```
Import 'package:flutter/material.dart';
import 'package:sqflite/sqflite.dart';
import 'package:path_provider/path_provider.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:local_auth/local_auth.dart';
import 'package:encrypt/encrypt.dart' as encrypt;
import 'package:syncfusion_flutter_charts/charts.dart';
import 'package:flutter_secure_storage/flutter_secure_storage.dart';
import 'package:flutter_local_notifications/flutter_local_notifications.dart';
import 'package:intl/intl.dart';
import 'package:connectivity_plus/connectivity_plus.dart';
import 'package:share_plus/share_plus.dart';
import 'package:pdf/pdf.dart';
import 'package:pdf/widgets.dart' as pw;
import 'package:printing/printing.dart';
import 'dart:io';
import 'dart:convert';

// ---------------------------------------------------------------------------
// 🔹 1️⃣ Main Function - نقطة بداية التطبيق
// ---------------------------------------------------------------------------
void main() async {
WidgetsFlutterBinding.ensureInitialized();

// تهيئة إعدادات الإشعارات المحلية
final FlutterLocalNotificationsPlugin notificationsPlugin = FlutterLocalNotificationsPlugin();
const AndroidInitializationSettings androidSettings =
AndroidInitializationSettings('app_icon'); // تأكد من وجود app_icon في res/drawable
const InitializationSettings settings = InitializationSettings(android: androidSettings);
await notificationsPlugin.initialize(settings);

// طلب إذن الإشعارات لأندرويد 13 فما فوق
if (Platform.isAndroid) {
await notificationsPlugin
.resolvePlatformSpecificImplementation<
AndroidFlutterLocalNotificationsPlugin>()
?.requestNotificationsPermission();
}

runApp(const FinancialManagerApp());
}

// ---------------------------------------------------------------------------
// 🔹 2️⃣ FinancialManagerApp - التطبيق الرئيسي
// ---------------------------------------------------------------------------
```

```
// ---------------------------------------------------------------------------
class FinancialManagerApp extends StatelessWidget {
const FinancialManagerApp({super.key});

@override
Widget build(BuildContext context) {
return FutureBuilder<bool>(
future: _getThemePreference(),
builder: (context, snapshot) {
final isDarkMode = snapshot.data ?? false;
return MaterialApp(
title: 'إدارة المالية الشخصية',
theme: ThemeData(
primarySwatch: Colors.green,
brightness: Brightness.light,
useMaterial3: true,
),
darkTheme: ThemeData(
primarySwatch: Colors.green,
brightness: Brightness.dark,
useMaterial3: true,
),
themeMode: isDarkMode ? ThemeMode.dark : ThemeMode.light,
home: const SecurityScreen(),
);
},
);
}

static Future<bool> _getThemePreference() async {
final prefs = await SharedPreferences.getInstance();
return prefs.getBool('isDarkMode') ?? false;
}
}

// ---------------------------------------------------------------------------
// 🔹 3️⃣ Models - نماذج البيانات
// ---------------------------------------------------------------------------
class SpendingHabit {
final int id;
final String amount; // مشفر
final String category;
final int frequency;
final String date;
```

```
SpendingHabit({
required this.id,
required this.amount,
required this.category,
required this.frequency,
required this.date,
});

factory SpendingHabit.fromMap(
Map<String, dynamic> map, encrypt.Encrypter encrypter, encrypt.IV iv) {
return SpendingHabit(
id: map['id'],
amount: encrypter.decrypt64(map['amount'], iv: iv),
category: map['category'],
frequency: map['frequency'],
date: map['date'],
);
}

Map<String, dynamic> toMap(encrypt.Encrypter encrypter, encrypt.IV iv) {
return {
'id': id,
'amount': encrypter.encrypt(amount, iv: iv).base64,
'category': category,
'frequency': frequency,
'date': date,
};
}
}

class Loan {
final int id;
final String friend;
final String amount; // مشفر
final bool isPaid;

Loan({
required this.id,
required this.friend,
required this.amount,
required this.isPaid,
});
```

```
factory Loan.fromMap(
Map<String, dynamic> map, encrypt.Encrypter encrypter, encrypt.IV iv) {
return Loan(
id: map['id'],
friend: map['friend'],
amount: encrypter.decrypt64(map['amount'], iv: iv),
isPaid: map['isPaid'] == 1,
);
}

Map<String, dynamic> toMap(encrypt.Encrypter encrypter, encrypt.IV iv) {
return {
'id': id,
'friend': friend,
'amount': encrypter.encrypt(amount, iv: iv).base64,
'isPaid': isPaid ? 1 : 0,
};
}
}

class Budget {
final int id;
final String category;
final double maxAmount;
final String period; // 'monthly' or 'weekly'

Budget({
required this.id,
required this.category,
required this.maxAmount,
required this.period,
});

factory Budget.fromMap(Map<String, dynamic> map) {
return Budget(
id: map['id'],
category: map['category'],
maxAmount: map['maxAmount'],
period: map['period'],
);
}

Map<String, dynamic> toMap() {
return {
```

```
'id': id,
'category': category,
'maxAmount': maxAmount,
'period': period,
};
}
}

class Income {
final int id;
final String source;
final String amount; // مشفر
final String date;

Income({
required this.id,
required this.source,
required this.amount,
required this.date,
});

factory Income.fromMap(
Map<String, dynamic> map, encrypt.Encrypter encrypter, encrypt.IV iv) {
return Income(
id: map['id'],
source: map['source'],
amount: encrypter.decrypt64(map['amount'], iv: iv),
date: map['date'],
);
}

Map<String, dynamic> toMap(encrypt.Encrypter encrypter, encrypt.IV iv) {
return {
'id': id,
'source': source,
'amount': encrypter.encrypt(amount, iv: iv).base64,
'date': date,
};
}
}

class SavingsGoal {
final int id;
final String goalName;
```

```
final String targetAmount; // مشفر
final String currentAmount; // مشفر

SavingsGoal({
required this.id,
required this.goalName,
required this.targetAmount,
required this.currentAmount,
});

factory SavingsGoal.fromMap(
Map<String, dynamic> map, encrypt.Encrypter encrypter, encrypt.IV iv) {
return SavingsGoal(
id: map['id'],
goalName: map['goalName'],
targetAmount: encrypter.decrypt64(map['targetAmount'], iv: iv),
currentAmount: encrypter.decrypt64(map['currentAmount'], iv: iv),
);
}

Map<String, dynamic> toMap(encrypt.Encrypter encrypter, encrypt.IV iv) {
return {
'id': id,
'goalName': goalName,
'targetAmount': encrypter.encrypt(targetAmount, iv: iv).base64,
'currentAmount': encrypter.encrypt(currentAmount, iv: iv).base64,
};
}
}

class Bill {
final int id;
final String name;
final String amount; // مشفر
final String dueDate;
final bool isPaid;

Bill({
required this.id,
required this.name,
required this.amount,
required this.dueDate,
required this.isPaid,
});
```

```
factory Bill.fromMap(
Map<String, dynamic> map, encrypt.Encrypter encrypter, encrypt.IV iv) {
return Bill(
id: map['id'],
name: map['name'],
amount: encrypter.decrypt64(map['amount'], iv: iv),
dueDate: map['dueDate'],
isPaid: map['isPaid'] == 1,
);
}

Map<String, dynamic> toMap(encrypt.Encrypter encrypter, encrypt.IV iv) {
return {
'id': id,
'name': name,
'amount': encrypter.encrypt(amount, iv: iv).base64,
'dueDate': dueDate,
'isPaid': isPaid ? 1 : 0,
};
}
}

class Category {
final int id;
final String name;
final String icon; // يمكن أن يكون اسم أيقونة أو مسار صورة

Category({
required this.id,
required this.name,
required this.icon,
});

factory Category.fromMap(Map<String, dynamic> map) {
return Category(
id: map['id'],
name: map['name'],
icon: map['icon'],
);
}

Map<String, dynamic> toMap() {
return {
```

```
'id': id,
'name': name,
'icon': icon,
};
}
}

// ---------------------------------------------------------------------------
// 🔹 4️⃣ SecurityScreen - شاشة المصادقة البيومترية
// ---------------------------------------------------------------------------
class SecurityScreen extends StatefulWidget {
const SecurityScreen({super.key});

@override
_SecurityScreenState createState() => _SecurityScreenState();
}

class _SecurityScreenState extends State<SecurityScreen> {
final LocalAuthentication auth = LocalAuthentication();
bool _isLoading = false;

@override
void initState() {
super.initState();
// يمكنك هنا استدعاء _authenticate() عند البدء تلقائياً إذا أردت فرض المصادقة
// _authenticate();
}

Future<void> _authenticate() async {
setState(() => _isLoading = true);
try {
bool canCheckBiometrics = await auth.canCheckBiometrics;
if (!canCheckBiometrics) {
ScaffoldMessenger.of(context).showSnackBar(
const SnackBar(content: Text("البصمة غير مدعومة على هذا الجهاز")),
);
Navigator.pushReplacement(
context,
MaterialPageRoute(builder: (context) => const HomeScreen()),
);
return;
}

bool authenticated = await auth.authenticate(
```

```
localizedReason: 'يرجى تأكيد هويتك للدخول',
options: const AuthenticationOptions(
biometricOnly: true,
stickyAuth: true,
),
);

if (authenticated) {
Navigator.pushReplacement(
context,
MaterialPageRoute(builder: (context) => const HomeScreen()),
);
} else {
ScaffoldMessenger.of(context).showSnackBar(
const SnackBar(content: Text("فشل المصادقة، حاول مرة أخرى")),
);
}
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text("خطأ أثناء المصادقة: $e")),
);
} finally {
setState(() => _isLoading = false);
}
}

@override
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(title: const Text("تسجيل الدخول الآمن")),
body: Center(
child: _isLoading
? const CircularProgressIndicator()
: Column(
mainAxisAlignment: MainAxisAlignment.center,
children: [
ElevatedButton(
onPressed: _authenticate,
child: const Text("تسجيل الدخول بالبصمة"),
),
const SizedBox(height: 16),
TextButton(
onPressed: () {
Navigator.pushReplacement(
```

```
context,
MaterialPageRoute(builder: (context) => const HomeScreen()),
);
},
child: const Text("تخطي المصادقة"),
),
],
),
),
);
}
}

// ---------------------------------------------------------------------------
// 🔹 5️⃣ HomeScreen - الشاشة الرئيسية لإدارة المالية
// ---------------------------------------------------------------------------
class HomeScreen extends StatefulWidget {
const HomeScreen({super.key});

@override
_HomeScreenState createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
late Database _database;
late encrypt.Encrypter encrypter;
late encrypt.Key key;
late encrypt.IV iv;
List<SpendingHabit> _spendingData = [];
List<Loan> _loansData = [];
List<Budget> _budgetsData = [];
List<Income> _incomesData = [];
List<SavingsGoal> _savingsGoalsData = [];
List<Bill> _billsData = [];
List<Category> _categoriesData = [];
bool _isDarkMode = false;
bool _isStealthMode = false;
bool _isLoading = false;
String _language = 'ar'; // الافتراضي هو العربية
final FlutterLocalNotificationsPlugin notificationsPlugin = FlutterLocalNotificationsPlugin();
final _secureStorage = const FlutterSecureStorage();

static const String _backupKey = 'backup_data'; // مفتاح للنسخ الاحتياطي المحلي
```

```
@override
void initState() {
super.initState();
_loadPreferences();
_initializeEncryptionAndDatabase();
}

@override
void dispose() {
حاول إغلاق قاعدة البيانات فقط إذا كانت مفتوحة //
إذا كانت لا تزال في حالة تهيئة database_ تجنب الوصول إلى //
try {
if (_database.isOpen) {
_database.close();
}
} catch (e) {
debugPrint("Error closing database: $e");
}
super.dispose();
}

Future<void> _loadPreferences() async {
final prefs = await SharedPreferences.getInstance();
setState(() {
_isDarkMode = prefs.getBool('isDarkMode') ?? false;
_isStealthMode = prefs.getBool('isStealthMode') ?? false;
_language = prefs.getString('language') ?? 'ar';
});
}

Future<void> _setThemePreference(bool value) async {
final prefs = await SharedPreferences.getInstance();
await prefs.setBool('isDarkMode', value);
setState(() {
_isDarkMode = value;
إعادة تشغيل التطبيق لتطبيق الثيم الجديد فوراً //
runApp(const FinancialManagerApp());
});
}

Future<void> _setStealthMode(bool value) async {
final prefs = await SharedPreferences.getInstance();
await prefs.setBool('isStealthMode', value);
setState(() {
```

```
_isStealthMode = value;
});
}

Future<void> _setLanguage(String value) async {
final prefs = await SharedPreferences.getInstance();
await prefs.setString('language', value);
setState(() {
_language = value;
});
قد تحتاج إلى إعادة تشغيل التطبيق أو تحديث الواجهة بشكل أعمق //
.لكي يتم تطبيق تغيير اللغة على جميع النصوص فوراً //
}

void _showLanguageDialog() {
showDialog(
context: context,
builder: (context) {
return AlertDialog(
title: Text(_language == 'ar' ? 'اختر اللغة' : 'Select Language'),
content: Column(
mainAxisSize: MainAxisSize.min,
children: [
RadioListTile<String>(
title: const Text('العربية'),
value: 'ar',
groupValue: _language,
onChanged: (value) {
if (value != null) {
_setLanguage(value);
Navigator.pop(context);
}
},
),
RadioListTile<String>(
title: const Text('English'),
value: 'en',
groupValue: _language,
onChanged: (value) {
if (value != null) {
_setLanguage(value);
Navigator.pop(context);
}
},
```

```
),
],
),
);
},
);
}

Future<void> _initializeEncryptionAndDatabase() async {
setState(() => _isLoading = true);
try {
// تهيئة التشفير
String? storedKey = await _secureStorage.read(key: 'encryption_key');
String? storedIV = await _secureStorage.read(key: 'encryption_iv');
if (storedKey == null || storedIV == null) {
key = encrypt.Key.fromSecureRandom(32);
iv = encrypt.IV.fromSecureRandom(16);
await _secureStorage.write(key: 'encryption_key', value: key.base64);
await _secureStorage.write(key: 'encryption_iv', value: iv.base64);
} else {
key = encrypt.Key.fromBase64(storedKey);
iv = encrypt.IV.fromBase64(storedIV);
}
encrypter = encrypt.Encrypter(encrypt.AES(key));

// تهيئة قاعدة البيانات
Directory directory = await getApplicationDocumentsDirectory();
String path = '${directory.path}/financial_manager.db';

_database = await openDatabase(
path,
version: 1, // رقم الإصدار الأولي
onCreate: (db, version) async {
await db.execute(
"CREATE TABLE spending(id INTEGER PRIMARY KEY AUTOINCREMENT, amount TEXT, category TEXT, frequency INTEGER, date TEXT)",
);
await db.execute(
"CREATE TABLE loans(id INTEGER PRIMARY KEY AUTOINCREMENT, friend TEXT, amount TEXT, isPaid INTEGER)",
);
await db.execute(
"CREATE TABLE budgets(id INTEGER PRIMARY KEY AUTOINCREMENT, category TEXT, maxAmount REAL, period TEXT)",
```

```
        );
        await db.execute(
          "CREATE TABLE incomes(id INTEGER PRIMARY KEY AUTOINCREMENT, source TEXT, amount TEXT, date TEXT)",
        );
        await db.execute(
          "CREATE TABLE savings_goals(id INTEGER PRIMARY KEY AUTOINCREMENT, goalName TEXT, targetAmount TEXT, currentAmount TEXT)",
        );
        await db.execute(
          "CREATE TABLE bills(id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT, amount TEXT, dueDate TEXT, isPaid INTEGER)",
        );
        await db.execute(
          "CREATE TABLE categories(id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT, icon TEXT)",
        );
      },
      // onUpgrade: إذا كان لديك إصدارات سابقة من قاعدة البيانات وتحتاج لترحيلها، يمكنك استخدام
      // onUpgrade: (db, oldVersion, newVersion) async {
      //   if (oldVersion < 2) {
      //     await db.execute("CREATE TABLE budgets(id INTEGER PRIMARY KEY AUTOINCREMENT, category TEXT, maxAmount REAL, period TEXT)");
      //     // أضف جداول أخرى هنا ...
      //   }
      // },
    );

    // جلب جميع البيانات بعد التهيئة
    await _fetchSpendingData();
    await _fetchLoansData();
    await _fetchBudgetsData();
    await _fetchIncomesData();
    await _fetchSavingsGoalsData();
    await _fetchBillsData();
    await _fetchCategoriesData();

    // فحص التنبيهات بعد جلب البيانات
    await _checkBudgetAlerts();
    await _checkBillReminders();
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء تهيئة التطبيق: $e' : 'Error initializing app: $e')),
```

```
    );
  } finally {
    setState(() => _isLoading = false);
  }
 }

 String _formatDate(String isoDate) {
  try {
   return DateFormat('dd/MM/yyyy').format(DateTime.parse(isoDate));
  } catch (e) {
   return isoDate; // في حالة وجود خطأ في تنسيق التاريخ
  }
 }

 // ---------------------------------------------------------------------------
 // 🔹 6️⃣ Data Management - إضافة/حذف/جلب البيانات
 // ---------------------------------------------------------------------------

 Future<void> _fetchSpendingData() async {
  try {
   final List<Map<String, dynamic>> data =
     await _database.query("spending", orderBy: "date DESC");
   setState(() {
    _spendingData =
      data.map((item) => SpendingHabit.fromMap(item, encrypter, iv)).toList();
   });
  } catch (e) {
   ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء جلب بيانات الإنفاق: $e' : 'Error fetching
      spending data: $e')),
   );
  }
 }

 Future<void> _addSpending(String amount, String category, int frequency, String date) async {
  try {
   await _database.insert(
    'spending',
    {'amount': amount, 'category': category, 'frequency': frequency, 'date': date},
   );
   await _fetchSpendingData();
   await _checkBudgetAlerts();
  } catch (e) {
   ScaffoldMessenger.of(context).showSnackBar(
```

```
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء إضافة الإنفاق: $e' : 'Error adding spending: $e')),
);
}
}

Future<void> _deleteSpending(int id) async {
try {
await _database.delete('spending', where: 'id = ?', whereArgs: [id]);
await _fetchSpendingData();
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء حذف الإنفاق: $e' : 'Error deleting spending: $e')),
);
}
}

Future<void> _fetchLoansData() async {
try {
final List<Map<String, dynamic>> data = await _database.query("loans");
setState(() {
_loansData = data.map((item) => Loan.fromMap(item, encrypter, iv)).toList();
});
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء جلب بيانات القروض: $e' : 'Error fetching loans data: $e')),
);
}
}

Future<void> _addLoan(String friend, String amount, bool isPaid) async {
try {
await _database.insert(
'loans',
{'friend': friend, 'amount': amount, 'isPaid': isPaid ? 1 : 0},
);
await _fetchLoansData();
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء إضافة القرض: $e' : 'Error adding loan: $e')),
);
```

```
  }
  }

  Future<void> _deleteLoan(int id) async {
    try {
      await _database.delete('loans', where: 'id = ?', whereArgs: [id]);
      await _fetchLoansData();
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء حذف القرض: $e' : 'Error deleting loan: $e')),
      );
    }
  }

  Future<void> _updateLoanStatus(int id, bool isPaid) async {
    try {
      await _database.update(
        'loans',
        {'isPaid': isPaid ? 1 : 0},
        where: 'id = ?',
        whereArgs: [id],
      );
      await _fetchLoansData();
      if (!isPaid) {
        // يمكنك إرسال تذكير هنا إذا لم يتم الدفع
        final loan = _loansData.firstWhere((l) => l.id == id);
        await _sendDebtReminder(loan.friend);
      }
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء تحديث حالة القرض: $e' : 'Error updating loan status: $e')),
      );
    }
  }

  Future<void> _fetchBudgetsData() async {
    try {
      final List<Map<String, dynamic>> data = await _database.query("budgets");
      setState(() {
        _budgetsData = data.map((item) => Budget.fromMap(item)).toList();
      });
    } catch (e) {
```

```
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء جلب بيانات الميزانية: $e' : 'Error fetching budgets data: $e')),
);
}
}

Future<void> _addBudget(String category, double maxAmount, String period) async {
try {
await _database.insert(
'budgets',
{
'category': category,
'maxAmount': maxAmount,
'period': period,
},
);
await _fetchBudgetsData();
await _checkBudgetAlerts();
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء إضافة الميزانية: $e' : 'Error adding budget: $e')),
);
}
}

Future<void> _deleteBudget(int id) async {
try {
await _database.delete('budgets', where: 'id = ?', whereArgs: [id]);
await _fetchBudgetsData();
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء حذف الميزانية: $e' : 'Error deleting budget: $e')),
);
}
}

Future<void> _fetchIncomesData() async {
try {
final List<Map<String, dynamic>> data = await _database.query("incomes");
setState(() {
_incomesData = data.map((item) => Income.fromMap(item, encrypter, iv)).toList();
```

```
      });
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء جلب بيانات الدخل: $e' : 'Error fetching incomes data: $e')),
      );
    }
  }

  Future<void> _addIncome(String source, double amount, String date) async {
    try {
      final encryptedAmount = encrypter.encrypt(amount.toString(), iv: iv).base64;
      await _database.insert(
        'incomes',
        {
          'source': source,
          'amount': encryptedAmount,
          'date': date,
        },
      );
      await _fetchIncomesData();
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء إضافة الدخل: $e' : 'Error adding income: $e')),
      );
    }
  }

  Future<void> _deleteIncome(int id) async {
    try {
      await _database.delete('incomes', where: 'id = ?', whereArgs: [id]);
      await _fetchIncomesData();
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء حذف الدخل: $e' : 'Error deleting income: $e')),
      );
    }
  }

  Future<void> _fetchSavingsGoalsData() async {
    try {
      final List<Map<String, dynamic>> data = await _database.query("savings_goals");
```

```
setState(() {
_savingsGoalsData = data.map((item) => SavingsGoal.fromMap(item, encrypter,
iv)).toList();
});
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء جلب بيانات أهداف الادخار: $e' : 'Error fetching
savings goals data: $e')),
);
}
}

Future<void> _addSavingsGoal(
String goalName, double targetAmount, double currentAmount) async {
try {
final encryptedTarget = encrypter.encrypt(targetAmount.toString(), iv: iv).base64;
final encryptedCurrent = encrypter.encrypt(currentAmount.toString(), iv: iv).base64;
await _database.insert(
'savings_goals',
{
'goalName': goalName,
'targetAmount': encryptedTarget,
'currentAmount': encryptedCurrent,
},
);
await _fetchSavingsGoalsData();
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء إضافة هدف الادخار: $e' : 'Error adding
savings goal: $e')),
);
}
}

Future<void> _deleteSavingsGoal(int id) async {
try {
await _database.delete('savings_goals', where: 'id = ?', whereArgs: [id]);
await _fetchSavingsGoalsData();
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء حذف هدف الادخار: $e' : 'Error deleting
savings goal: $e')),
);
}
```

```
}

Future<void> _fetchBillsData() async {
try {
final List<Map<String, dynamic>> data = await _database.query("bills");
setState(() {
_billsData = data.map((item) => Bill.fromMap(item, encrypter, iv)).toList();
});
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء جلب بيانات الفواتير: $e' : 'Error fetching bills
data: $e')),
);
}
}

Future<void> _addBill(String name, double amount, String dueDate, bool isPaid) async {
try {
final encryptedAmount = encrypter.encrypt(amount.toString(), iv: iv).base64;
await _database.insert(
'bills',
{
'name': name,
'amount': encryptedAmount,
'dueDate': dueDate,
'isPaid': isPaid ? 1 : 0,
},
);
await _fetchBillsData();
if (!isPaid) {
await _scheduleBillReminder(name, dueDate);
}
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء إضافة الفاتورة: $e' : 'Error adding bill: $e')),
);
}
}

Future<void> _deleteBill(int id) async {
try {
await _database.delete('bills', where: 'id = ?', whereArgs: [id]);
await _fetchBillsData();
} catch (e) {
```

```
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء حذف الفاتورة: $e' : 'Error deleting bill: $e')),
);
}
}

Future<void> _updateBillStatus(int id, bool isPaid) async {
try {
await _database.update(
'bills',
{'isPaid': isPaid ? 1 : 0},
where: 'id = ?',
whereArgs: [id],
);
await _fetchBillsData();
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء تحديث حالة الفاتورة: $e' : 'Error updating bill
status: $e')),
);
}
}

Future<void> _fetchCategoriesData() async {
try {
final List<Map<String, dynamic>> data = await _database.query("categories");
setState(() {
_categoriesData = data.map((item) => Category.fromMap(item)).toList();
if (_categoriesData.isEmpty) {
// إذا لم تكن هناك فئات، أضف الافتراضية
_addDefaultCategories();
}
});
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء جلب بيانات الفئات: $e' : 'Error fetching
categories data: $e')),
);
}
}

Future<void> _addCategory(String name, String icon) async {
try {
await _database.insert(
```

```
        'categories',
        {'name': name, 'icon': icon},
      );
      await _fetchCategoriesData();
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء إضافة الفئة: $e' : 'Error adding category: $e')),
      );
    }
  }

  Future<void> _deleteCategory(int id) async {
    try {
      await _database.delete('categories', where: 'id = ?', whereArgs: [id]);
      await _fetchCategoriesData();
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء حذف الفئة: $e' : 'Error deleting category: $e')),
      );
    }
  }

  Future<void> _addDefaultCategories() async {
    await _addCategory('طعام', 'food');
    await _addCategory('مواصلات', 'transport');
    await _addCategory('ترفيه', 'entertainment');
    await _addCategory('تسوق', 'shopping');
    await _addCategory('فواتير', 'bills');
    await _addCategory('صحة', 'health');
    await _addCategory('تعليم', 'education');
    await _addCategory('منزل', 'home');
    await _addCategory('أخرى', 'other');
    // بعد إضافة الافتراضيات، أعد جلب البيانات لتحديث الـ UI
    await _fetchCategoriesData();
  }

  // ---------------------------------------------------------------------------
  // 🔹 7️⃣ Advanced Features - ميزات متقدمة 🔹
  // ---------------------------------------------------------------------------

  Future<double> _getSpendingForCategory(String category, String period) async {
    try {
```

```
DateTime startDate;
DateTime endDate = DateTime.now();

if (period == 'monthly') {
startDate = DateTime(endDate.year, endDate.month, 1);
} else {
// أسبوعي (من بداية الأسبوع الحالي)
startDate = endDate.subtract(Duration(days: endDate.weekday - 1));
}

final spendingData = await _database.query('spending',
where: 'category = ? AND date BETWEEN ? AND ?',
whereArgs: [
category,
startDate.toIso8601String(),
endDate.toIso8601String(),
]);
double total = 0.0;
for (var item in spendingData) {
total += double.parse(encrypter.decrypt64(item['amount'] as String, iv: iv));
}
return total;
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء حساب الإنفاق: $e' : 'Error calculating spending: $e')),
);
return 0.0;
}
}

Future<void> _checkBudgetAlerts() async {
final budgets = _budgetsData;
for (var budget in budgets) {
final category = budget.category;
final maxAmount = budget.maxAmount;
final period = budget.period;
final spent = await _getSpendingForCategory(category, period);

if (spent >= maxAmount * 0.9) {
const androidDetails = AndroidNotificationDetails(
'budget_alert',
'تنبيه الميزانية',
channelDescription: 'تنبيهات عند اقتراب تجاوز الميزانية',
```

```
                importance: Importance.high,
              );
              const details = NotificationDetails(android: androidDetails);
              await notificationsPlugin.show(
                category.hashCode,
                _language == 'ar' ? 'تحذير الميزانية' : 'Budget Warning',
                _language == 'ar'
                    ? 'وصل إلى $category إنفاقك على ${((spent / maxAmount) * 100).toStringAsFixed(1)}%'
                    : 'Your spending on $category reached ${((spent / maxAmount) *
                        100).toStringAsFixed(1)}%',
                details,
              );
            }
          }
        }

        Future<void> _scheduleBillReminder(String name, String dueDate) async {
          try {
            final due = DateTime.parse(dueDate);
            final now = DateTime.now();
            if (due.isAfter(now)) {
              // جدولة التذكير قبل 3 أيام من تاريخ الاستحقاق
              final scheduledDate = due.subtract(const Duration(days: 3));
              if (scheduledDate.isAfter(now)) {
                const androidDetails = AndroidNotificationDetails(
                  'bill_reminder_channel',
                  'تذكير بالفواتير',
                  channelDescription: 'تنبيهات لتذكير الفواتير المستحقة',
                  importance: Importance.high,
                );
                const details = NotificationDetails(android: androidDetails);
                await notificationsPlugin.schedule(
                  name.hashCode + 1, // معرف فريد
                  _language == 'ar' ? 'تذكير بالفاتورة' : 'Bill Reminder',
                  _language == 'ar'
                      ? 'فاتورة "$name" تستحق في ${DateFormat('dd/MM/yyyy').format(due)}'
                      : 'Bill "$name" is due on ${DateFormat('dd/MM/yyyy').format(due)}',
                  scheduledDate,
                  details,
                  androidScheduleMode: AndroidScheduleMode.exactAllowWhileIdle,
                );
              }
            }
          } catch (e) {
```

```
debugPrint("Error scheduling bill reminder: $e");
}
}

Future<void> _checkBillReminders() async {
for (var bill in _billsData) {
if (!bill.isPaid) {
await _scheduleBillReminder(bill.name, bill.dueDate);
}
}
}

Future<void> _sendDebtReminder(String friend) async {
const AndroidNotificationDetails androidDetails = AndroidNotificationDetails(
'debt_reminder_channel',
'تذكير بالديون',
channelDescription: 'تنبيهات لتذكيرك بالديون',
importance: Importance.high,
);
const NotificationDetails details = NotificationDetails(android: androidDetails);
await notificationsPlugin.show(
معرف فريد // ,2 + friend.hashCode
_language == 'ar' ? 'تذكير بالدين' : 'Debt Reminder',
_language == 'ar'
? 'يجب سداد الدين المستحق لـ $friend'
: 'You need to settle the debt owed to $friend',
details,
);
}

Future<void> _transferBetweenAccounts(
String fromAccount, String toAccount, double amount) async {
try {
final date = DateTime.now().toIso8601String();
String fromCategory = '';
String toCategory = '';

if (fromAccount == 'incomes') {
fromCategory = _language == 'ar' ? 'سحب من الدخل' : 'Withdraw from Income';
} else if (fromAccount == 'savings_goals') {
fromCategory = _language == 'ar' ? 'سحب من المدخرات' : 'Withdraw from Savings';
} else {
fromCategory = _language == 'ar' ? 'نقل من مصروفات عامة' : 'Transfer from General Spending';
}
```

```
if (toAccount == 'incomes') {
toCategory = _language == 'ar' ? 'إضافة للدخل' : 'Add to Income';
} else if (toAccount == 'savings_goals') {
toCategory = _language == 'ar' ? 'إضافة للمدخرات' : 'Add to Savings';
} else {
toCategory = _language == 'ar' ? 'إضافة لمصروفات عامة' : 'Add to General Spending';
}

// تسجيل عملية سحب (مبلغ سالب)
await _database.insert('spending', {
'amount': encrypter.encrypt((-amount).toString(), iv: iv).base64,
'category': fromCategory,
'frequency': 1,
'date': date,
});

// تسجيل عملية إضافة (مبلغ موجب)
await _database.insert('spending', {
'amount': encrypter.encrypt(amount.toString(), iv: iv).base64,
'category': toCategory,
'frequency': 1,
'date': date,
});

await _fetchSpendingData();
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'تم التحويل بنجاح!' : 'Transfer successful!')),
);
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء التحويل: $e' : 'Error transferring amount: $e')),
);
}
}

Future<List<Map<String, dynamic>>> _searchTransactions(
String query, String table,
{DateTime? startDate, DateTime? endDate, double? minAmount, double? maxAmount})
async {
try {
final List<Map<String, dynamic>> rawResults = await _database.query(table);
final List<Map<String, dynamic>> decryptedResults = rawResults.map((item) {
```

```
قبل فك التشفير String وأنه من نوع 'amount' تأكد من وجود مفتاح //
if (item.containsKey('amount') && item['amount'] is String) {
return {
...item,
'decryptedAmount': double.tryParse(encrypter.decrypt64(item['amount'] as String, iv: iv))
?? 0.0,
};
}
return item; // أعد العنصر كما هو إذا لم يكن يحتوي على مفتاح 'amount'
}).toList();

final filteredResults = decryptedResults.where((item) {
bool matchesQuery = true;
if (query.isNotEmpty) {
if (table == 'spending' && item.containsKey('category')) {
matchesQuery = (item['category'] as
String).toLowerCase().contains(query.toLowerCase());
} else if (table == 'loans' && item.containsKey('friend')) {
matchesQuery = (item['friend'] as String).toLowerCase().contains(query.toLowerCase());
} else if (table == 'incomes' && item.containsKey('source')) {
matchesQuery = (item['source'] as
String).toLowerCase().contains(query.toLowerCase());
} else if (table == 'bills' && item.containsKey('name')) {
matchesQuery = (item['name'] as String).toLowerCase().contains(query.toLowerCase());
} else if (table == 'savings_goals' && item.containsKey('goalName')) {
matchesQuery = (item['goalName'] as
String).toLowerCase().contains(query.toLowerCase());
}
}

bool matchesDate = true;
if (item.containsKey('date') && item['date'] is String) {
try {
final itemDate = DateTime.parse(item['date'] as String);
if (startDate != null && itemDate.isBefore(startDate.subtract(const Duration(days: 1)))) {
matchesDate = false;
}
if (endDate != null && itemDate.isAfter(endDate.add(const Duration(days: 1)))) {
matchesDate = false;
}
} catch (e) {
matchesDate = false; // إذا كان التاريخ غير صالح، لا يتطابق
}
}
```

```
                bool matchesAmount = true;
                if (item.containsKey('decryptedAmount')) {
                  final amount = item['decryptedAmount'] as double;
                  if (minAmount != null && amount < minAmount) matchesAmount = false;
                  if (maxAmount != null && amount > maxAmount) matchesAmount = false;
                }

                return matchesQuery && matchesDate && matchesAmount;
              }).toList();

              // إعادة العناصر بالشكل الأصلي مع المبلغ المشفّر إذا لم تكن هناك حاجة لـ decryptedAmount
              return filteredResults.map((item) {
                final Map<String, dynamic> cleanItem = Map.from(item);
                if (cleanItem.containsKey('decryptedAmount')) {
                  cleanItem.remove('decryptedAmount'); // إزالة المفتاح الإضافي
                }
                return cleanItem;
              }).toList();

            } catch (e) {
              ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء البحث عن المعاملات: $e' : 'Error searching transactions: $e')),
              );
              return [];
            }
          }

          Future<void> _exportReportAsPDF() async {
            try {
              final pdf = pw.Document();

              // جلب البيانات الخام من قاعدة البيانات
              final spendingData = await _database.query('spending');
              final incomesData = await _database.query('incomes');
              final loansData = await _database.query('loans');
              final budgetsData = await _database.query('budgets');
              final savingsGoalsData = await _database.query('savings_goals');
              final billsData = await _database.query('bills');

              // دالة مساعدة لإنشاء صف في الجدول
              pw.Widget _buildRow(String label, String value) {
                return pw.Row(
```

```
children: [
pw.Expanded(flex: 2, child: pw.Text(label, style: const pw.TextStyle(fontSize: 10))),
pw.Expanded(flex: 3, child: pw.Text(value, style: const pw.TextStyle(fontSize: 10))),
],
);
}

pdf.addPage(
pw.MultiPage(
pageFormat: PdfPageFormat.a4,
build: (pw.Context context) {
return [
pw.Center(
child: pw.Text(
_language == 'ar' ? 'تقرير المالية الشخصية' : 'Personal Finance Report',
style: pw.TextStyle(fontSize: 24, fontWeight: pw.FontWeight.bold),
),
),
pw.SizedBox(height: 20),
// الإنفاق
pw.Text(_language == 'ar' ? 'الإنفاق:' : 'Spending:', style: pw.TextStyle(fontSize: 16,
fontWeight: pw.FontWeight.bold)),
pw.Divider(),
...spendingData.map((item) {
return _buildRow(
item['category'] as String,
'${encrypter.decrypt64(item['amount'] as String, iv: iv)} ${_language == 'ar' ? 'جنيه' :
'EGP'} (${_formatDate(item['date'] as String)})',
);
}),
pw.SizedBox(height: 10),

// الدخل
pw.Text(_language == 'ar' ? 'الدخل:' : 'Income:', style: pw.TextStyle(fontSize: 16,
fontWeight: pw.FontWeight.bold)),
pw.Divider(),
...incomesData.map((item) {
return _buildRow(
item['source'] as String,
'${encrypter.decrypt64(item['amount'] as String, iv: iv)} ${_language == 'ar' ? 'جنيه' :
'EGP'} (${_formatDate(item['date'] as String)})',
);
}),
pw.SizedBox(height: 10),
```

```
القروض //
pw.Text(_language == 'ar' ? 'القروض:' : 'Loans:', style: pw.TextStyle(fontSize: 16,
fontWeight: pw.FontWeight.bold)),
pw.Divider(),
...loansData.map((item) {
return _buildRow(
item['friend'] as String,
'${encrypter.decrypt64(item['amount'] as String, iv: iv)} ${_language == 'ar' ? 'جنيه' :
'EGP'} (${item['isPaid'] == 1 ? (_language == 'ar' ? 'تم السداد' : 'Paid') : (_language == 'ar' ? 'لم يُسدد' :
'Unpaid')})',
);
}),
pw.SizedBox(height: 10),

الميزانيات //
pw.Text(_language == 'ar' ? 'الميزانيات:' : 'Budgets:', style: pw.TextStyle(fontSize: 16,
fontWeight: pw.FontWeight.bold)),
pw.Divider(),
...budgetsData.map((item) {
return _buildRow(
item['category'] as String,
'${item['maxAmount']} ${_language == 'ar' ? 'جنيه' : 'EGP'} (${item['period'] ==
'monthly' ? (_language == 'ar' ? 'شهري' : 'Monthly') : (_language == 'ar' ? 'أسبوعي' : 'Weekly')})',
);
}),
pw.SizedBox(height: 10),

أهداف الادخار //
pw.Text(_language == 'ar' ? 'أهداف الادخار:' : 'Savings Goals:', style: pw.TextStyle(fontSize:
16, fontWeight: pw.FontWeight.bold)),
pw.Divider(),
...savingsGoalsData.map((item) {
return _buildRow(
item['goalName'] as String,
'${encrypter.decrypt64(item['currentAmount'] as String, iv:
iv)}/${encrypter.decrypt64(item['targetAmount'] as String, iv: iv)} ${_language == 'ar' ? 'جنيه' :
'EGP'}',
);
}),
pw.SizedBox(height: 10),

الفواتير //
```

```
pw.Text(_language == 'ar' ? 'الفواتير:' : 'Bills:', style: pw.TextStyle(fontSize: 16, fontWeight:
pw.FontWeight.bold)),
pw.Divider(),
...billsData.map((item) {
return _buildRow(
item['name'] as String,
'${encrypter.decrypt64(item['amount'] as String, iv: iv)} ${_language == 'ar' ? 'جنيه' :
'EGP'} (${_formatDate(item['dueDate'] as String)}) (${item['isPaid'] == 1 ? (_language == 'ar' ?
'مدفوعة' : 'Paid') : (_language == 'ar' ? 'غير مدفوعة' : 'Unpaid')})',
);
}),
pw.SizedBox(height: 10),
];
},
),
);

// حفظ ومشاركة ملف PDF
final output = await pdf.save();
await Printing.sharePdf(bytes: output, filename: 'financial_report.pdf');

ScaffoldMessenger.of(context).showSnackBar(
SnackBar(
content: Text(_language == 'ar' ? 'تم تصدير التقرير بنجاح' : 'Report exported successfully'),
),
);
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء تصدير التقرير: $e' : 'Error exporting report:
$e')),
);
}
}

Future<void> _localBackup() async {
try {
final spendingData = await _database.query('spending');
final incomesData = await _database.query('incomes');
final loansData = await _database.query('loans');
final budgetsData = await _database.query('budgets');
final savingsGoalsData = await _database.query('savings_goals');
final billsData = await _database.query('bills');
final categoriesData = await _database.query('categories');
```

```
final backupData = {
'spending': spendingData,
'incomes': incomesData,
'loans': loansData,
'budgets': budgetsData,
'savings_goals': savingsGoalsData,
'bills': billsData,
'categories': categoriesData,
'timestamp': DateTime.now().toIso8601String(),
};

final jsonString = jsonEncode(backupData);
final prefs = await SharedPreferences.getInstance();
await prefs.setString(_backupKey, jsonString);

// يمكن حفظ النسخة الاحتياطية كملف أيضًا إذا أردت
final backupDir = await getTemporaryDirectory();
final backupFile = File('${backupDir.path}/financial_backup.json');
await backupFile.writeAsString(jsonString);

ScaffoldMessenger.of(context).showSnackBar(
SnackBar(
content: Text(_language == 'ar' ? 'تم إنشاء النسخة الاحتياطية بنجاح' : 'Backup created successfully'),
),
);
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'خطأ أثناء النسخ الاحتياطي: $e' : 'Error during backup: $e')),
);
}
}

Future<void> _localRestore() async {
try {
final prefs = await SharedPreferences.getInstance();
final String? jsonString = prefs.getString(_backupKey);

if (jsonString == null) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'لا توجد نسخة احتياطية سابقة' : 'No previous backup found')),
);
```

```
return;
}

final Map<String, dynamic> backupData = jsonDecode(jsonString);

// تأكيد من المستخدم قبل مسح البيانات
bool? confirm = await showDialog<bool>(
context: context,
builder: (BuildContext dialogContext) {
return AlertDialog(
title: Text(_language == 'ar' ? 'تأكيد الاستعادة' : 'Confirm Restore'),
content: Text(_language == 'ar'
? 'هل أنت متأكد أنك تريد استعادة البيانات؟ هذا سيحل محل البيانات الحالية.'
: 'Are you sure you want to restore data? This will overwrite current data.'),
actions: <Widget>[
TextButton(
child: Text(_language == 'ar' ? 'إلغاء' : 'Cancel'),
onPressed: () => Navigator.of(dialogContext).pop(false),
),
TextButton(
child: Text(_language == 'ar' ? 'استعادة' : 'Restore'),
onPressed: () => Navigator.of(dialogContext).pop(true),
),
],
);
},
);

if (confirm != true) {
return;
}

// مسح البيانات الحالية قبل الاستعادة
await _database.delete('spending');
await _database.delete('incomes');
await _database.delete('loans');
await _database.delete('budgets');
await _database.delete('savings_goals');
await _database.delete('bills');
await _database.delete('categories');

// إدراج البيانات المستعادة
for (var item in (backupData['spending'] as List)) {
await _database.insert('spending', item);
```

```
      }
    for (var item in (backupData['incomes'] as List)) {
      await _database.insert('incomes', item);
      }
    for (var item in (backupData['loans'] as List)) {
      await _database.insert('loans', item);
      }
    for (var item in (backupData['budgets'] as List)) {
      await _database.insert('budgets', item);
      }
  for (var item in (backupData['savings_goals'] as List)) {
      await _database.insert('savings_goals', item);
      }
    for (var item in (backupData['bills'] as List)) {
      await _database.insert('bills', item);
      }
  for (var item in (backupData['categories'] as List)) {
      await _database.insert('categories', item);
      }

    // إعادة جلب البيانات لتحديث واجهة المستخدم
      await _fetchSpendingData();
      await _fetchLoansData();
      await _fetchBudgetsData();
      await _fetchIncomesData();
  await _fetchSavingsGoalsData();
      await _fetchBillsData();
  await _fetchCateg// ... (الكود السابق لدالة _localRestore) ...

    // إعادة جلب البيانات لتحديث واجهة المستخدم
      await _fetchSpendingData();
      await _fetchLoansData();
      await _fetchBudgetsData();
      await _fetchIncomesData();
  await _fetchSavingsGoalsData();
      await _fetchBillsData();
  await _fetchCategoriesData(); // هذا هو الاستدعاء المفقود

    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
    content: Text(_language == 'ar'
        ? 'تم استعادة البيانات بنجاح!'
        : 'Data restored successfully!')),
      );
```

```
} catch (e) {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(
content: Text(_language == 'ar'
'$e' :خطأ أثناء الاستعادة' ?
: 'Error during restore: $e')),
);
}
}

// ---------------------------------------------------------------------------
(عناصر واجهة المستخدم (أمثلة - 8️⃣ UI Elements 🔹 //
// ---------------------------------------------------------------------------

.هذه أمثلة لعناصر واجهة المستخدم. ستحتاج إلى بناء الشاشات الفعلية //

@override
Widget build(BuildContext context) {
if (_isLoading) {
return Scaffold(
appBar: AppBar(
title: Text(_language == 'ar' ? 'جاري التحميل...' : 'Loading...'),
),
body: const Center(child: CircularProgressIndicator()),
);
}

return Scaffold(
appBar: AppBar(
title: Text(_language == 'ar' ? 'إدارة المالية الشخصية' : 'Personal Financial Manager'),
actions: [
IconButton(
icon: Icon(_isStealthMode ? Icons.visibility_off : Icons.visibility),
onPressed: () => _setStealthMode(!_isStealthMode),
tooltip: _language == 'ar' ? 'وضع التخفي' : 'Stealth Mode',
),
IconButton(
icon: Icon(_isDarkMode ? Icons.wb_sunny : Icons.nightlight_round),
onPressed: () => _setThemePreference(!_isDarkMode),
tooltip: _language == 'ar' ? 'تغيير الثيم' : 'Toggle Theme',
),
IconButton(
icon: const Icon(Icons.language),
onPressed: _showLanguageDialog,
```

```
tooltip: _language == 'ar' ? 'تغيير اللغة' : 'Change Language',
),
PopupMenuButton<String>(
onSelected: (value) async {
if (value == 'export_pdf') {
await _exportReportAsPDF();
} else if (value == 'backup') {
await _localBackup();
} else if (value == 'restore') {
await _localRestore();
}
},
itemBuilder: (BuildContext context) => <PopupMenuEntry<String>>[
PopupMenuItem<String>(
value: 'export_pdf',
child: Text(_language == 'ar' ? 'تصدير كـ PDF' : 'Export as PDF'),
),
PopupMenuItem<String>(
value: 'backup',
child: Text(_language == 'ar' ? 'نسخ احتياطي' : 'Backup'),
),
PopupMenuItem<String>(
value: 'restore',
child: Text(_language == 'ar' ? 'استعادة بيانات' : 'Restore Data'),
),
],
),
],
),
drawer: Drawer(
child: ListView(
padding: EdgeInsets.zero,
children: <Widget>[
DrawerHeader(
decoration: BoxDecoration(
color: Theme.of(context).primaryColor,
),
child: Text(
_language == 'ar' ? 'القائمة' : 'Menu',
style: const TextStyle(
color: Colors.white,
fontSize: 24,
),
),
```

```
),
ListTile(
leading: const Icon(Icons.monetization_on),
title: Text(_language == 'ar' ? 'إدارة الإنفاق' : 'Manage Spending'),
onTap: () {
Navigator.pop(context);
// navigate to spending screen
Navigator.push(context, MaterialPageRoute(builder: (context) => SpendingScreen(
addSpending: _addSpending,
deleteSpending: _deleteSpending,
spendingData: _spendingData,
categoriesData: _categoriesData,
language: _language,
isStealthMode: _isStealthMode,
formatDate: _formatDate,
)));
},
),
ListTile(
leading: const Icon(Icons.account_balance_wallet),
title: Text(_language == 'ar' ? 'إدارة الدخل' : 'Manage Income'),
onTap: () {
Navigator.pop(context);
// navigate to income screen
Navigator.push(context, MaterialPageRoute(builder: (context) => IncomeScreen(
addIncome: _addIncome,
deleteIncome: _deleteIncome,
incomesData: _incomesData,
language: _language,
isStealthMode: _isStealthMode,
formatDate: _formatDate,
)));
},
),
ListTile(
leading: const Icon(Icons.handshake),
title: Text(_language == 'ar' ? 'القروض والديون' : 'Loans and Debts'),
onTap: () {
Navigator.pop(context);
// navigate to loans screen
Navigator.push(context, MaterialPageRoute(builder: (context) => LoansScreen(
addLoan: _addLoan,
deleteLoan: _deleteLoan,
updateLoanStatus: _updateLoanStatus,
```

```
                loansData: _loansData,
                language: _language,
                isStealthMode: _isStealthMode,
              )));
            },
          ),
          ListTile(
            leading: const Icon(Icons.pie_chart),
            title: Text(_language == 'ar' ? 'الميزانيات' : 'Budgets'),
            onTap: () {
              Navigator.pop(context);
              // navigate to budgets screen
              Navigator.push(context, MaterialPageRoute(builder: (context) => BudgetScreen(
                addBudget: _addBudget,
                deleteBudget: _deleteBudget,
                budgetsData: _budgetsData,
                categoriesData: _categoriesData,
                getSpendingForCategory: _getSpendingForCategory,
                language: _language,
              )));
            },
          ),
          ListTile(
            leading: const Icon(Icons.savings),
            title: Text(_language == 'ar' ? 'أهداف الادخار' : 'Savings Goals'),
            onTap: () {
              Navigator.pop(context);
              // navigate to savings goals screen
              Navigator.push(context, MaterialPageRoute(builder: (context) => SavingsGoalScreen(
                addSavingsGoal: _addSavingsGoal,
                deleteSavingsGoal: _deleteSavingsGoal,
                savingsGoalsData: _savingsGoalsData,
                language: _language,
                isStealthMode: _isStealthMode,
              )));
            },
          ),
          ListTile(
            leading: const Icon(Icons.receipt_long),
            title: Text(_language == 'ar' ? 'الفواتير' : 'Bills'),
            onTap: () {
              Navigator.pop(context);
              // navigate to bills screen
              Navigator.push(context, MaterialPageRoute(builder: (context) => BillsScreen(
```

```
                addBill: _addBill,
                deleteBill: _deleteBill,
                updateBillStatus: _updateBillStatus,
                billsData: _billsData,
                language: _language,
                isStealthMode: _isStealthMode,
                formatDate: _formatDate,
              )));
            },
          ),
          ListTile(
            leading: const Icon(Icons.category),
            title: Text(_language == 'ar' ? 'إدارة الفئات' : 'Manage Categories'),
            onTap: () {
              Navigator.pop(context);
              // navigate to categories screen
              Navigator.push(context, MaterialPageRoute(builder: (context) => CategoryScreen(
                addCategory: _addCategory,
                deleteCategory: _deleteCategory,
                categoriesData: _categoriesData,
                language: _language,
              )));
            },
          ),
          ListTile(
            leading: const Icon(Icons.compare_arrows),
            title: Text(_language == 'ar' ? 'تحويل بين الحسابات' : 'Transfer Between Accounts'),
            onTap: () {
              Navigator.pop(context);
              // navigate to transfer screen
              _showTransferDialog();
            },
          ),
          ListTile(
            leading: const Icon(Icons.search),
            title: Text(_language == 'ar' ? 'بحث المعاملات' : 'Search Transactions'),
            onTap: () {
              Navigator.pop(context);
              _showSearchDialog();
            },
          ),
          ListTile(
            leading: const Icon(Icons.bar_chart),
            title: Text(_language == 'ar' ? 'التقارير والإحصائيات' : 'Reports & Statistics'),
```

```
onTap: () {
Navigator.pop(context);
Navigator.push(context, MaterialPageRoute(builder: (context) => ReportsScreen(
spendingData: _spendingData,
incomesData: _incomesData,
categoriesData: _categoriesData,
language: _language,
formatDate: _formatDate,
encrypter: encrypter,
iv: iv,
)));
},
),
],
),
),
body: Center(
child: Column(
mainAxisAlignment: MainAxisAlignment.center,
children: [
Text(
_language == 'ar' ? 'مرحبًا بك في تطبيق إدارة المالية!' : 'Welcome to your Financial Manager!',
style: Theme.of(context).textTheme.headlineMedium,
textAlign: TextAlign.center,
),
const SizedBox(height: 20),
// عرض ملخص سريع للبيانات (يمكن تحسينه)
_buildSummaryCard(),
],
),
),
);
}

// ---------------------------------------------------------------------------
// 🔹 9️⃣ UI Helper Functions - دوال مساعدة لواجهة المستخدم
// ---------------------------------------------------------------------------

Widget _buildSummaryCard() {
double totalIncome = _incomesData.fold(0.0, (sum, item) {
if (_isStealthMode) return 0.0; // إخفاء الأرقام في وضع التخفي
return sum + (double.tryParse(encrypter.decrypt64(item.amount, iv: iv)) ?? 0.0);
});
double totalSpending = _spendingData.fold(0.0, (sum, item) {
```

```
إخفاء الأرقام في وضع التخفي // ;if (_isStealthMode) return 0.0
return sum + (double.tryParse(encrypter.decrypt64(item.amount, iv: iv)) ?? 0.0);
});
double netBalance = totalIncome - totalSpending;

return Card(
margin: const EdgeInsets.all(16),
child: Padding(
padding: const EdgeInsets.all(16.0),
child: Column(
crossAxisAlignment: CrossAxisAlignment.start,
children: [
Text(
_language == 'ar' ? 'ملخص مالي سريع' : 'Quick Financial Summary',
style: Theme.of(context).textTheme.titleLarge,
),
const Divider(),
_isStealthMode
? Text(_language == 'ar' ? 'الأرقام مخفية في وضع التخفي' : 'Numbers hidden in stealth mode',
style: const TextStyle(fontStyle: FontStyle.italic))
: Column(
children: [
Row(
mainAxisAlignment: MainAxisAlignment.spaceBetween,
children: [
Text(_language == 'ar' ? 'إجمالي الدخل:' : 'Total Income:', style:
Theme.of(context).textTheme.bodyLarge),
Text('${totalIncome.toStringAsFixed(2)} ${_language == 'ar' ? 'جنيه' : 'EGP'}',
style: Theme.of(context).textTheme.bodyLarge?.copyWith(color: Colors.green)),
],
),
Row(
mainAxisAlignment: MainAxisAlignment.spaceBetween,
children: [
Text(_language == 'ar' ? 'إجمالي الإنفاق:' : 'Total Spending:', style:
Theme.of(context).textTheme.bodyLarge),
Text('${totalSpending.toStringAsFixed(2)} ${_language == 'ar' ? 'جنيه' : 'EGP'}',
style: Theme.of(context).textTheme.bodyLarge?.copyWith(color: Colors.red)),
],
),
const SizedBox(height: 8),
Row(
mainAxisAlignment: MainAxisAlignment.spaceBetween,
children: [
```

```
Text(_language == 'ar' ? 'الرصيد الصافي:' : 'Net Balance:', style:
Theme.of(context).textTheme.headlineSmall),
Text('${netBalance.toStringAsFixed(2)} ${_language == 'ar' ? 'جنيه' : 'EGP'}',
style: Theme.of(context).textTheme.headlineSmall?.copyWith(color: netBalance >= 0 ?
Colors.green : Colors.red)),
],
),
],
),
],
),
),
);
}

void _showTransferDialog() {
final TextEditingController amountController = TextEditingController();
String? fromAccount;
String? toAccount;

showDialog(
context: context,
builder: (context) {
return AlertDialog(
title: Text(_language == 'ar' ? 'تحويل مبلغ' : 'Transfer Amount'),
content: Column(
mainAxisSize: MainAxisSize.min,
children: [
DropdownButtonFormField<String>(
decoration: InputDecoration(labelText: _language == 'ar' ? 'من حساب' : 'From Account'),
value: fromAccount,
items: <String>['incomes', 'savings_goals', 'general_spending']
.map<DropdownMenuItem<String>>((String value) {
return DropdownMenuItem<String>(
value: value,
child: Text(_language == 'ar' ? (value == 'incomes' ? 'الدخل' : (value ==
'savings_goals' ? 'المصروفات العامة' : 'المدخرات')) : (value == 'incomes' ? 'Income' : (value ==
'savings_goals' ? 'Savings' : 'General Spending'))),
);
}).toList(),
onChanged: (String? newValue) {
fromAccount = newValue;
},
),
```

```
DropdownButtonFormField<String>(
decoration: InputDecoration(labelText: _language == 'ar' ? 'إلى حساب' : 'To Account'),
value: toAccount,
items: <String>['incomes', 'savings_goals', 'general_spending']
.map<DropdownMenuItem<String>>((String value) {
return DropdownMenuItem<String>(
value: value,
child: Text(_language == 'ar' ? (value == 'incomes' ? 'الدخل' : (value ==
'savings_goals' ? 'المصروفات العامة' : 'المدخرات')) : (value == 'incomes' ? 'Income' : (value ==
'savings_goals' ? 'Savings' : 'General Spending'))),
);
}).toList(),
onChanged: (String? newValue) {
toAccount = newValue;
},
),
TextField(
controller: amountController,
keyboardType: TextInputType.number,
decoration: InputDecoration(labelText: _language == 'ar' ? 'المبلغ' : 'Amount'),
),
],
),
actions: [
TextButton(
onPressed: () {
Navigator.pop(context);
},
child: Text(_language == 'ar' ? 'إلغاء' : 'Cancel'),
),
ElevatedButton(
onPressed: () {
if (fromAccount != null &&
toAccount != null &&
amountController.text.isNotEmpty) {
final amount = double.tryParse(amountController.text);
if (amount != null && amount > 0) {
_transferBetweenAccounts(fromAccount!, toAccount!, amount);
Navigator.pop(context);
} else {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'الرجاء إدخال مبلغ صحيح.' : 'Please enter a
valid amount.')),
);
```

```
                }
              } else {
                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(content: Text(_language == 'ar' ? 'الرجاء ملء جميع الحقول.' : 'Please fill all fields.')),
                );
              }
            },
            child: Text(_language == 'ar' ? 'تحويل' : 'Transfer'),
          ),
        ],
      );
    },
  );
}

void _showSearchDialog() {
  final TextEditingController queryController = TextEditingController();
  DateTime? selectedStartDate;
  DateTime? selectedEndDate;
  final TextEditingController minAmountController = TextEditingController();
  final TextEditingController maxAmountController = TextEditingController();
  String? selectedTable;

  showDialog(
    context: context,
    builder: (context) {
      return StatefulBuilder(
        builder: (context, setState) {
          return AlertDialog(
            title: Text(_language == 'ar' ? 'بحث المعاملات' : 'Search Transactions'),
            content: SingleChildScrollView(
              child: Column(
                mainAxisSize: MainAxisSize.min,
                children: [
                  TextField(
                    controller: queryController,
                    decoration: InputDecoration(labelText: _language == 'ar' ? 'كلمة البحث' : 'Search Query'),
                  ),
                  DropdownButtonFormField<String>(
                    decoration: InputDecoration(labelText: _language == 'ar' ? 'النوع (الجدول)' : 'Table (Type)'),
                    value: selectedTable,
```

```
items: <String>[
'spending',
'incomes',
'loans',
'bills',
'savings_goals'
].map<DropdownMenuItem<String>>((String value) {
String displayValue = '';
if (value == 'spending') displayValue = _language == 'ar' ? 'الإنفاق' : 'Spending';
if (value == 'incomes') displayValue = _language == 'ar' ? 'الدخل' : 'Income';
if (value == 'loans') displayValue = _language == 'ar' ? 'القروض' : 'Loans';
if (value == 'bills') displayValue = _language == 'ar' ? 'الفواتير' : 'Bills';
if (value == 'savings_goals') displayValue = _language == 'ar' ? 'أهداف الادخار' :
'Savings Goals';
return DropdownMenuItem<String>(
value: value,
child: Text(displayValue),
);
}).toList(),
onChanged: (String? newValue) {
setState(() {
selectedTable = newValue;
});
},
),
ListTile(
title: Text(_language == 'ar' ? 'من تاريخ:' : 'Start Date:'),
subtitle: Text(selectedStartDate == null ? (_language == 'ar' ? 'لم يتم الاختيار' : 'Not
selected') : _formatDate(selectedStartDate!.toIso8601String())),
trailing: const Icon(Icons.calendar_today),
onTap: () async {
final DateTime? picked = await showDatePicker(
context: context,
initialDate: selectedStartDate ?? DateTime.now(),
firstDate: DateTime(2000),
lastDate: DateTime(2101),
);
if (picked != null && picked != selectedStartDate) {
setState(() {
selectedStartDate = picked;
});
}
},
),
```

```
ListTile(
title: Text(_language == 'ar' ? 'إلى تاريخ:' : 'End Date:'),
subtitle: Text(selectedEndDate == null ? (_language == 'ar' ? 'لم يتم الاختيار' : 'Not selected') : _formatDate(selectedEndDate!.toIso8601String())),
trailing: const Icon(Icons.calendar_today),
onTap: () async {
final DateTime? picked = await showDatePicker(
context: context,
initialDate: selectedEndDate ?? DateTime.now(),
firstDate: DateTime(2000),
lastDate: DateTime(2101),
);
if (picked != null && picked != selectedEndDate) {
setState(() {
selectedEndDate = picked;
});
}
},
),
TextField(
controller: minAmountController,
keyboardType: TextInputType.number,
decoration: InputDecoration(labelText: _language == 'ar' ? 'الحد الأدنى للمبلغ' : 'Min Amount'),
),
TextField(
controller: maxAmountController,
keyboardType: TextInputType.number,
decoration: InputDecoration(labelText: _language == 'ar' ? 'الحد الأقصى للمبلغ' : 'Max Amount'),
),
],
),
),
actions: [
TextButton(
onPressed: () {
Navigator.pop(context);
},
child: Text(_language == 'ar' ? 'إلغاء' : 'Cancel'),
),
ElevatedButton(
onPressed: () async {
if (selectedTable == null || selectedTable!.isEmpty) {
```

```
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(_language == 'ar' ? 'الرجاء اختيار نوع المعاملة للبحث.' : 'Please select a transaction type to search.')),
);
return;
}

final query = queryController.text;
final minAmount = double.tryParse(minAmountController.text);
final maxAmount = double.tryParse(maxAmountController.text);

final results = await _searchTransactions(
query,
selectedTable!,
startDate: selectedStartDate,
endDate: selectedEndDate,
minAmount: minAmount,
maxAmount: maxAmount,
);

Navigator.pop(context); // إغلاق نافذة البحث
_showSearchResultsDialog(results, selectedTable!); // عرض النتائج
},
child: Text(_language == 'ar' ? 'بحث' : 'Search'),
),
],
);
},
);
},
);
}

void _showSearchResultsDialog(List<Map<String, dynamic>> results, String table) {
showDialog(
context: context,
builder: (context) {
return AlertDialog(
title: Text(_language == 'ar' ? 'نتائج البحث' : 'Search Results'),
content: results.isEmpty
? Text(_language == 'ar' ? 'لا توجد نتائج.' : 'No results found.')
: SizedBox(
width: double.maxFinite,
child: ListView.builder(
```

```
shrinkWrap: true,
itemCount: results.length,
itemBuilder: (context, index) {
final item = results[index];
String title = '';
String subtitle = '';
double displayAmount = 0.0;

// فك تشفير المبلغ للعرض
if (item.containsKey('amount') && item['amount'] is String) {
displayAmount = double.tryParse(encrypter.decrypt64(item['amount'] as String,
iv: iv)) ?? 0.0;
} else if (item.containsKey('maxAmount')) { // للميزانيات
displayAmount = item['maxAmount'] as double;
} else if (item.containsKey('targetAmount') && item.containsKey('currentAmount'))
{ // لأهداف الادخار
final target = double.tryParse(encrypter.decrypt64(item['targetAmount'] as String,
iv: iv)) ?? 0.0;
final current = double.tryParse(encrypter.decrypt64(item['currentAmount'] as
String, iv: iv)) ?? 0.0;
subtitle += _language == 'ar' ? 'هدف: ${target.toStringAsFixed(2)}، حالي:
${current.toStringAsFixed(2)}' : 'Target: ${target.toStringAsFixed(2)}, Current:
${current.toStringAsFixed(2)}';
}


if (table == 'spending') {
title = item['category'] as String;
subtitle = '${displayAmount.toStringAsFixed(2)} ${_language == 'ar' ? 'جنيه' :
'EGP'} - ${_formatDate(item['date'] as String)}';
} else if (table == 'incomes') {
title = item['source'] as String;
subtitle = '${displayAmount.toStringAsFixed(2)} ${_language == 'ar' ? 'جنيه' :
'EGP'} - ${_formatDate(item['date'] as String)}';
} else if (table == 'loans') {
title = item['friend'] as String;
subtitle = '${displayAmount.toStringAsFixed(2)} ${_language == 'ar' ? 'جنيه' :
'EGP'} - (${item['isPaid'] == 1 ? (_language == 'ar' ? 'تم السداد' : 'Paid') : (_language == 'ar' ? 'لم يُسدد'
: 'Unpaid')})';
} else if (table == 'bills') {
title = item['name'] as String;
subtitle = '${displayAmount.toStringAsFixed(2)} ${_language == 'ar' ? 'جنيه' :
'EGP'} - ${_formatDate(item['dueDate'] as String)} (${item['isPaid'] == 1 ? (_language == 'ar' ?
'مدفوعة' : 'Paid') : (_language == 'ar' ? 'غير مدفوعة' : 'Unpaid')})';
```

```
} else if (table == 'savings_goals') {
title = item['goalName'] as String;
// Subtitle already handled above for savings_goals
}


return ListTile(
title: Text(_isStealthMode ? '****' : title),
subtitle: Text(_isStealthMode ? '****' : subtitle),
);
},
),
),
actions: [
TextButton(
onPressed: () {
Navigator.pop(context);
},
child: Text(_language == 'ar' ? 'إغلاق' : 'Close'),
),
],
);
},
);
}
}

// ----------------------------------------------------------------------------
// 🔹 10️⃣ Dedicated Screens - أمثلة مبدئية (شاشات منفصلة)
// ----------------------------------------------------------------------------

// مثال (spending_screen.dart): يمكنك إنشاء ملفات منفصلة لكل شاشة
// هذه مجرد هياكل لتوضيح كيفية تمرير البيانات والدوال.

// شاشة إدارة الإنفاق
class SpendingScreen extends StatefulWidget {
final Function(String amount, String category, int frequency, String date) addSpending;
final Function(int id) deleteSpending;
final List<SpendingHabit> spendingData;
final List<Category> categoriesData;
final String language;
final bool isStealthMode;
final String Function(String isoDate) formatDate;
```

```
const SpendingScreen({
super.key,
required this.addSpending,
required this.deleteSpending,
required this.spendingData,
required this.categoriesData,
required this.language,
required this.isStealthMode,
required this.formatDate,
});

@override
State<SpendingScreen> createState() => _SpendingScreenState();
}

class _SpendingScreenState extends State<SpendingScreen> {
final TextEditingController _amountController = TextEditingController();
String? _selectedCategory;
DateTime _selectedDate = DateTime.now();

@override
void initState() {
super.initState();
if (widget.categoriesData.isNotEmpty) {
_selectedCategory = widget.categoriesData.first.name;
}
}

@override
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
title: Text(widget.language == 'ar' ? 'إدارة الإنفاق' : 'Manage Spending'),
),
body: Padding(
padding: const EdgeInsets.all(16.0),
child: Column(
children: [
TextField(
controller: _amountController,
keyboardType: TextInputType.number,
decoration: InputDecoration(
labelText: widget.language == 'ar' ? 'المبلغ' : 'Amount',
border: const OutlineInputBorder(),
```

```
),
),
const SizedBox(height: 10),
DropdownButtonFormField<String>(
decoration: InputDecoration(
labelText: widget.language == 'ar' ? 'الفئة' : 'Category',
border: const OutlineInputBorder(),
),
value: _selectedCategory,
items: widget.categoriesData.map((category) {
return DropdownMenuItem(
value: category.name,
child: Text(category.name),
);
}).toList(),
onChanged: (newValue) {
setState(() {
_selectedCategory = newValue;
});
},
),
const SizedBox(height: 10),
ListTile(
title: Text(widget.language == 'ar' ? 'التاريخ' : 'Date'),
subtitle: Text(widget.formatDate(_selectedDate.toIso8601String())),
trailing: const Icon(Icons.calendar_today),
onTap: () async {
final DateTime? picked = await showDatePicker(
context: context,
initialDate: _selectedDate,
firstDate: DateTime(2000),
lastDate: DateTime(2101),
);
if (picked != null && picked != _selectedDate) {
setState(() {
_selectedDate = picked;
});
}
},
),
const SizedBox(height: 20),
ElevatedButton(
onPressed: () {
if (_amountController.text.isNotEmpty && _selectedCategory != null) {
```

```
widget.addSpending(
_addSpending _amountController.text, // سيتم تشفيرها داخل
_selectedCategory!,
1, // تردد افتراضي
_selectedDate.toIso8601String(),
);
_amountController.clear();
// لا داعي لتغيير _selectedCategory لأنها ستبقى الفئة المختارة أو تعود للأولى
// _selectedCategory = widget.categoriesData.isNotEmpty ?
widget.categoriesData.first.name : null;
setState(() {}); // لتحديث الواجهة
} else {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(widget.language == 'ar' ? 'الرجاء إدخال المبلغ واختيار الفئة.' :
'Please enter amount and select category.')),
);
}
},
child: Text(widget.language == 'ar' ? 'إضافة إنفاق' : 'Add Spending'),
),
const SizedBox(height: 20),
Expanded(
child: ListView.builder(
itemCount: widget.spendingData.length,
itemBuilder: (context, index) {
final item = widget.spendingData[index];
return Card(
margin: const EdgeInsets.symmetric(vertical: 8.0),
child: ListTile(
title: Text(widget.isStealthMode ? '****' : '${item.category}'),
subtitle: Text(widget.isStealthMode ? '****' : '${item.amount} ${_language == 'ar' ?
'جنيه' : 'EGP'} - ${widget.formatDate(item.date)}'),
trailing: IconButton(
icon: const Icon(Icons.delete),
onPressed: () => widget.deleteSpending(item.id),
),
),
);
},
),
),
],
),
),
```

```
);
}
}

// شاشة إدارة الدخل
class IncomeScreen extends StatelessWidget {
final Function(String source, double amount, String date) addIncome;
final Function(int id) deleteIncome;
final List<Income> incomesData;
final String language;
final bool isStealthMode;
final String Function(String isoDate) formatDate;

const IncomeScreen({
super.key,
required this.addIncome,
required this.deleteIncome,
required this.incomesData,
required this.language,
required this.isStealthMode,
required this.formatDate,
});

@override
Widget build(BuildContext context) {
final TextEditingController sourceController = TextEditingController();
final TextEditingController amountController = TextEditingController();
DateTime selectedDate = DateTime.now();

return Scaffold(
appBar: AppBar(
title: Text(language == 'ar' ? 'إدارة الدخل' : 'Manage Income'),
),
body: Padding(
padding: const EdgeInsets.all(16.0),
child: Column(
children: [
TextField(
controller: sourceController,
decoration: InputDecoration(
labelText: language == 'ar' ? 'المصدر' : 'Source',
border: const OutlineInputBorder(),
),
),
```

```
const SizedBox(height: 10),
TextField(
controller: amountController,
keyboardType: TextInputType.number,
decoration: InputDecoration(
labelText: language == 'ar' ? 'المبلغ' : 'Amount',
border: const OutlineInputBorder(),
),
),
const SizedBox(height: 10),
ListTile(
title: Text(language == 'ar' ? 'التاريخ' : 'Date'),
subtitle: StatefulBuilder(
builder: (context, setState) {
return Text(formatDate(selectedDate.toIso8601String()));
},
),
trailing: const Icon(Icons.calendar_today),
onTap: () async {
final DateTime? picked = await showDatePicker(
context: context,
initialDate: selectedDate,
firstDate: DateTime(2000),
lastDate: DateTime(2101),
);
if (picked != null && picked != selectedDate) {
selectedDate = picked;
(context as Element).markNeedsBuild(); // لإعادة بناء الـ StatefulBuilder
}
},
),
const SizedBox(height: 20),
ElevatedButton(
onPressed: () {
if (sourceController.text.isNotEmpty && amountController.text.isNotEmpty) {
final amount = double.tryParse(amountController.text);
if (amount != null && amount > 0) {
addIncome(sourceController.text, amount, selectedDate.toIso8601String());
sourceController.clear();
amountController.clear();
(context as Element).markNeedsBuild(); // لتحديث القائمة بعد الإضافة
} else {
ScaffoldMessenger.of(context).showSnackBar(
```

```
SnackBar(content: Text(language == 'ar' ? 'الرجاء إدخال مبلغ صحيح.' : 'Please enter a valid amount.')),
);
}
} else {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(language == 'ar' ? 'الرجاء ملء جميع الحقول.' : 'Please fill all fields.')),
);
}
},
child: Text(language == 'ar' ? 'إضافة دخل' : 'Add Income'),
),
const SizedBox(height: 20),
Expanded(
child: ListView.builder(
itemCount: incomesData.length,
itemBuilder: (context, index) {
final item = incomesData[index];
return Card(
margin: const EdgeInsets.symmetric(vertical: 8.0),
child: ListTile(
title: Text(isStealthMode ? '****' : item.source),
subtitle: Text(isStealthMode ? '****' : '${item.amount} ${language == 'ar' ? 'جنيه' : 'EGP'} - ${formatDate(item.date)}'),
trailing: IconButton(
icon: const Icon(Icons.delete),
onPressed: () => deleteIncome(item.id),
),
),
);
},
),
),
],
),
),
);
}
}

// شاشة إدارة القروض
class LoansScreen extends StatelessWidget {
final Function(String friend, String amount, bool isPaid) addLoan;
```

```
final Function(int id) deleteLoan;
final Function(int id, bool isPaid) updateLoanStatus;
final List<Loan> loansData;
final String language;
final bool isStealthMode;

const LoansScreen({
super.key,
required this.addLoan,
required this.deleteLoan,
required this.updateLoanStatus,
required this.loansData,
required this.language,
required this.isStealthMode,
});

@override
Widget build(BuildContext context) {
final TextEditingController friendController = TextEditingController();
final TextEditingController amountController = TextEditingController();

return Scaffold(
appBar: AppBar(
title: Text(language == 'ar' ? 'القروض والديون' : 'Loans and Debts'),
),
body: Padding(
padding: const EdgeInsets.all(16.0),
child: Column(
children: [
TextField(
controller: friendController,
decoration: InputDecoration(
labelText: language == 'ar' ? 'اسم الصديق/الجهة' : 'Friend/Entity Name',
border: const OutlineInputBorder(),
),
),
const SizedBox(height: 10),
TextField(
controller: amountController,
keyboardType: TextInputType.number,
decoration: InputDecoration(
labelText: language == 'ar' ? 'المبلغ' : 'Amount',
border: const OutlineInputBorder(),
),
```

```
),
const SizedBox(height: 20),
ElevatedButton(
onPressed: () {
if (friendController.text.isNotEmpty && amountController.text.isNotEmpty) {
final amount = double.tryParse(amountController.text);
if (amount != null && amount > 0) {
addLoan(friendController.text, amount.toString(), false); // يتم تشفير المبلغ داخل _addLoan
friendController.clear();
amountController.clear();
(context as Element).markNeedsBuild();
} else {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(language == 'ar' ? 'الرجاء إدخال مبلغ صحيح.' : 'Please enter a valid amount.')),
);
}
} else {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(language == 'ar' ? 'الرجاء ملء جميع الحقول.' : 'Please fill all fields.')),
);
}
},
child: Text(language == 'ar' ? 'إضافة قرض' : 'Add Loan'),
),
const SizedBox(height: 20),
Expanded(
child: ListView.builder(
itemCount: loansData.length,
itemBuilder: (context, index) {
final loan = loansData[index];
return Card(
margin: const EdgeInsets.symmetric(vertical: 8.0),
child: ListTile(
title: Text(isStealthMode ? '****' : loan.friend),
subtitle: Text(isStealthMode ? '****' : '${loan.amount} ${language == 'ar' ? 'جنيه' : 'EGP'} - ${loan.isPaid ? (language == 'ar' ? 'تم السداد' : 'Paid') : (language == 'ar' ? 'لم يُسدد' : 'Unpaid')}'),
trailing: Row(
mainAxisSize: MainAxisSize.min,
children: [
IconButton(
icon: Icon(Icons.check, color: loan.isPaid ? Colors.green : Colors.grey),
```

```
                onPressed: () => updateLoanStatus(loan.id, !loan.isPaid),
              ),
              IconButton(
                icon: const Icon(Icons.delete),
                onPressed: () => deleteLoan(loan.id),
              ),
            ],
          ),
        ),
      );
    },
  ),
),
],
),
),
);
}
}

// شاشة إدارة الميزانيات
class BudgetScreen extends StatefulWidget {
  final Function(String category, double maxAmount, String period) addBudget;
  final Function(int id) deleteBudget;
  final List<Budget> budgetsData;
  final List<Category> categoriesData;
  final Future<double> Function(String category, String period) getSpendingForCategory;
  final String language;

  const BudgetScreen({
    super.key,
    required this.addBudget,
    required this.deleteBudget,
    required this.budgetsData,
    required this.categoriesData,
    required this.getSpendingForCategory,
    required this.language,
  });

  @override
  State<BudgetScreen> createState() => _BudgetScreenState();
}

class _BudgetScreenState extends State<BudgetScreen> {
```

```
final TextEditingController _maxAmountController = TextEditingController();
String? _selectedCategory;
String _selectedPeriod = 'monthly'; // الافتراضي

@override
void initState() {
super.initState();
if (widget.categoriesData.isNotEmpty) {
_selectedCategory = widget.categoriesData.first.name;
}
}

@override
Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(
title: Text(widget.language == 'ar' ? 'الميزانيات' : 'Budgets'),
),
body: Padding(
padding: const EdgeInsets.all(16.0),
child: Column(
children: [
DropdownButtonFormField<String>(
decoration: InputDecoration(
labelText: widget.language == 'ar' ? 'الفئة' : 'Category',
border: const OutlineInputBorder(),
),
value: _selectedCategory,
items: widget.categoriesData.map((category) {
return DropdownMenuItem(
value: category.name,
child: Text(category.name),
);
}).toList(),
onChanged: (newValue) {
setState(() {
_selectedCategory = newValue;
});
},
),
const SizedBox(height: 10),
TextField(
controller: _maxAmountController,
keyboardType: TextInputType.number,
```

```
decoration: InputDecoration(
labelText: widget.language == 'ar' ? 'الحد الأقصى للميزانية' : 'Max Budget Amount',
border: const OutlineInputBorder(),
),
),
const SizedBox(height: 10),
DropdownButtonFormField<String>(
decoration: InputDecoration(
labelText: widget.language == 'ar' ? 'الفترة' : 'Period',
border: const OutlineInputBorder(),
),
value: _selectedPeriod,
items: <String>['monthly', 'weekly'].map((String value) {
return DropdownMenuItem<String>(
value: value,
child: Text(widget.language == 'ar' ? (value == 'monthly' ? 'شهري' : 'أسبوعي') : (value ==
'monthly' ? 'Monthly' : 'Weekly')),
);
}).toList(),
onChanged: (String? newValue) {
setState(() {
_selectedPeriod = newValue!;
});
},
),
const SizedBox(height: 20),
ElevatedButton(
onPressed: () {
if (_selectedCategory != null && _maxAmountController.text.isNotEmpty) {
final maxAmount = double.tryParse(_maxAmountController.text);
if (maxAmount != null && maxAmount > 0) {
widget.addBudget(_selectedCategory!, maxAmount, _selectedPeriod);
_maxAmountController.clear();
setState(() {});
} else {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(widget.language == 'ar' ? 'الرجاء إدخال مبلغ صحيح.' : 'Please
enter a valid amount.')),
);
}
} else {
ScaffoldMessenger.of(context).showSnackBar(
SnackBar(content: Text(widget.language == 'ar' ? 'الرجاء ملء جميع الحقول.' : 'Please fill
all fields.')),
```

```
);
}
},
child: Text(widget.language == 'ar' ? 'إضافة ميزانية' : 'Add Budget'),
),
const SizedBox(height: 20),
Expanded(
child: ListView.builder(
itemCount: widget.budgetsData.length,
itemBuilder: (context, index) {
final budget = widget.budgetsData[index];
return FutureBuilder<double>(
future: widget.getSpendingForCategory(budget.category, budget.period),
builder: (context, snapshot) {
final currentSpent = snapshot.data ?? 0.0;
final percentage = (currentSpent / budget.maxAmount) * 100;
final color = percentage >= 100 ? Colors.red : (percentage >= 80 ? Colors.orange
: Colors.green);
return Card(
margin: const EdgeInsets.symmetric(vertical: 8.0),
child: ListTile(
title: Text(budget.category),
subtitle: Column(
crossAxisAlignment: CrossAxisAlignment.start,
children: [
Text('${widget.language == 'ar' ? 'الميزانية القصوى' : 'Max Budget'}:
${budget.maxAmount.toStringAsFixed(2)} ${widget.language == 'ar' ? 'جنيه' : 'EGP'}
(${budget.period == 'monthly' ? (widget.language == 'ar' ? 'شهري' : 'Monthly') : (widget.language
== 'ar' ? 'أسبوعي' : 'Weekly')})'),
Text(
'${widget.language == 'ar' ? 'المنفق حالياً' : 'Currently Spent'}:
${currentSpent.toStringAsFixed(2)} ${widget.language == 'ar' ? 'جنيه' : 'EGP'}
(${percentage.toStringAsFixed(1)}%)',
style: TextStyle(color: color, fontWeight: FontWeight.bold),
),
],
),
trailing: IconButton(
icon: const Icon(Icons.delete),
onPressed: () => widget.deleteBudget(budget.id),
),
),
);
},
```

```
);
},
),
),
],
),
),
);
 }
 }

شاشة أهداف الادخار //
class SavingsGoalScreen extends StatelessWidget {
final Function(String goalName, double targetAmount, double currentAmount)
addSavingsGoal;
final Function(int id) deleteSavingsGoal;
final List<SavingsGoal> savingsGoalsData;
final String language;
final bool isStealthMode;

const SavingsGoalScreen({
super.key,
required this.addSavingsGoal,
required this.deleteSavingsGoal,
required this.savingsGoalsData,
required this.language,
required this.isStealthMode,
});

@override
Widget build(BuildContext context) {
final TextEditingController goalNameController = TextEditingController();
final TextEditingController targetAmountController = TextEditingController();
final TextEditingController currentAmountController = TextEditingController();

return Scaffold(
appBar: AppBar(
title: Te
```