèle.
- **HTTPException (500)** – En cas d'erreur interne lors du calcul des importances.

Note

Le modèle utilise généralement les valeurs SHAP ou les scores de gain d'information natifs de CatBoost pour classer les variables.

Module de définition du router pour l'accès aux métadonnées du modèle.

Ce module expose un endpoint permettant de consulter les caractéristiques techniques et la configuration du modèle en production. Il fournit une transparence sur les variables attendues, le type d'algorithme et les seuils de décision appliqués lors de l'inférence.

app.api.routes.model_info.**model_info**(*request*)

Récupère les métadonnées et la configuration technique du modèle ML.

Cet endpoint permet de vérifier l'état du modèle chargé en mémoire et d'obtenir des informations cruciales pour l'intégration client, telles que la liste ordonnée des variables (features) et le seuil de classification (threshold) utilisé pour séparer les classes.

Paramètres

request (*Request*) – Objet requête FastAPI permettant d'accéder au "state" global de l'application où réside l'instance du modèle.

Renvoie

Un objet contenant le type de modèle, le nombre de features,
les noms des variables (catégorielles et numériques), les labels de classes et le seuil de décision.

Type renvoyé
ModelInfoOutput

Lève

- **HTTPException (503)** – Si l'instance du modèle n'est pas initialisée dans le state de l'application.
- **HTTPException (422)** – Si une erreur de validation survient lors de la récupération des informations du modèle.
- **HTTPException (500)** – En cas d'erreur interne imprévue sur le serveur.

Module de définition des schémas Pydantic pour l'API.

Ce module définit les contrats d'interface (Data Transfer Objects) pour les requêtes et les réponses de l'API FastAPI. Il inclut une couche de validation métier robuste pour garantir que les données envoyées au modèle CatBoost respectent les plages de valeurs attendues.

Schémas principaux :

- PredictionInput : Données d'entrée pour l'inférence avec validation métier.
- PredictionOutput : Résultat de la prédiction (classe et confiance).
- FeatureImportanceOutput : Liste des variables les plus influentes.
- ModelInfoOutput : Métadonnées et configuration du modèle ML.

Endpoint | Méthode ML |
——————- | ———————— |
/predict | *MLModel.predict* |
/feature-importance | *MLModel.get_feature_importance* |
/health | *ml.model is not None* |
/model-info | attributs du modèle |

class app.api.schemas.**PredictionInput**(** (Keyword-only parameters separator (PEP 3102))*, *features*)

Schéma d'entrée pour la prédiction d'attrition.

Ce modèle valide la présence des 21 caractéristiques obligatoires requises par le modèle CatBoost et applique des contraintes de logique métier (ex: âge entre 18 et 65 ans).

features

classmethod validate_business_logic(*features_dict*)

Applique les règles de validation métier sur les caractéristiques d'entrée.

Vérifie :

1. La présence de tous les champs obligatoires.
2. La cohérence des valeurs numériques (min/max).
3. La validité des catégories pour les variables qualitatives.

Paramètres
features_dict (*dict*) – Dictionnaire brut reçu par l'API.

Lève
ValueError – Si un champ manque ou si une règle métier est violée.

Renvoie
Le dictionnaire validé.

Type renvoyé
dict

model_config = {}
Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

class app.api.schemas.**PredictionOutput**(*, *prediction*, *confidence*, *class_name*)
Schéma de réponse après une prédiction.

prediction

confidence

class_name

model_config = {}
Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

class app.api.schemas.**FeatureImportanceOutput**(*, *top_features*)
Schéma de réponse pour l'importance des variables.

top_features

model_config = {}
Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

class app.api.schemas.**ModelInfoOutput**(*, *model_type*, *n_features*, *feature_names*, *cat_features*, *num_features*, *classes*, *threshold*)
Schéma de réponse détaillant la configuration interne du modèle ML.

model_type

n_features

feature_names

cat_features

num_features

classes

threshold

model_config = {}
Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

class app.api.schemas.**ErrorResponse**(*, *error*, *detail=None*)

Schéma standardisé pour les messages d'erreur de l'API.

error

detail

model_config = {}

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

CHAPTER 2

Modèle Machine Learning

Module wrapper pour le modèle de Machine Learning CatBoost.

Ce module encapsule toute la logique liée au modèle pré-entraîné : chargement des artefacts (modèle compressé, liste des variables, seuil de classification), prétraitement des données d'entrée, inférence avec gestion de seuil personnalisé et analyse de l'importance des caractéristiques (explicabilité).

class app.ml.model.**MLModel**(*model_path=PosixPath('/home/shipoz/Documents/OPENCLASSROOMS/P5/livrable_P5/app/ml/mo*
feature_names_path=PosixPath('/home/shipoz/Documents/OPENCLASSROOMS/P5/livrable_P5/ap
threshold_path=PosixPath('/home/shipoz/Documents/OPENCLASSROOMS/P5/livrable_P5/app/ml/

Classe de gestion du cycle de vie et de l'inférence du modèle CatBoost.

Cette classe sert d'interface entre l'API et le modèle de classification. Elle assure que les données entrantes sont formatées selon l'ordre appris lors de l'entraînement et permet d'appliquer un seuil de décision (threshold) optimisé pour maximiser le score métier (ex: rappel ou F1-score).

model

L'instance du modèle chargé.

Type
CatBoostClassifier

feature_names

Liste ordonnée des variables d'entrée.

Type
List[str]

cat_features

Liste des variables identifiées comme catégorielles.

Type
List[str]

num_features

Liste des variables identifiées comme numériques.

Type
List[str]

`threshold`

Seuil de probabilité pour la classification binaire.

Type
float

`classes`

Noms des classes cibles (["Employé", "Démissionnaire"]).

Type
List[str]

`__init__`(*model_path=PosixPath('/home/shipoz/Documents/OPENCLASSROOMS/P5/livrable_P5/app/ml/model/datas/results/*
feature_names_path=PosixPath('/home/shipoz/Documents/OPENCLASSROOMS/P5/livrable_P5/app/ml/model/datas
threshold_path=PosixPath('/home/shipoz/Documents/OPENCLASSROOMS/P5/livrable_P5/app/ml/model/datas/resu)

Initialise les chemins vers les artefacts du modèle.

Paramètres

- **model_path** (*Path*) – Chemin vers le fichier .cbm (format natif CatBoost).
- **feature_names_path** (*Path*) – Chemin vers le pickle contenant l'ordre des colonnes.
- **threshold_path** (*Path, optional*) – Chemin vers le seuil de décision optimisé.

`load`()

Charge en mémoire le modèle et ses fichiers de configuration associés.

Cette méthode initialise l'objet CatBoost, restaure la liste des features et le seuil. Elle identifie également automatiquement les types de variables (numériques vs catégorielles) via les métadonnées du modèle.

Lève
FileNotFoundError – Si l'un des artefacts critiques est manquant.

`get_model_info`()

Expose les métadonnées techniques du modèle pour l'API.

Renvoie

Dictionnaire contenant le type de modèle, le nombre de features,
les noms des variables par type, les classes et le seuil.

Type renvoyé
dict

Lève
ValueError – Si le modèle n'a pas été chargé préalablement.

`predict`(*features*)

Réalise une inférence à partir d'un dictionnaire de caractéristiques.

Le processus comprend la conversion en DataFrame, la validation de la présence des colonnes, le réordonnancement selon le schéma d'entraînement et l'application du seuil de décision.

Paramètres
features (*Dict[str, any]*) – Dictionnaire des variables d'entrée.

Renvoie

Un tuple contenant :

- prediction (int): 0 ou 1.
- confidence (float): Score de probabilité (0.0 à 1.0).
- class_name (str): Label humain de la prédiction.

Type renvoyé
Tuple[int, float, str]

Lève
ValueError – Si le modèle est absent ou si les features fournies ne correspondent pas à l'attendu.

`get_feature_importance`(*top_n=5*)

Calcule l'importance des variables (Global Feature Importance).

Paramètres
top_n (*int*) – Nombre de variables les plus influentes à retourner.

Renvoie

Liste de tuples (nom_variable, importance)
triée par importance décroissante.

Type renvoyé
List[Tuple[str, float]]

CHAPTER 3

Base de Données

3.1 Modèles de Données

Module de définition des modèles de données ORM (Object-Relational Mapping).

Ce module contient les schémas SQL pour PostgreSQL via SQLAlchemy. Il définit l'organisation des données stockées, incluant les enregistrements de prédictions détaillés et le système de journalisation (logging) pour la traçabilité des requêtes.

class app.db.models_db.**RequestLog**(***kwargs*)

Modèle représentant les logs d'activité de l'API.

Stocke les métadonnées de chaque requête entrante pour permettre l'audit de performance et la traçabilité des erreurs. Chaque log peut être lié à un enregistrement de prédiction spécifique.

id

Clé primaire auto-incrémentée.

Type
int

created_at

Horodatage de la requête (géré par le serveur SQL).

Type
datetime

endpoint

Le point d'entrée API sollicité (ex: "/predict").

Type
str

status_code

Le code de statut HTTP retourné (ex: 200, 422, 500).

Type
int

response_time_ms

Temps de traitement de la requête en millisecondes.

Type
float

prediction_id

Clé étrangère pointant vers l'ID unique de la prédiction associée.

Type
str

prediction_record

Relation ORM vers l'objet PredictionRecord correspondant.

Type
relationship

id

created_at

endpoint

status_code

response_time_ms

prediction_id

prediction_record

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class app.db.models_db.**PredictionRecord**(***kwargs*)

Modèle représentant une prédiction stockée et ses caractéristiques d'entrée.

Cette table contient l'ensemble des variables métier (features) envoyées par l'utilisateur, le résultat du modèle ML, et une version sérialisée (JSONB) pour la flexibilité. L'ID est un hash SHA-256 des entrées servant de mécanisme de dédoublonnage.

id

Hash unique des features servant de clé primaire.

Type
str

created_at

Date d'enregistrement.

Type
datetime

age

Âge de l'employé.

Type
int

genre

Genre (m/f).

Type
str

revenu_mensuel

Revenu mensuel.

Type
int

 # ...

Type
autres colonnes de caractéristiques métier

a_quitte_l_entreprise

Valeur réelle observée

Type
int

(utilisée pour le réentraînement/historique).

inputs

Copie de sauvegarde de l'intégralité des entrées au format JSON.

Type
JSONB

prediction

Classe prédite par le modèle (0 ou 1).

Type
int

confidence

Score de probabilité associé à la prédiction.

Type
float

class_name

Traduction textuelle de la classe (ex: "Employé", "Démissionnaire").

Type
str

model_version

Version du modèle utilisé lors de l'inférence.

Type
str

logs

Liste des logs de requêtes ayant sollicité cette prédiction précise.

Type
relationship

id

created_at

age

genre

revenu_mensuel

statut_marital

poste

annees_dans_le_poste_actuel

heure_supplementaires

augementation_salaire_precedente

nombre_participation_pee

nb_formations_suivies

distance_domicile_travail

niveau_education

domaine_etude

frequence_deplacement

evolution_note

stagnation_promo

freq_chgt_poste

revenu_mensuel_ajuste_par_nv_hierarchique

revenu_mensuel_par_annee_xp

freq_chgt_responsable

satisfaction_globale_employee

a_quitte_l_entreprise

inputs

prediction

confidence

class_name

model_version

logs

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

3.2 Scripts et Actions

Note

Les modèles techniques sont détaillés dans la section Modèles.

Module de persistance des prédictions dans la base de données.

Ce module assure l'enregistrement des résultats d'inférence, la gestion de la concurrence pour éviter les doublons accidentels et la traçabilité des opérations en liant les enregistrements aux logs de requêtes.

app.db.actions.save_prediction_to_db.**save_prediction**(*db*, *features*, *pred_data*, *log_id*)

Enregistre une nouvelle prédiction et la lie à un log d'activité.

Cette fonction effectue les étapes suivantes : 1. Calcule un hash SHA-256 unique à partir des features d'entrées (la requete). 2. Vérifie une dernière fois l'existence de l'ID (protection contre les "race conditions"). 3. Si nouveau, crée un enregistrement "PredictionRecord" en dépaquetant les features. 4. Établit une relation de traçabilité entre la prédiction et le log via "link_log".

Paramètres

- **db** (*Session*) – Session SQLAlchemy active pour les transactions.
- **features** (*dict*) – Dictionnaire des variables d'entrée utilisées pour la prédiction.
- **pred_data** (*tuple*) – Un tuple contenant (valeur_prediction, confiance, nom_classe). Exemple : (1, 0.85, « Démissionnaire »).
- **log_id** (*int*) – Identifiant unique de la ligne de log (RequestLog) à l'origine de l'action.

Renvoie

L'identifiant unique (hash hexadécimal) de la requête enregistrée.

Type renvoyé

str

Lève

Exception – Relance toute exception survenant lors de la transaction après avoir effectué un rollback pour maintenir l'intégrité de la base.

Note

L'utilisation de *db.flush()* permet d'envoyer l'objet à la base de données pour obtenir une confirmation sans pour autant clore la transaction globale (permettant le rollback en cas d'échec de la liaison des logs).

Module de gestion des doublons dans la base de données.

Ce module permet de vérifier l'existence d'une prédiction avant toute nouvelle insertion afin de garantir l'intégrité des données et d'optimiser les performances en évitant les calculs redondants.

app.db.actions.get_prediction_from_db.**get_prediction**(*db*, *features*)

Recherche une prédiction existante basée sur le hash des caractéristiques.

Cette fonction calcule un identifiant unique (SHA-256) à partir des "features" fournies, puis interroge la table "PredictionRecord" pour voir si cet ID est déjà présent. Cela sert de mécanisme de cache et de protection contre les doublons.

Paramètres

- **db** (*Session*) – La session active de la base de données SQLAlchemy.
- **features** (*dict*) – Le dictionnaire contenant les caractéristiques de l'employé (ex: age, salaire, poste, etc.).

Renvoie

Retourne l'objet de prédiction complet si trouvé,
sinon None si aucune requête identique n'a été enregistrée.

Type renvoyé
PredictionRecord | None

Exemple

```
>>> result = get_prediction(db_session, {"age": 30, "salaire": 5000})
>>> if result:
...     print(f"Prédiction trouvée : {result.prediction}")
```

Module d'importation de données historiques depuis un fichier CSV vers PostgreSQL.

Ce module permet d'étoffer la base de données avec des datasets pré-existants. Il intègre une logique de dédoublonnage par hachage (SHA-256) pour éviter les insertions redondantes et assure la traçabilité de l'opération via le système de logging applicatif.

app.db.import_dataset_to_db.**import_csv**(*file_path*)

Lit un fichier CSV et importe les enregistrements uniques dans la table PredictionRecord.

Le processus suit les étapes suivantes : 1. Chargement du fichier via Pandas et conversion des NaN en "None" pour compatibilité JSON. 2. Initialisation d'un log d'activité pour l'endpoint virtuel "/import". 3. Pour chaque ligne : extraction de la target, génération d'un hash ID unique sur les features. 4. Vérification de l'existence de l'ID en base pour ignorer les doublons. 5. Construction et insertion massive (bulk insert) des nouveaux enregistrements.

Paramètres
file_path (*str*) – Chemin local vers le fichier CSV contenant les données historiques.

Lève
Exception – En cas d'erreur de lecture, de hachage ou de contrainte d'intégrité SQL, une annulation (rollback) est effectuée.

Note

- La "confidence" est fixée à 1.0 car il s'agit de données historiques observées.
- Le statut HTTP 201 est loggé en cas de succès avec insertion.

- Le statut HTTP 204 est loggé si aucun nouvel enregistrement n'a été trouvé.

CHAPTER 4

Utilities

Module de génération d'identifiants uniques par hachage cryptographique.

Ce module fournit une fonction permettant de transformer un dictionnaire de caractéristiques (features) en une empreinte numérique unique (ID). Ce mécanisme est la pierre angulaire du système pour : 1. Identifier de manière déterministe chaque profil d'employé. 2. Éviter les doublons lors des imports de données historiques. 3. Implémenter un système de cache pour les requêtes API en temps réel.

app.utils.hash_id.**generate_feature_hash**(*features*)

Génère un identifiant unique (SHA-256) à partir d'un dictionnaire de caractéristiques.

Le processus garantit le déterminisme (le même dictionnaire produit toujours le même hash) en suivant deux étapes critiques : 1. Tri alphabétique des clés du dictionnaire pour neutraliser l'ordre d'insertion. 2. Sérialisation JSON et encodage en UTF-8 avant le passage dans l'algorithme SHA-256.

Paramètres
features (*dict*) – Le dictionnaire contenant les variables d'entrée de la requête.

Renvoie
Une chaîne de 64 caractères hexadécimaux représentant l'empreinte unique.

Type renvoyé
str

Exemple

```
>>> features = {"age": 30, "poste": "Manager"}
>>> generate_feature_hash(features)
'7a1b...8f2e'
```

Module de gestion de la journalisation (logging) en base de données.

Ce module fournit les outils nécessaires pour suivre le cycle de vie d'une requête HTTP. Il permet de mesurer la performance (temps de réponse), de capturer les codes de statut et d'établir un lien de parenté entre un log technique et une prédiction métier. Ces données sont essentielles pour le monitoring et l'audit du service.

app.utils.logger_db.**init_log**(*db*, *endpoint*)

Initialise une entrée de log dans la table “request_logs”.

Crée l’objet de log au début de la requête. L’utilisation de *db.flush()* permet de récupérer l’ID auto-incrémenté généré par PostgreSQL sans pour autant clore la transaction SQL, permettant ainsi d’associer cet ID à d’autres opérations.

Paramètres

- **db** (*Session*) – Session SQLAlchemy active.
- **endpoint** (*str*) – Le chemin de l’URL sollicité (ex: “/predict”).

Renvoie
L’instance du log nouvellement créée.

Type renvoyé
RequestLog

app.utils.logger_db.**closing_log**(*db*, *log_obj*, *start_time*, *status_code=None*, *prediction_id=None*)

Finalise et persiste le log en base de données.

Cette fonction calcule la latence totale de la requête, met à jour le code de statut final (200, 422, 500, etc.) et valide la transaction (commit).

Paramètres

- **db** (*Session*) – Session SQLAlchemy active.
- **log_obj** (*RequestLog*) – L’objet log initialisé par *init_log*.
- **start_time** (*float*) – Le timestamp de début (provenant de *time.time()*).
- **status_code** (*int, optional*) – Le code HTTP final. Si None, conserve la valeur initiale.
- **prediction_id** (*str, optional*) – L’identifiant unique (hash) de la prédiction associée.

app.utils.logger_db.**link_log**(*db*, *log_id*, *prediction_id*)

Établit un lien a posteriori entre un log technique et un enregistrement métier.

Cette fonction est utile car elle garantie l’intégrité de la relation entre les tables “request_logs” et “predictions”.

Paramètres

- **db** (*Session*) – Session SQLAlchemy active.
- **log_id** (*int*) – Identifiant numérique du log.
- **prediction_id** (*str*) – Hash SHA-256 de la prédiction.

CHAPTER 5

Indices et tables

- genindex
- modindex
- search

Index des modules Python

a

Index

Symboles

A

C

D

E

F

G

I

L

M

N

P

T

V