import os

import re

import subprocess

import argparse

# Configuration based on user input

# The script assumes it is run from the 'mobile-app/scripts' directory

FLUTTER\_PROJECT\_ROOT\_RELATIVE = ".."

APP\_THEME\_RELATIVE\_PATH = "lib/theme/app\_theme.dart"

PACKAGE\_NAME = "kdp\_creator\_suite"

# The correct package import path is without 'lib/'

APP\_THEME\_IMPORT = f"import 'package:{PACKAGE\_NAME}/theme/app\_theme.dart';"

def get\_project\_root(script\_path):

"""Calculates the absolute path to the Flutter project root."""

# Assuming script is in mobile-app/scripts and project root is mobile-app

return os.path.abspath(os.path.join(script\_path, FLUTTER\_PROJECT\_ROOT\_RELATIVE))

def scan\_directory(project\_root):

"""Scans all Dart files for the AppTheme import."""

already\_defined = []

not\_defined = []

# os.walk will start from the project root

for root, \_, files in os.walk(project\_root):

for file in files:

if file.endswith('.dart'):

filepath = os.path.join(root, file)

# Skip the app\_theme.dart file itself

if filepath == os.path.join(project\_root, APP\_THEME\_RELATIVE\_PATH):

continue

with open(filepath, 'r', encoding='utf-8') as f:

content = f.read()

if APP\_THEME\_IMPORT in content:

already\_defined.append((file, filepath))

else:

not\_defined.append((file, filepath))

return already\_defined, not\_defined

def fix\_file(filepath):

"""Inserts the AppTheme import into the specified file."""

with open(filepath, 'r', encoding='utf-8') as f:

content = f.readlines()

insert\_index = 0

for i, line in enumerate(content):

# Find the best place to insert the import statement (after other imports, library, or part)

if line.strip().startswith('import '):

insert\_index = i + 1

elif line.strip().startswith('library '):

insert\_index = i + 1

elif line.strip().startswith('part '):

insert\_index = i + 1

# Insert the import statement

new\_content = content[:insert\_index] + [APP\_THEME\_IMPORT + '\n'] + content[insert\_index:]

with open(filepath, 'w', encoding='utf-8') as f:

f.writelines(new\_content)

return True

def check\_dart\_syntax(relative\_filepath, project\_root):

"""Runs flutter analyze on a single file from the project root."""

try:

# Use the full path to the flutter executable to ensure it's found

flutter\_executable = os.path.join(os.path.expanduser("~"), "flutter", "bin", "flutter")

# Run flutter analyze from the project root (cwd=project\_root)

process = subprocess.run([flutter\_executable, "analyze", relative\_filepath],

cwd=project\_root, capture\_output=True, text=True, check=False)

# Flutter analyze returns non-zero exit code for errors/warnings

if process.returncode != 0 and "No issues found!" not in process.stdout:

# Return False and the combined output for error details

return False, process.stdout.strip() + process.stderr.strip()

return True, "No syntax issues found."

except FileNotFoundError:

return False, "'flutter' command not found. Ensure Flutter SDK is installed and in PATH."

except Exception as e:

return False, f"Error running flutter analyze: {e}"

def main():

parser = argparse.ArgumentParser(description='Scan and fix AppTheme imports in a Flutter project.')

parser.add\_argument('--yes', '-y', action='store\_true', help='Automatically answer yes to all prompts.')

args = parser.parse\_args()

script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

project\_root = get\_project\_root(script\_dir)

print(f"Flutter project root: {project\_root}")

print(f"Scanning directory: {project\_root}")

# --- Step 2 Part 1: Check app\_theme.dart currency ---

app\_theme\_filepath = os.path.join(project\_root, APP\_THEME\_RELATIVE\_PATH)

if not os.path.exists(app\_theme\_filepath):

print(f"Error: app\_theme.dart not found at {app\_theme\_filepath}. Cannot proceed.")

return

print(f"\nChecking currency of {APP\_THEME\_RELATIVE\_PATH}...")

app\_theme\_relative\_filepath = os.path.relpath(app\_theme\_filepath, project\_root)

is\_current, currency\_message = check\_dart\_syntax(app\_theme\_relative\_filepath, project\_root)

if not is\_current:

print(f"Warning: {APP\_THEME\_RELATIVE\_PATH} might not be current or has syntax errors:\n{currency\_message}")

if not args.yes:

response = input("Do you want to continue scanning and fixing despite this warning? (yes/no): ").lower()

if response != 'yes':

print("Operation aborted by user.")

return

else:

print(f"{APP\_THEME\_RELATIVE\_PATH} appears to be syntactically correct and current.")

# --- Step 1: Initial Scan and Categorization ---

already\_defined, not\_defined = scan\_directory(project\_root)

print(f"\nNumber of files with AppTheme defined: {len(already\_defined)}")

print(f"Number of files with AppTheme not defined: {len(not\_defined)}")

# --- Step 3 Part 1: Check syntax for ALL Dart files (including already\_defined) ---

all\_dart\_files = already\_defined + not\_defined

syntax\_errors\_found = []

print("\nPerforming syntax validation on all Dart files...")

for filename, filepath in all\_dart\_files:

relative\_filepath = os.path.relpath(filepath, project\_root)

is\_valid, error\_details = check\_dart\_syntax(relative\_filepath, project\_root)

# We ignore the 'unused\_import' warning for the defined\_widget.dart in this check

if not is\_valid and "unused\_import" not in error\_details:

syntax\_errors\_found.append((filename, filepath, error\_details))

if syntax\_errors\_found:

print(f"\n{len(syntax\_errors\_found)} files have non-trivial syntax errors or warnings:")

for filename, filepath, errors in syntax\_errors\_found:

print(f"- {filename} ({filepath}):\n{errors}")

if not args.yes:

response = input("Do you want to proceed with fixing AppTheme imports despite syntax errors? (yes/no): ").lower()

if response != 'yes':

print("Operation aborted by user.")

return

else:

print("No general syntax issues found in Dart files.")

# --- Step 2 Part 2: Fix 'not defined' files ---

if not\_defined:

print("\nFiles where AppTheme is NOT defined:")

for filename, filepath in not\_defined:

print(f"- {filename}: {filepath}")

response = 'yes' if args.yes else input("\nWould you like to fix the files in the 'not defined' list? (yes/no): ").lower()

if response == 'yes':

defined\_fixed = []

for filename, filepath in not\_defined:

try:

if fix\_file(filepath):

defined\_fixed.append((filename, filepath))

print(f"Fixed: {filename}")

except Exception as e:

print(f"Error fixing {filename}: {e}")

print(f"\nNumber of files fixed: {len(defined\_fixed)}")

print("Files fixed:")

for filename, filepath in defined\_fixed:

print(f"- {filename}: {filepath}")

# --- Step 3 Part 2: Recheck entire directory ---

print("\nRechecking directory for files still not defined...")

\_, files\_still\_not\_defined\_after\_fix = scan\_directory(project\_root)

# --- Step 4: Compare lists and display results ---

# Files that were in the original 'not\_defined' list and are STILL 'not defined' after the fix attempt.

original\_not\_defined\_paths = {fp for \_, fp in not\_defined}

truly\_still\_not\_defined = []

for filename, filepath in files\_still\_not\_defined\_after\_fix:

if filepath in original\_not\_defined\_paths:

truly\_still\_not\_defined.append((filename, filepath))

print(f"\nNumber of files still not defined after fix attempt: {len(truly\_still\_not\_defined)}")

if truly\_still\_not\_defined:

print("Files still not defined:")

for filename, filepath in truly\_still\_not\_defined:

print(f"- {filename}: {filepath}")

else:

print("All previously 'not defined' files are now fixed.")

if truly\_still\_not\_defined:

print(f"\nSummary: {len(truly\_still\_not\_defined)} files were initially 'not defined' and remain unfixed after the attempt.")

for filename, filepath in truly\_still\_not\_defined:

print(f"- {filename}: {filepath}")

else:

print("\nAll files initially identified as 'not defined' were successfully fixed or remained fixed.")

else: # User chose not to fix

output\_filename = os.path.join(script\_dir, "not\_defined\_files.txt")

with open(output\_filename, 'w', encoding='utf-8') as f:

for filename, filepath in not\_defined:

f.write(f"{filename}: {filepath}\n")

print(f"\nList of 'not defined' files saved to {output\_filename}")

else:

print("\nNo files found where AppTheme is not defined. No action needed.")

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

main()