import 'package:desktop\_drop/desktop\_drop\_web.dart';import 'package:file\_picker/\_internal/file\_picker\_web.dart';import 'package:flutter\_web\_plugins/flutter\_web\_plugins.dart';void registerPlugins(Registrar registrar) {  DesktopDropWeb.registerWith(registrar);  FilePickerWeb.registerWith(registrar);  registrar.registerMessageHandler();}import 'dart:io';import 'dart:math';import 'dart:typed\_data';  //这个必须引入，因为用到了Fileimport 'package:archive/archive\_io.dart';import 'package:path/path.dart' as path;import 'package:flutter/material.dart';import 'package:desktop\_drop/desktop\_drop.dart';import 'package:bot\_toast/bot\_toast.dart';import 'package:file\_picker/file\_picker.dart';import 'package:flutter/services.dart';import 'package:window\_manager/window\_manager.dart';import 'widget/custom\_check\_box.dart';void main() async {  WidgetsFlutterBinding.ensureInitialized();  await windowManager.ensureInitialized();  windowManager.waitUntilReadyToShow().then((\_) async{    await windowManager.setTitle('软著代码文档生成');    await windowManager.setSize(const Size(550, 620));    await windowManager.setPosition(const Offset(1000, 200));    windowManager.show();  });  runApp(const MyApp());}class MyApp extends StatelessWidget {  const MyApp({Key? key}) : super(key: key);  @override  Widget build(BuildContext context) {    return MaterialApp(      title: '软著代码生成',      builder: BotToastInit(),      debugShowCheckedModeBanner: false,      navigatorObservers: [BotToastNavigatorObserver()],      theme: ThemeData(primarySwatch: Colors.blue),      home: const MyHomePage(),    );  }}class MyHomePage extends StatefulWidget {  const MyHomePage({Key? key}) : super(key: key);  @override  State<MyHomePage> createState() => \_MyHomePageState();}class \_MyHomePageState extends State<MyHomePage> {  late TextEditingController headerInputController;  late FocusNode headerInputFocusNode;  String? headerInputErrorText;  Color directoryFocusColor = Colors.grey;  bool directoryDragging = false;  Directory? selectedDirectory;  late TextEditingController codeSuffixInputController;  late FocusNode codeSuffixInputFocusNode;  String? codeSuffixErrorText;  List<String> codeSuffix = ['dart'];  bool removeAnnotation = true;  bool removeEmptyLine = true;  @override  void initState() {    super.initState();    headerInputController = TextEditingController(text: '');    headerInputFocusNode = FocusNode();    codeSuffixInputController = TextEditingController(text: '');    codeSuffixInputFocusNode = FocusNode();  }  void startExtractCode () async {    if (headerInputController.text.trim().isEmpty) {      setState(() {headerInputErrorText = '请输入文档头部标题';});      FocusScope.of(context).requestFocus(headerInputFocusNode);      return;    }    if (selectedDirectory == null) {      setState(() {directoryFocusColor = Colors.red;});      BotToast.showText(text: '请选中代码文件夹');      return;    }    if (codeSuffix.isEmpty) {      setState(() {codeSuffixErrorText = '至少填写一种代码文件';});      FocusScope.of(context).requestFocus(codeSuffixInputFocusNode);      return;    }    var hideLoading = BotToast.showLoading();    List<FileSystemEntity>? list = await selectedDirectory?.list().toList();    String content = '';    int len = (list?.length ?? 0);    if (len <= 0) {      hideLoading();      BotToast.showText(text: '选中文件夹为空');      return;    }    try {      for (int i = 0; i < len; i++) {        FileSystemEntity item = list![i];        content += await readPath(item);      }    } catch (e) {      BotToast.showText(text: '文件读取失败');    }    content = content.replaceAll('<', '<').replaceAll('>', '>');    content = content.replaceAll('\n\r', '\n');    if (removeAnnotation) {      content = removeAnnotationFromCode(content);    }    List<String> strList = [];    content.split('\n').forEach((item) {      String str = item.trimRight(); // 去除右边空格      strList.add(str.replaceAll(' ', ' ')); // 空格转义      strList.add('<br/>');    });    if (removeEmptyLine) {      strList = strList.where((element) {        String text = element.trim();        if (text.isEmpty) return false;        text = text.replaceAll(' ', '');        if (text.isEmpty) return false;        text = text.replaceAll('<br/>', '');        if (text.isEmpty) return false;        return true;      }).toList();    }    content = strList.join('<br/>');    Directory tempDir = Directory.systemTemp;    ByteData data = await rootBundle.load('assets/tpl.docx');    File docFile = File(path.join(tempDir.path, 'copyright\_gen\_tpl.docx'));    await docFile.writeAsBytes(data.buffer.asUint8List());    Uint8List bytes = docFile.readAsBytesSync();    Archive zip = ZipDecoder().decodeBytes(bytes);    String zipExtName = path.join(tempDir.path, generateRandomId());    Directory zipDir = Directory(zipExtName);    await zipDir.create();    for (ArchiveFile file in zip.files) {      if (file.isFile) {        File f = File(path.join(zipExtName, file.name));        String tpl = bytesToString(file.content);        tpl = tpl.replaceAll('{{content}}', content);        tpl = tpl.replaceAll('某某某有限公司', headerInputController.text);        await f.writeAsString(tpl);      } else {        Directory d = Directory(path.join(zipExtName, file.name));        await d.create(recursive: true);      }    }    File zipFile = File(path.join(tempDir.path, generateRandomId() + '.zip'));    ZipFileEncoder newZip = ZipFileEncoder();    newZip.create(zipFile.path);    newZip.addDirectory(zipDir, includeDirName: false);    newZip.close();    Directory rootDir = Directory.current;    int now = DateTime.now().millisecondsSinceEpoch;    Uint8List docData = await zipFile.readAsBytes();    File docFile2 = File(path.join(rootDir.path, 'output\_$now.docx'));    await docFile2.writeAsBytes(docData);    await zipDir.delete(recursive: true);    await docFile.delete();    await zipFile.delete();    hideLoading();    showDialog(      context: context,      builder: (context) {        return AlertDialog(          title: const Text('提取代码成功'),          content: Text('文件已经生成在 ${docFile2.path}'),          actions: <Widget>[            ElevatedButton(              child: const Text('关闭'),              style: ElevatedButton.styleFrom(primary: Colors.grey),              onPressed: Navigator.of(context).pop,            ),            ElevatedButton(              child: const Text('打开文档'),              onPressed: () {                if (Platform.isWindows) {                  Process.run('start', [docFile2.path], runInShell: true);                }                Navigator.of(context).pop();              },            ),          ],        );      },    );  }  String removeAnnotationFromCode (String code) {    String result = code;    List<String> lines = code.split('\n');    for (int i = 0; i < lines.length; i++) {      String line = lines[i].trim(); //  去除字符串前后空白      if (line.startsWith('//')) lines[i] = ''; // 如果是// 开头就清除当前行    }    result = lines.join('\n');    return result;  }  Future<String> readPath (FileSystemEntity item) async {    FileStat stat = await item.stat();    String \_content = '';    if (stat.type == FileSystemEntityType.directory) {      Directory dir = Directory(item.path);      List<FileSystemEntity> list = await dir.list().toList();      int len = list.length;      for (int i = 0; i < len; i++) {        \_content += await readPath(list[i]);      }    } else {      File file = File(item.path);      String extname = path.extension(item.path);      extname = extname.isEmpty ? '' : extname.substring(1);      if (codeSuffix.contains(extname)) {        \_content += await file.readAsString();      }    }    return \_content;  }  OutlineInputBorder getOutlineInputBorder (Color color) {    return OutlineInputBorder(      borderRadius: const BorderRadius.all(Radius.circular(10)),      borderSide: BorderSide(color: color, width: 2.0),    );  }  @override  Widget build(BuildContext context) {    return Container(      decoration: BoxDecoration(        border: Border.all(color: Colors.grey, width: 1.0),      ),      child: Scaffold(        body: Column(          children: [            Expanded(child: Center(              child: Container(                constraints: const BoxConstraints(                  maxWidth: 500                ),                child: ListView(                  padding: const EdgeInsets.all(8),                  children: [                    const SizedBox(height: 20),                    TextField(                      controller: headerInputController,                      focusNode: headerInputFocusNode,                      decoration: InputDecoration(                        contentPadding: const EdgeInsets.symmetric(horizontal: 20),                        labelText: "文档头部标题",                        labelStyle: const TextStyle(color: Colors.grey),                        hintText: "输入文档描述",                        errorText: headerInputErrorText,                        prefixIcon: const Icon(Icons.title),                        border: getOutlineInputBorder(Colors.grey),                        enabledBorder: getOutlineInputBorder(Colors.grey),                        errorBorder: getOutlineInputBorder(Colors.red),                        focusedBorder: getOutlineInputBorder(Colors.blue),                      ),                      onChanged: (val) {                        setState(() {                          headerInputErrorText = null;                        });                      },                    ),                    Stack(                      children: [                        MouseRegion(                          cursor: SystemMouseCursors.click,                          onHover: (e) {                            setState(() {                              directoryFocusColor = Colors.blue;                            });                          },                          onExit: (e) {                            setState(() {                              directoryFocusColor = Colors.grey;                            });                          },                          child: DropTarget(                            child: GestureDetector(                              onTap: () async {                                var hideLoading = BotToast.showLoading();                                await Future.delayed(const Duration(milliseconds: 150));                                FilePicker.platform.getDirectoryPath().then((value) {                                  hideLoading();                                  if (value != null) {                                    setState(() {                                      selectedDirectory = Directory(value);                                    });                                  }                                });                              },                              child: Container(                                margin: const EdgeInsets.symmetric(vertical: 20),                                height: 200, alignment: Alignment.center,                                decoration: BoxDecoration(                                  borderRadius: const BorderRadius.all(Radius.circular(10)),                                  border: Border.all(width: 2, color: directoryFocusColor)                                ),                                child: Column(                                  mainAxisAlignment: MainAxisAlignment.center,                                  children: [                                    Icon(                                      Icons.create\_new\_folder, size: 55,                                      color: directoryFocusColor,                                    ),                                    Text('点击选择，或者拖拽进入', style: TextStyle(color: directoryFocusColor)),                                    selectedDirectory == null                                    ? const SizedBox()                                    : Text(selectedDirectory!.path, style: TextStyle(color: directoryFocusColor))                                  ],                                ),                              ),                            ),                            onDragEntered: (e) {                              setState(() {                                directoryFocusColor = Colors.blue;                                directoryDragging = true;                              });                            },                            onDragExited: (detail) {                              setState(() {                                directoryFocusColor = Colors.grey;                                directoryDragging = false;                              });                            },                            onDragDone: (detail) {                              Directory directory = Directory(detail.urls[0].toFilePath());                              if (!directory.existsSync()) {                                BotToast.showText(text: '不是一个有效的目录');                              } else {                                setState(() {                                  selectedDirectory = directory;                                });                              }                            }                          ),                        ),                        Positioned(                          top: 13,                          left: 37,                          child: Container(                            padding: const EdgeInsets.symmetric(horizontal: 4),                            color: Colors.white,                            child: Text(                              '打开代码文件夹', style: TextStyle(                                fontSize: 12,                                color: directoryFocusColor                              ),                            ),                          )                        ),                      ],                    ),                    TextField(                      controller: codeSuffixInputController,                      focusNode: codeSuffixInputFocusNode,                      decoration: InputDecoration(                        contentPadding: const EdgeInsets.symmetric(horizontal: 20),                        labelText: "代码后缀名",                        labelStyle: const TextStyle(color: Colors.grey),                        hintText: "输入后回车确认",                        errorText: codeSuffixErrorText,                        prefixIcon: const Icon(Icons.code),                        border: getOutlineInputBorder(Colors.grey),                        errorBorder: getOutlineInputBorder(Colors.red),                        enabledBorder: getOutlineInputBorder(Colors.grey),                        focusedBorder: getOutlineInputBorder(Colors.blue),                      ),                      onSubmitted: (val) {                        codeSuffixInputController.text = '';                        if (codeSuffix.contains(val.trim())) {                          BotToast.showText(text: '已存在后缀名');                        } else {                          setState(() {                            codeSuffix.add(val);                          });                        }                        FocusScope.of(context).requestFocus(codeSuffixInputFocusNode);                      },                    ),                    Container(                      padding: const EdgeInsets.all(10),                      margin: const EdgeInsets.symmetric(vertical: 5),                      decoration: BoxDecoration(                        borderRadius: const BorderRadius.all(Radius.circular(10)),                        border: Border.all(width: 2, color: Colors.grey)                      ),                      child: Wrap(                        direction: Axis.horizontal,                        children: [                          ...codeSuffix.map((suffix) {                            return Padding(                              padding: const EdgeInsets.all(5),                              child: Chip(                                label: Text(suffix),                                deleteIcon: const Icon(Icons.close, size: 16),                                deleteIconColor: Colors.black54,                                onDeleted: () {                                  setState(() {                                    codeSuffix.remove(suffix);                                  });                                },                              ),                            );                          }).toList(),                          codeSuffix.isEmpty                            ? const Text(' 请输入需要提取代码的文件后缀名', style: TextStyle(color: Colors.grey))                            : const SizedBox(),                        ],                      ),                    ),                    Row(                      children: [                        CustomCheckBox(                          value: removeAnnotation,                          label: '去除注释',                          onChanged: (val) {                            removeAnnotation = val;                          },                        ),                        CustomCheckBox(                          value: removeEmptyLine,                          label: '去除空行',                          onChanged: (val) {                            removeEmptyLine = val;                          },                        )                      ],                    ),                    const SizedBox(height: 20),                    ElevatedButton(                      child: Container(                        height: 40, alignment: Alignment.center,                        child: const Text('开始提取代码'),                      ),                      onPressed: startExtractCode,                    ),                    const SizedBox(height: 20),                  ],                )              ),            ))          ],        ),      ),    );  }}String generateRandomId() {  var random = Random();  var id = '';  for (var i = 0; i < 10; i++) {    id += random.nextInt(10).toString();  }  return 'copyright\_gen\_' + id;}String bytesToString (Uint8List bytes) {  String string = String.fromCharCodes(bytes);  return string;}import 'package:flutter/material.dart';class CustomCheckBox extends StatefulWidget {  final bool value;  final String label;  final Function(bool)? onChanged;  const CustomCheckBox({    Key? key,    required this.value,    required this.label,    this.onChanged,  }) : super(key: key);  @override  \_CustomCheckBoxState createState() => \_CustomCheckBoxState();}class \_CustomCheckBoxState extends State<CustomCheckBox> {  late bool value;  @override  void initState() {    super.initState();    value = widget.value;  }  @override  Widget build(BuildContext context) {    return MouseRegion(      cursor: SystemMouseCursors.click,      child: GestureDetector(        onTap: () {          setState(() {            value = !value;          });          widget.onChanged?.call(value);        },        child: Row(          children: [            Checkbox(              value: value,              onChanged: (val) {                setState(() {                  value = val!;                });                widget.onChanged?.call(value);              },            ),            Text(widget.label),          ],        ),      ),    );  }}