### Explications des fichiers pour la préparation à l'examen

#### ****Fichier**** api.pyapi.papi.pyapiaaaqqa****(API FastAPI)api.py****

****Essentiel à savoir :****

1. ****Fonction principale :****

* Offre un endpoint /predic /predict qui accepte un fichier CSV via une requête POST, effectue une prédiction de faux billets à l'aide d'un modèle Random Forest, et retourne les résultats avec des statistiques.
* Utilise joblib joblib pour charger le modèle (random\_forest\_model.savrandom\_forest\_model.sav) et le scaler (scaler.savscaler.sav).

1. ****Points clés :****

****Validation des données :**** Vérifie que le CSV contient les colonnes requises diagonal,height-left agonal, height\_lediagonal, height\_leddetc.).

****Gestion des erreurs :**** Logging détaillé et réponses HTTP explicites (400 pour les erreurs client, 500 pour les serveur).

****Format de réponse :****

Prédictions individuelles (ID, "Genuine"/"Fake", probabilité).

Statistiques globales (nombre total, pourcentages).

1. ****Technologies :****

FastAPI pour la création de l'API.

Pydantic pour la validation des modèles de réponse(PredictionResponse, StatsResult).

****Questions possibles :****

* Comment gérez-vous les erreurs de validation des données ?  
  → Réponse : Via HTTPExceptionHH HTTPException avec un statut 400 et un message listant les colonnes manquantes.
* Pourquoi convertir les types NumPy en types Python natifs ?  
  → Réponse : Pour assurer la compatibilité avec JSON (FastAPI sérialise les réponses en JSON).

#### ****Fichier****streamlit.py****(Application Streamlit)****

****Essentiel à savoir :****

1. ****Fonctionnalités :****

* Interface utilisateur pour uploader un CSV, afficher un aperçu des données, et lancer la détection.
* Affichage des résultats sous forme de cartes stylisées ("Vrai"/"Faux") avec images, probabilités, et statistiques.
* Visualisation avec un graphique Plotly (camembert ou barres).

1. ****Points clés :****

****CSS personnalisé :**** Utilise des classes pour styliser les cartes (ex:is-genuine card .genuine-card avec bordure verte).

****Optimisation :**** Cache le modèle avec@st.cache\_resource @st.cache\_resource pour éviter de le recharger à chaque interaction.

****Gestion des fichiers :****

Lit le CSV avec pd.read-csvpd.read\_csp  pd.read\_csv (séparateur ;;:);.

Vérifie la présence des colonnes requises avant prédiction.

1. ****Images :****

Convertit les images (vraibillet.PNG, fauxbillet.png) en base64 pour les afficher dans l'interface.

****Questions possibles :****

* Comment êtes-vous assuré que le modèle n'est chargé qu'une seule fois ?  
  → Réponse : Grâce au décorateur @st.cache\_resource.@st.cache\_resource
* Comment gérez-vous l'affichage conditionnel des résultats ("Vrai" vs "Faux") ?  
  → Réponse : Via des blocs HTML/CSS dynamiques (genuine-card/fake-card)genuine-card/ genuine et fake card fake-c générés avec st.markdown. st.markdown

#### ****Déploiement sur Streamlit Cloud****

****Étapes clés :****

1. ****Prérequis :****

Un compte Streamlit Cloud (lié à GitHub).

Fichiers dans un dépôt GitHub (api.py, streamlit.py, api.py, streamlit.py,modèles, images).

1. ****Configuration :****

****Fichier requirement****requirements.txre

****Déploiement :****

* Sur Streamlit Cloud, cliquer sur "New app" → Sélectionner le dépôt et le fichier streamlit.py.
* Streamlit Cloud détecte automatiquement les dépendances et déploie l'app.