**Rapport de Plateforme de Suivi des Performances Sportives avec Visualisation et Exportation**

**Sohail Charef / GLSID 1 / Groupe A**

**Abdellhadi El Omari / BDCC 1 / Groupe A**

**Les bibliothèques utilisée :**

import streamlit as st

import pandas as pd

import matplotlib.pyplot as plt

from fpdf import FPDF

import os

from hashlib import sha256

from datetime import datetime

import requests

from bs4 import BeautifulSoup

**Creation des fichiers data:**

# Constantes pour les fichiers

USER\_DATA\_FILE = "users.csv"

BASE\_DIR = "user\_data"  # Répertoire pour stocker les données des utilisateurs

# Initialisation des fichiers et répertoires

def initialize():

    if not os.path.exists(USER\_DATA\_FILE):

        pd.DataFrame(columns=["Username", "PasswordHash"]).to\_csv(USER\_DATA\_FILE, index=False)

    if not os.path.exists(BASE\_DIR):

        os.makedirs(BASE\_DIR)

**Hachage des mots de passe pour plus de sécurité**

def hash\_password(password):

    return sha256(password.encode()).hexdigest()

**Gestion des comptes utilisateurs**

def create\_account():

    st.title("Créer un compte")

    username = st.text\_input("Nom d'utilisateur")

    password = st.text\_input("Mot de passe", type="password")

    confirm\_password = st.text\_input("Confirmez le mot de passe", type="password")

    if st.button("Créer un compte"):

        if not username or not password:

            st.warning("Veuillez remplir tous les champs.")

        elif password != confirm\_password:

            st.error("Les mots de passe ne correspondent pas.")

        else:

            users = pd.read\_csv(USER\_DATA\_FILE)

            if username in users["Username"].values:

                st.error("Ce nom d'utilisateur est déjà pris.")

            else:

                password\_hash = hash\_password(password)

                users = pd.concat([users, pd.DataFrame([{"Username": username, "PasswordHash": password\_hash}])], ignore\_index=True)

                users.to\_csv(USER\_DATA\_FILE, index=False)

                os.makedirs(os.path.join(BASE\_DIR, username))

                st.success("Compte créé avec succès !")

**Login**

def login():

    st.title("Connexion")

    st.subheader("Veuillez entrer vos identifiants")

    username = st.text\_input("Nom d'utilisateur")

    password = st.text\_input("Mot de passe", type="password")

    if st.button("Se connecter"):

        if not username or not password:

            st.warning("Veuillez remplir tous les champs.")

        else:

            users = pd.read\_csv(USER\_DATA\_FILE)

            if username in users["Username"].values:

                user\_data = users[users["Username"] == username].iloc[0]

                if hash\_password(password) == user\_data["PasswordHash"]:

                    st.session\_state["logged\_in"] = True

                    st.session\_state["username"] = username

                    st.session\_state["page"] = "Navigation"  # Rediriger après connexion

                    st.success(f"Bienvenue, {username} !")

                else:

                    st.error("Mot de passe incorrect.")

            else:

                st.error("Utilisateur non trouvé.")

**Charger les données spécifiques à l'utilisateur**

def load\_user\_data():

    username = st.session\_state["username"]

    user\_file = os.path.join(BASE\_DIR, username, "data.csv")

    if os.path.exists(user\_file):

        return pd.read\_csv(user\_file)

    else:

        return pd.DataFrame(columns=["Date", "Distance (km)", "Temps (min)", "Calories (kcal)", "FC Moyenne (bpm)"])

**Sauvegarder les données spécifiques à l'utilisateur**

def save\_user\_data(data):

    username = st.session\_state["username"]

    user\_file = os.path.join(BASE\_DIR, username, "data.csv")

    data.to\_csv(user\_file, index=False)

**Saisie d'une nouvelle session**

def add\_session():

    st.title("Ajouter une nouvelle session de course")

    st.subheader("Veuillez entrer les détails de votre session")

    with st.form("new\_session"):

        date = st.date\_input("Date", value=datetime.now())

        distance = st.number\_input("Distance (km)", min\_value=0.0, step=0.1)

        time = st.number\_input("Temps (min)", min\_value=0.0, step=0.1)

        calories = st.number\_input("Calories (kcal)", min\_value=0.0, step=1.0)

        heart\_rate = st.number\_input("Fréquence cardiaque moyenne (bpm)", min\_value=0, step=1)

        submitted = st.form\_submit\_button("Enregistrer")

    if submitted:

        new\_entry = {

            "Date": date,

            "Distance (km)": distance,

            "Temps (min)": time,

            "Calories (kcal)": calories,

            "FC Moyenne (bpm)": heart\_rate,

        }

        data = load\_user\_data()

        data = pd.concat([data, pd.DataFrame([new\_entry])], ignore\_index=True)

        save\_user\_data(data)

        st.success("Session enregistrée avec succès !")

**Visualisation des données**

def visualize\_data():

    st.title("Visualisation des progrès")

    st.subheader("Graphiques interactifs des performances")

    data = load\_user\_data()

    if data.empty:

        st.warning("Aucune donnée à visualiser.")

        return

    start\_date = pd.to\_datetime(st.date\_input("Date de début", value=pd.to\_datetime(data["Date"]).min()))

    end\_date = pd.to\_datetime(st.date\_input("Date de fin", value=pd.to\_datetime(data["Date"]).max()))

    filtered\_data = data[(pd.to\_datetime(data["Date"]) >= start\_date) & (pd.to\_datetime(data["Date"]) <= end\_date)]

    if not filtered\_data.empty:

        fig, ax = plt.subplots(3, 1, figsize=(12, 10))

        fig.suptitle("Analyse des performances au fil du temps", fontsize=16, fontweight='bold')

        # Distance cumulée

        ax[0].plot(filtered\_data["Date"], filtered\_data["Distance (km)"], label="Distance (km)", marker='o')

        ax[0].set\_title("Distance parcourue (en km)", fontsize=12)

        ax[0].set\_ylabel("Kilomètres", fontsize=10)

        ax[0].grid(True)

        ax[0].legend()

        # Temps moyen par kilomètre

        filtered\_data["Temps/km (min)"] = filtered\_data["Temps (min)"] / filtered\_data["Distance (km)"]

        ax[1].plot(filtered\_data["Date"], filtered\_data["Temps/km (min)"], label="Temps moyen (min/km)", color="orange", marker='o')

        ax[1].set\_title("Temps moyen par kilomètre", fontsize=12)

        ax[1].set\_ylabel("Minutes par km", fontsize=10)

        ax[1].grid(True)

        ax[1].legend()

        # Calories brûlées

        ax[2].plot(filtered\_data["Date"], filtered\_data["Calories (kcal)"], label="Calories brûlées", color="green", marker='o')

        ax[2].set\_title("Calories brûlées (en kcal)", fontsize=12)

        ax[2].set\_ylabel("Calories", fontsize=10)

        ax[2].grid(True)

        ax[2].legend()

        # Rotation des étiquettes des dates

        for axis in ax:

            axis.set\_xticks(filtered\_data["Date"])

            axis.set\_xticklabels(filtered\_data["Date"], rotation=45, ha='right', fontsize=9)

        plt.tight\_layout(rect=[0, 0, 1, 0.95])  # Ajustement pour le titre global

        st.pyplot(fig)

    else:

        st.warning("Aucune donnée pour cette période.")

**Fonction de scraping pour récupérer des benchmarks**

def scrape\_articles():

    st.title("Lire Des articles")

    url = "https://www.runnersworld.com"  # Remplacer par un autre site si nécessaire

    response = requests.get(url)

    contenu = BeautifulSoup(response.text, 'html.parser')

    # Extraction des titres des articles

    titles\_args = contenu.find\_all('span', class\_='css-1hc7p2m e10ip9lg5')

    titles = [title.text.strip() for title in titles\_args]

    # Extraction des liens des articles

    link\_args = contenu.find\_all('a', class\_='ee4ms352 css-mg2r4i e1c1bym14')

    links = [url + link['href'] for link in link\_args]

    # Extraction des images des articles

    pic\_args = contenu.find\_all('img', class\_='css-0 e1g79fud0')

    pics = [pic['src'] for pic in pic\_args]

    # Afficher les articles avec les titres, images et liens

    for i in range(0, len(titles), 3):

        cols = st.columns(3)

        for j in range(3):

            if i + j < len(titles):

                with cols[j]:

                    st.subheader(titles[i + j])

                    st.write(f"[Lire l'article]({links[i + j]})")

                    # Afficher l'image

                    if pics[i + j].startswith('http'):

                        img\_url = pics[i + j]

                    else:

                        img\_url = url + pics[i + j]

                    st.image(img\_url, width=300)

**Calculer les statistiques pour la période choisie**

def calculate\_statistics(data, start\_date, end\_date):

    filtered\_data = data[(pd.to\_datetime(data["Date"]) >= start\_date) & (pd.to\_datetime(data["Date"]) <= end\_date)]

    if not filtered\_data.empty:

        total\_distance = filtered\_data["Distance (km)"].sum()

        total\_time = filtered\_data["Temps (min)"].sum()

        total\_calories = filtered\_data["Calories (kcal)"].sum()

        avg\_heart\_rate = filtered\_data["FC Moyenne (bpm)"].mean()

        avg\_speed = total\_distance / (total\_time / 60) if total\_time > 0 else 0

        statistics = {

            "Total Distance (km)": total\_distance,

            "Total Time (min)": total\_time,

            "Total Calories (kcal)": total\_calories,

            "Average Heart Rate (bpm)": avg\_heart\_rate,

            "Average Speed (km/h)": avg\_speed

        }

    else:

        statistics = {}

    return statistics, filtered\_data

**Fonction pour générer le PDF**

def generate\_pdf(statistics, filtered\_data, username):

    # Créer le PDF

    pdf = FPDF()

    pdf.add\_page()

    # Ajouter un titre

    pdf.set\_font("Arial", 'B', 16)

    pdf.cell(200, 10, f"Rapport des Performances de {username}", ln=True, align="C")

    pdf.ln(10)

    # Ajouter les statistiques

    pdf.set\_font("Arial", '', 12)

    for key, value in statistics.items():

        pdf.cell(200, 10, f"{key}: {value}", ln=True)

    pdf.ln(10)

    # Ajouter un tableau avec les données détaillées

    pdf.cell(200, 10, "Données des sessions", ln=True, align="C")

    pdf.ln(5)

    # Ajouter l'entête du tableau

    pdf.set\_font("Arial", 'B', 10)

    pdf.cell(40, 10, "Date", border=1)

    pdf.cell(40, 10, "Distance (km)", border=1)

    pdf.cell(40, 10, "Temps (min)", border=1)

    pdf.cell(40, 10, "Calories (kcal)", border=1)

    pdf.cell(40, 10, "FC Moyenne (bpm)", border=1)

    pdf.ln()

    # Ajouter les lignes du tableau

    pdf.set\_font("Arial", '', 10)

    for index, row in filtered\_data.iterrows():

        pdf.cell(40, 10, str(row["Date"]), border=1)

        pdf.cell(40, 10, str(row["Distance (km)"]), border=1)

        pdf.cell(40, 10, str(row["Temps (min)"]), border=1)

        pdf.cell(40, 10, str(row["Calories (kcal)"]), border=1)

        pdf.cell(40, 10, str(row["FC Moyenne (bpm)"]), border=1)

        pdf.ln()

    # Sauvegarder le PDF

    output\_pdf\_path = f"{username}\_performance\_report.pdf"

    pdf.output(output\_pdf\_path)

    return output\_pdf\_path

**Fonction pour générer et télécharger le PDF**

def download\_pdf():

    st.title("Générer un PDF de vos performances")

    data = load\_user\_data()

    if data.empty:

        st.warning("Aucune donnée disponible pour générer le PDF.")

        return

    start\_date = pd.to\_datetime(st.date\_input("Date de début", value=pd.to\_datetime(data["Date"]).min()))

    end\_date = pd.to\_datetime(st.date\_input("Date de fin", value=pd.to\_datetime(data["Date"]).max()))

    # Calculer les statistiques pour la période

    statistics, filtered\_data = calculate\_statistics(data, start\_date, end\_date)

    if not statistics:

        st.warning("Aucune donnée pour la période sélectionnée.")

        return

    # Générer le PDF

    pdf\_path = generate\_pdf(statistics, filtered\_data, st.session\_state["username"])

    st.success("Le PDF a été généré avec succès !")

    # Ajouter un bouton pour télécharger le PDF

    with open(pdf\_path, "rb") as f:

        st.download\_button("Télécharger le PDF", f, file\_name=pdf\_path)

**Application principale**

def main():

    initialize()

    if "logged\_in" not in st.session\_state:

        st.session\_state["logged\_in"] = False

        st.session\_state["page"] = "Accueil"

    if not st.session\_state["logged\_in"]:

        choice = st.sidebar.radio("Navigation", ["Se connecter", "Créer un compte"])

        if choice == "Créer un compte":

            create\_account()

        elif choice == "Se connecter":

            login()

    else:

        if st.session\_state["page"] == "Navigation":

            st.sidebar.title("Navigation")

            choice = st.sidebar.radio("Choisissez une option", ["Ajouter une session", "Visualiser les données", "Lire des articles", "Générer un PDF", "Se déconnecter"])

            if choice == "Ajouter une session":

                add\_session()

            elif choice == "Visualiser les données":

                visualize\_data()

            elif choice == "Lire des articles":

                scrape\_articles()

            elif choice == "Générer un PDF":

                download\_pdf()

            elif choice == "Se déconnecter":

                st.session\_state["logged\_in"] = False

                st.session\_state["page"] = "Accueil"

                st.session\_state.pop("username", None)

                st.success("Déconnexion réussie.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()