# Pravass — Flutter Tourism App: Complete Blockchain Integration Guide (Beginner-friendly)

Guide Verification • Handicraft Verification (NFT) • Secure Payments (Escrow)  
Sepolia (testnet) demo with Flutter (web3dart)

Generated by ChatGPT — Full end-to-end instructions, code & assets included

## Executive Summary

This document explains step-by-step how to integrate blockchain features into Pravass, a Flutter-based tourism app.   
It covers: what blockchain is and why it helps, prerequisites, detailed Solidity contracts (GuideRegistry, HandicraftNFT, Escrow),   
how to deploy them on Sepolia using Remix and MetaMask, how to connect Flutter using web3dart (or WalletConnect), full Flutter code (services, screens, main.dart, pubspec.yaml), project structure, ABI usage, flowcharts, security considerations, and testing/run instructions.   
All code blocks are highlighted. This is intended to be self-contained so you can build a mini demo end-to-end.

## 1) What is Blockchain & Why Pravass Needs It

Simple explanation:  
- Blockchain is a distributed ledger where transactions are grouped into cryptographically linked blocks. Each block references the previous block by hash, creating tamper-evidence. Public blockchains (like Ethereum) are decentralized and allow deployment of smart contracts — small programs that run deterministically on-chain.  
- Smart contracts enable verifiable logic: storing data (like verified guide IDs), issuing tokens/NFTs (like authenticated handicrafts), and escrow/payment logic with transparent rules (like deposit -> hold -> release).  
  
Why it fits Pravass:  
- Trust: Tourists can independently verify guide credentials stored on-chain rather than relying solely on a central server.  
- Provenance & Authenticity: Minted NFTs for tribal handicrafts provide immutable proof of origin and ownership history.  
- Secure payments: Using an escrow smart contract reduces risk of fraud; funds are only released when conditions are met (tourist confirms or guide marks completed).  
- Auditability: All important events (registrations, verifications, mints, payments) are on-chain and auditable via transaction hashes or block explorers (Sepolia/Goerli/Etherscan).  
  
Limitations & trade-offs:  
- Gas costs on mainnet (use testnets like Sepolia for demo).   
- On-chain storage is expensive — store large files off-chain (IPFS) and put references (tokenURI) on-chain.  
- Usability: users need wallets and some UX (WalletConnect) for signing transactions.

## 2) Full list of prerequisites (detailed, all free)

Hardware & OS:  
- Standard laptop/PC (Windows/macOS/Linux) with internet.

Software & accounts (free):  
1. Flutter SDK & Dart — follow the official install guide for your OS. Verify with `flutter doctor`.  
2. Code editor: VS Code (recommended) + Flutter & Dart extensions.  
3. Android Studio (or Android SDK) for emulator OR use a physical Android device and enable USB debugging.  
4. MetaMask browser extension (Chrome/Firefox) and MetaMask mobile app (for WalletConnect flows).  
5. Remix IDE — web version: https://remix.ethereum.org (no install).  
6. Infura or Alchemy free account — to get Sepolia RPC endpoints (HTTP WebSocket). You can also use public RPCs but Infura/Alchemy are reliable.  
7. Sepolia testnet ETH from faucets (claim test ETH to pay gas). Examples: Alchemy faucet, Sepolia-specific faucets (search 'Sepolia faucet').  
8. Optional: Node.js + npm (only if you want to use Hardhat/Truffle locally).  
9. Flutter packages (add to `pubspec.yaml`): `web3dart`, `http`, `flutter\_dotenv`, (optional) `walletconnect\_dart` or `wallet\_connect\_v2` for WalletConnect support.  
10. (Optional) IPFS pinning service for storing token metadata (e.g., Pinata, nft.storage) — free tiers exist for small demos.

## 3) Solidity Contracts (complete)

Below are the three full Solidity contracts you can copy into Remix. All code blocks are highlighted and each contract includes comments explaining each part. Copy-paste each `.sol` file into Remix and compile with Solidity ^0.8.19 (or exact 0.8.19).

## GuideRegistry.sol (code)

// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.19;  
  
import "@openzeppelin/contracts/access/Ownable.sol";  
  
/\*\*  
 \* @title GuideRegistry  
 \* @dev Register guides (their wallet addresses) and allow contract owner (admin) to verify them.  
 \* Guides register themselves by calling registerGuide() from their wallet.  
 \* Admin (owner) calls verifyGuide() to mark them as verified.  
 \*/  
contract GuideRegistry is Ownable {  
 struct Guide {  
 address wallet;  
 string name;  
 string nationalId; // can be IPFS hash or encrypted reference in prod  
 string metadataURI; // off-chain profile (IPFS/HTTPS)  
 bool verified;  
 }  
  
 uint256 public guideCount;  
 mapping(address => uint256) public guideIndex;  
 mapping(uint256 => Guide) public guides;  
  
 event GuideRegistered(uint256 indexed id, address wallet, string name);  
 event GuideVerified(uint256 indexed id, bool verified);  
  
 /\*\*   
 \* @dev Guides call this to create their profile on-chain.  
 \*/  
 function registerGuide(string calldata name, string calldata nationalId, string calldata metadataURI) external {  
 require(guideIndex[msg.sender] == 0, "already registered");  
 guideCount++;  
 guideIndex[msg.sender] = guideCount;  
 guides[guideCount] = Guide({ wallet: msg.sender, name: name, nationalId: nationalId, metadataURI: metadataURI, verified: false });  
 emit GuideRegistered(guideCount, msg.sender, name);  
 }  
  
 /\*\*  
 \* @dev Owner verifies a guide after offline checks.  
 \*/  
 function verifyGuide(address guideWallet, bool verified) external onlyOwner {  
 uint256 idx = guideIndex[guideWallet];  
 require(idx != 0, "guide not found");  
 guides[idx].verified = verified;  
 emit GuideVerified(idx, verified);  
 }  
  
 /\*\*  
 \* @dev Check if a given guide wallet is verified.  
 \*/  
 function isGuideVerified(address guideWallet) external view returns (bool) {  
 uint256 idx = guideIndex[guideWallet];  
 if (idx == 0) return false;  
 return guides[idx].verified;  
 }  
  
 /\*\*  
 \* @dev Returns full guide struct by wallet.  
 \*/  
 function getGuideByAddress(address guideWallet) external view returns (Guide memory) {  
 uint256 idx = guideIndex[guideWallet];  
 require(idx != 0, "guide not found");  
 return guides[idx];  
 }  
}

## HandicraftNFT.sol (code)

// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.19;  
  
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";  
import "@openzeppelin/contracts/access/Ownable.sol";  
import "@openzeppelin/contracts/utils/Counters.sol";  
  
/\*\*  
 \* @title HandicraftNFT  
 \* @dev Mint ERC-721 tokens for authenticated handicrafts. Admin mints after offline verification.  
 \*/  
contract HandicraftNFT is ERC721URIStorage, Ownable {  
 using Counters for Counters.Counter;  
 Counters.Counter private \_tokenIds;  
  
 event HandicraftRegistered(uint256 tokenId, address owner, string tokenURI);  
  
 constructor() ERC721("PravassHandicraft", "PRVHC") {}  
  
 /\*\*  
 \* @dev Owner-only mint function. `tokenURI` should point to IPFS/HTTPS metadata JSON describing the craft.  
 \*/  
 function mintAuthenticated(address to, string memory tokenURI) external onlyOwner returns (uint256) {  
 \_tokenIds.increment();  
 uint256 newId = \_tokenIds.current();  
 \_safeMint(to, newId);  
 \_setTokenURI(newId, tokenURI);  
 emit HandicraftRegistered(newId, to, tokenURI);  
 return newId;  
 }  
}

## Escrow.sol (code)

// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.19;  
  
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";  
import "@openzeppelin/contracts/access/Ownable.sol";  
  
/\*\*  
 \* @title Escrow  
 \* @dev Simple booking escrow: tourist deposits ETH -> guide marks complete -> tourist releases funds.  
 \* Owner can refund in emergencies (for demo only); in production use a proper dispute resolution.  
 \*/  
contract Escrow is ReentrancyGuard, Ownable {  
 struct Booking {  
 address tourist;  
 address guide;  
 uint256 amountWei;  
 bool completedByGuide;  
 bool released;  
 }  
  
 uint256 public bookingCount;  
 mapping(uint256 => Booking) public bookings;  
  
 event BookingCreated(uint256 indexed bookingId, address tourist, address guide, uint256 amount);  
 event MarkedCompleted(uint256 indexed bookingId, address guide);  
 event Released(uint256 indexed bookingId, address guide, uint256 amount);  
 event Refunded(uint256 indexed bookingId, address tourist, uint256 amount);  
  
 // Tourist creates booking and sends ETH  
 function createBooking(address guide) external payable returns (uint256) {  
 require(msg.value > 0, "deposit required");  
 bookingCount++;  
 bookings[bookingCount] = Booking({ tourist: msg.sender, guide: guide, amountWei: msg.value, completedByGuide: false, released: false });  
 emit BookingCreated(bookingCount, msg.sender, guide, msg.value);  
 return bookingCount;  
 }  
  
 // Guide marks as completed  
 function markCompleted(uint256 bookingId) external {  
 Booking storage b = bookings[bookingId];  
 require(b.guide == msg.sender, "only guide");  
 require(!b.completedByGuide, "already marked");  
 b.completedByGuide = true;  
 emit MarkedCompleted(bookingId, msg.sender);  
 }  
  
 // Tourist confirms and releases funds  
 function releaseToGuide(uint256 bookingId) external nonReentrant {  
 Booking storage b = bookings[bookingId];  
 require(msg.sender == b.tourist, "only tourist can confirm");  
 require(b.completedByGuide, "guide hasn't marked completed");  
 require(!b.released, "already released");  
 b.released = true;  
 (bool ok, ) = b.guide.call{value: b.amountWei}("");  
 require(ok, "transfer failed");  
 emit Released(bookingId, b.guide, b.amountWei);  
 }  
  
 // Admin refund (demo only)  
 function refund(uint256 bookingId) external onlyOwner nonReentrant {  
 Booking storage b = bookings[bookingId];  
 require(!b.released, "already released");  
 b.released = true;  
 (bool ok, ) = b.tourist.call{value: b.amountWei}("");  
 require(ok, "refund failed");  
 emit Refunded(bookingId, b.tourist, b.amountWei);  
 }  
}

## 4) Deploying Contracts on Sepolia using Remix + MetaMask (step-by-step)

Step-by-step deployment (Remix + MetaMask):  
1. Open https://remix.ethereum.org in your browser.  
2. In File Explorer, create three files: GuideRegistry.sol, HandicraftNFT.sol, Escrow.sol and paste the respective code.  
3. In the 'Solidity Compiler' tab, choose compiler version 0.8.19 and compile each contract. If you import OpenZeppelin, use a specific release import path, e.g., `https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.9.3/...` to avoid version resolution issues.  
4. In 'Deploy & Run Transactions' select 'Injected Provider - MetaMask' as environment. MetaMask should already be connected and switched to Sepolia network.  
5. Fund your MetaMask Sepolia account using a Sepolia faucet (search for 'Sepolia faucet' and paste your wallet address).  
6. Deploy each contract—click Deploy; MetaMask pops up asking you to confirm gas fee (test ETH). Wait for transaction mining and copy the deployed contract address displayed by Remix.  
7. For each contract note: Deployer (owner) address — this account can call owner-only functions (like verifyGuide or mintAuthenticated). Save the addresses for the Flutter app `.env`.  
Notes:  
- If Remix shows errors importing OpenZeppelin, ensure the import path references a tagged release (e.g., /blob/v4.9.3/...).  
- You can also deploy using Hardhat or Remixd if you prefer local dev environments.

## 5) Getting Sepolia test ETH & MetaMask setup

MetaMask setup:  
- Install MetaMask extension and create/import a wallet.  
- In Settings -> Advanced ensure 'Show test networks' is enabled. Switch network to Sepolia.  
- Copy the account address and request Sepolia test ETH from a faucet (there are multiple community faucets).

Security note: Never use your mainnet wallet or real funds for testing. Use separate test accounts and private keys for demo/testing.

## 6) Flutter Integration (web3dart) — architecture & choices

Two common integration choices:  
A) Private-key demo (fast): the Flutter app holds a test private key (in .env) to sign transactions using web3dart. This is insecure for production but expedient for demos.  
B) WalletConnect (recommended for real users): the app initiates a WalletConnect session and the user's mobile wallet (MetaMask, Trust) signs transactions — keys remain with the user.  
We provide full code for option (A) and notes for switching to WalletConnect (B).

## 7) Flutter Project File Structure

A recommended clean project structure:

pravass\_blockchain\_demo/  
├─ android/  
├─ ios/  
├─ lib/  
│ ├─ main.dart  
│ ├─ screens/  
│ │ ├─ home\_screen.dart  
│ │ ├─ guide\_screen.dart  
│ │ ├─ handicraft\_screen.dart  
│ │ └─ payment\_screen.dart  
│ ├─ services/  
│ │ └─ blockchain\_service.dart  
│ ├─ models/  
│ │ └─ guide.dart  
│ └─ abi/  
│ ├─ GuideRegistry.abi.json  
│ ├─ HandicraftNFT.abi.json  
│ └─ Escrow.abi.json  
├─ pubspec.yaml  
└─ .env (not checked into git)

## 8) Full Flutter Code (main files)

Below are full files (copy into /lib). All code examples assume you placed ABI JSON into `assets/abi/` and created a `.env` with your Sepolia RPC, PRIVATE\_KEY (demo), and the deployed contract addresses.  
  
`.env` example (do NOT commit):  
SEPOLIA\_RPC=https://sepolia.infura.io/v3/YOUR\_INFURA\_PROJECT\_ID  
PRIVATE\_KEY=0xYOUR\_TEST\_PRIVATE\_KEY  
GUIDE\_REGISTRY\_ADDR=0x...  
HANDICRAFT\_NFT\_ADDR=0x...  
ESCROW\_ADDR=0x...  
CHAIN\_ID=11155111

## blockchain\_service.dart (code)

import 'dart:convert';  
import 'package:http/http.dart';  
import 'package:web3dart/web3dart.dart';  
import 'package:flutter\_dotenv/flutter\_dotenv.dart';  
import 'package:flutter/services.dart' show rootBundle;  
  
class BlockchainService {  
 final Web3Client client;  
 final Credentials credentials;  
 final EthereumAddress myAddress;  
  
 late DeployedContract guideContract;  
 late DeployedContract nftContract;  
 late DeployedContract escrowContract;  
  
 BlockchainService.\_(this.client, this.credentials, this.myAddress);  
  
 static Future<BlockchainService> create() async {  
 await dotenv.load();  
  
 final rpcUrl = dotenv.env['SEPOLIA\_RPC']!;  
 final privateKey = dotenv.env['PRIVATE\_KEY']!;  
 final httpClient = Client();  
 final client = Web3Client(rpcUrl, httpClient);  
  
 final credentials = EthPrivateKey.fromHex(privateKey);  
 final myAddress = await credentials.extractAddress();  
  
 final instance = BlockchainService.\_(client, credentials, myAddress);  
  
 final guideAbi = await rootBundle.loadString('assets/abi/GuideRegistry.abi.json');  
 final nftAbi = await rootBundle.loadString('assets/abi/HandicraftNFT.abi.json');  
 final escrowAbi = await rootBundle.loadString('assets/abi/Escrow.abi.json');  
  
 final guideAddr = EthereumAddress.fromHex(dotenv.env['GUIDE\_REGISTRY\_ADDR']!);  
 final nftAddr = EthereumAddress.fromHex(dotenv.env['HANDICRAFT\_NFT\_ADDR']!);  
 final escrowAddr = EthereumAddress.fromHex(dotenv.env['ESCROW\_ADDR']!);  
  
 instance.guideContract = DeployedContract(ContractAbi.fromJson(guideAbi, "GuideRegistry"), guideAddr);  
 instance.nftContract = DeployedContract(ContractAbi.fromJson(nftAbi, "HandicraftNFT"), nftAddr);  
 instance.escrowContract = DeployedContract(ContractAbi.fromJson(escrowAbi, "Escrow"), escrowAddr);  
  
 return instance;  
 }  
  
 Future<bool> isGuideVerified(EthereumAddress guideWallet) async {  
 final fn = guideContract.function('isGuideVerified');  
 final res = await client.call(contract: guideContract, function: fn, params: [guideWallet]);  
 return res.first as bool;  
 }  
  
 Future<String> registerGuide(String name, String nationalId, String metadataURI) async {  
 final fn = guideContract.function('registerGuide');  
 final tx = Transaction.callContract(  
 contract: guideContract,  
 function: fn,  
 parameters: [name, nationalId, metadataURI],  
 from: myAddress,  
 );  
 return client.sendTransaction(credentials, tx, chainId: int.parse(dotenv.env['CHAIN\_ID'] ?? '11155111'));  
 }  
  
 Future<String> verifyGuide(EthereumAddress guideWallet, bool verified) async {  
 final fn = guideContract.function('verifyGuide');  
 final tx = Transaction.callContract(contract: guideContract, function: fn, parameters: [guideWallet, verified], from: myAddress);  
 return client.sendTransaction(credentials, tx, chainId: int.parse(dotenv.env['CHAIN\_ID'] ?? '11155111'));  
 }  
  
 Future<String> mintHandicraft(EthereumAddress to, String tokenURI) async {  
 final fn = nftContract.function('mintAuthenticated');  
 final tx = Transaction.callContract(contract: nftContract, function: fn, parameters: [to, tokenURI], from: myAddress);  
 return client.sendTransaction(credentials, tx, chainId: int.parse(dotenv.env['CHAIN\_ID'] ?? '11155111'));  
 }  
  
 Future<String> createBooking(EthereumAddress guideAddr, EtherAmount amount) async {  
 final fn = escrowContract.function('createBooking');  
 final tx = Transaction.callContract(  
 contract: escrowContract,  
 function: fn,  
 parameters: [guideAddr],  
 value: amount,  
 from: myAddress,  
 );  
 return client.sendTransaction(credentials, tx, chainId: int.parse(dotenv.env['CHAIN\_ID'] ?? '11155111'));  
 }  
  
 Future<String> markCompleted(BigInt bookingId) async {  
 final fn = escrowContract.function('markCompleted');  
 final tx = Transaction.callContract(contract: escrowContract, function: fn, parameters: [bookingId], from: myAddress);  
 return client.sendTransaction(credentials, tx, chainId: int.parse(dotenv.env['CHAIN\_ID'] ?? '11155111'));  
 }  
  
 Future<String> releaseToGuide(BigInt bookingId) async {  
 final fn = escrowContract.function('releaseToGuide');  
 final tx = Transaction.callContract(contract: escrowContract, function: fn, parameters: [bookingId], from: myAddress);  
 return client.sendTransaction(credentials, tx, chainId: int.parse(dotenv.env['CHAIN\_ID'] ?? '11155111'));  
 }  
}

## main.dart (code)

import 'package:flutter/material.dart';  
import 'screens/home\_screen.dart';  
  
void main() async {  
 WidgetsFlutterBinding.ensureInitialized();  
 runApp(const PravassApp());  
}  
  
class PravassApp extends StatelessWidget {  
 const PravassApp({Key? key}) : super(key: key);  
  
 @override  
 Widget build(BuildContext context) {  
 return MaterialApp(  
 title: 'Pravass Demo',  
 theme: ThemeData(primarySwatch: Colors.teal),  
 home: const HomeScreen(),  
 );  
 }  
}

## home\_screen.dart (code)

import 'package:flutter/material.dart';  
import 'guide\_screen.dart';  
import 'handicraft\_screen.dart';  
import 'payment\_screen.dart';  
  
class HomeScreen extends StatelessWidget {  
 const HomeScreen({Key? key}) : super(key: key);  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(title: const Text('Pravass Demo')),  
 body: Center(  
 child: Column(mainAxisSize: MainAxisSize.min, children: [  
 ElevatedButton(onPressed: () => Navigator.push(context, MaterialPageRoute(builder: (\_) => GuideScreen())), child: const Text('Guide Verification')),  
 ElevatedButton(onPressed: () => Navigator.push(context, MaterialPageRoute(builder: (\_) => HandicraftScreen())), child: const Text('Handicraft (NFT)')),  
 ElevatedButton(onPressed: () => Navigator.push(context, MaterialPageRoute(builder: (\_) => PaymentScreen())), child: const Text('Secure Payment (Escrow)')),  
 ]),  
 ),  
 );  
 }  
}

## guide\_screen.dart (code)

import 'package:flutter/material.dart';  
import '../services/blockchain\_service.dart';  
import 'package:web3dart/web3dart.dart';  
  
class GuideScreen extends StatefulWidget {  
 const GuideScreen({Key? key}) : super(key: key);  
  
 @override  
 State<GuideScreen> createState() => \_GuideScreenState();  
}  
  
class \_GuideScreenState extends State<GuideScreen> {  
 final \_nameCtrl = TextEditingController();  
 final \_idCtrl = TextEditingController();  
 final \_metaCtrl = TextEditingController();  
 final \_checkCtrl = TextEditingController();  
  
 String \_status = '';  
 bool \_loading = false;  
  
 void \_show(String s) => setState(() => \_status = s);  
  
 Future<void> \_register() async {  
 setState(() => \_loading = true);  
 final svc = await BlockchainService.create();  
 try {  
 final tx = await svc.registerGuide(\_nameCtrl.text, \_idCtrl.text, \_metaCtrl.text);  
 \_show('Register tx sent: ' + tx);  
 } catch (e) {  
 \_show('Error: $e');  
 }  
 setState(() => \_loading = false);  
 }  
  
 Future<void> \_check() async {  
 setState(() => \_loading = true);  
 final svc = await BlockchainService.create();  
 try {  
 final addr = EthereumAddress.fromHex(\_checkCtrl.text);  
 final ok = await svc.isGuideVerified(addr);  
 \_show('Verified: $ok');  
 } catch (e) {  
 \_show('Error: $e');  
 }  
 setState(() => \_loading = false);  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(title: const Text('Guide Verification')),  
 body: Padding(  
 padding: const EdgeInsets.all(12.0),  
 child: SingleChildScrollView(  
 child: Column(children: [  
 TextField(controller: \_nameCtrl, decoration: const InputDecoration(labelText: 'Guide Name')),  
 TextField(controller: \_idCtrl, decoration: const InputDecoration(labelText: 'National ID (or reference)')),  
 TextField(controller: \_metaCtrl, decoration: const InputDecoration(labelText: 'Metadata URI (IPFS/HTTP)')),  
 const SizedBox(height: 8),  
 ElevatedButton(onPressed: \_register, child: \_loading ? const CircularProgressIndicator() : const Text('Register Guide')),  
 const Divider(),  
 TextField(controller: \_checkCtrl, decoration: const InputDecoration(labelText: 'Guide Wallet to Check (0x...)')),  
 ElevatedButton(onPressed: \_check, child: const Text('Check Verified')),  
 const SizedBox(height: 12),  
 Text(\_status),  
 ]),  
 ),  
 ),  
 );  
 }  
}

## handicraft\_screen.dart (code)

import 'package:flutter/material.dart';  
import '../services/blockchain\_service.dart';  
import 'package:web3dart/web3dart.dart';  
  
class HandicraftScreen extends StatefulWidget {  
 const HandicraftScreen({Key? key}) : super(key: key);  
  
 @override  
 State<HandicraftScreen> createState() => \_HandicraftScreenState();  
}  
  
class \_HandicraftScreenState extends State<HandicraftScreen> {  
 final \_addrCtrl = TextEditingController();  
 final \_uriCtrl = TextEditingController();  
 String \_status = '';  
  
 void \_show(String s) => setState(() => \_status = s);  
  
 Future<void> \_mint() async {  
 final svc = await BlockchainService.create();  
 try {  
 final to = EthereumAddress.fromHex(\_addrCtrl.text);  
 final tx = await svc.mintHandicraft(to, \_uriCtrl.text);  
 \_show('Mint tx: ' + tx);  
 } catch (e) {  
 \_show('Error: $e');  
 }  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(title: const Text('Handicraft NFT')),  
 body: Padding(  
 padding: const EdgeInsets.all(12.0),  
 child: Column(children: [  
 TextField(controller: \_addrCtrl, decoration: const InputDecoration(labelText: 'Artisan Wallet (0x...)')),  
 TextField(controller: \_uriCtrl, decoration: const InputDecoration(labelText: 'Token Metadata URI (IPFS/HTTP)')),  
 const SizedBox(height: 8),  
 ElevatedButton(onPressed: \_mint, child: const Text('Mint Authenticated NFT')),  
 const SizedBox(height: 12),  
 Text(\_status),  
 ]),  
 ),  
 );  
 }  
}

## payment\_screen.dart (code)

import 'package:flutter/material.dart';  
import '../services/blockchain\_service.dart';  
import 'package:web3dart/web3dart.dart';  
  
class PaymentScreen extends StatefulWidget {  
 const PaymentScreen({Key? key}) : super(key: key);  
  
 @override  
 State<PaymentScreen> createState() => \_PaymentScreenState();  
}  
  
class \_PaymentScreenState extends State<PaymentScreen> {  
 final \_guideCtrl = TextEditingController();  
 final \_amountCtrl = TextEditingController();  
 final \_bookingCtrl = TextEditingController();  
 String \_status = '';  
  
 void \_show(String s) => setState(() => \_status = s);  
  
 Future<void> \_createBooking() async {  
 final svc = await BlockchainService.create();  
 try {  
 final guide = EthereumAddress.fromHex(\_guideCtrl.text);  
 final amountEth = double.parse(\_amountCtrl.text);  
 final amountWei = EtherAmount.fromUnitAndValue(EtherUnit.ether, amountEth);  
 final tx = await svc.createBooking(guide, amountWei);  
 \_show('Booking tx: ' + tx);  
 } catch (e) {  
 \_show('Error: $e');  
 }  
 }  
  
 Future<void> \_markCompleted() async {  
 final svc = await BlockchainService.create();  
 try {  
 final bookingId = BigInt.parse(\_bookingCtrl.text);  
 final tx = await svc.markCompleted(bookingId);  
 \_show('Marked complete tx: ' + tx);  
 } catch (e) {  
 \_show('Error: $e');  
 }  
 }  
  
 Future<void> \_release() async {  
 final svc = await BlockchainService.create();  
 try {  
 final bookingId = BigInt.parse(\_bookingCtrl.text);  
 final tx = await svc.releaseToGuide(bookingId);  
 \_show('Release tx: ' + tx);  
 } catch (e) {  
 \_show('Error: $e');  
 }  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(title: const Text('Escrow Payment')),  
 body: Padding(  
 padding: const EdgeInsets.all(12.0),  
 child: SingleChildScrollView(  
 child: Column(children: [  
 TextField(controller: \_guideCtrl, decoration: const InputDecoration(labelText: 'Guide Wallet (0x...)')),  
 TextField(controller: \_amountCtrl, decoration: const InputDecoration(labelText: 'Amount (ETH)'), keyboardType: TextInputType.number),  
 const SizedBox(height: 8),  
 ElevatedButton(onPressed: \_createBooking, child: const Text('Create Booking (deposit)')),  
 const Divider(),  
 TextField(controller: \_bookingCtrl, decoration: const InputDecoration(labelText: 'Booking ID (number)')),  
 ElevatedButton(onPressed: \_markCompleted, child: const Text('Guide: Mark Completed')),  
 ElevatedButton(onPressed: \_release, child: const Text('Tourist: Release to Guide')),  
 const SizedBox(height: 12),  
 Text(\_status),  
 ]),  
 ),  
 ),  
 );  
 }  
}

## 9) Example ABI JSON & usage

After compiling a contract in Remix, copy the ABI JSON (Contract → Compilation details → ABI) into a file `assets/abi/ContractName.abi.json`. Example minimal ABI for GuideRegistry (only the isGuideVerified function):  
[  
 {  
 "inputs":[{"internalType":"address","name":"guideWallet","type":"address"}],  
 "name":"isGuideVerified",  
 "outputs":[{"internalType":"bool","name":"","type":"bool"}],  
 "stateMutability":"view",  
 "type":"function"  
 }  
]  
Usage in Flutter (snippet):  
String abi = await rootBundle.loadString('assets/abi/GuideRegistry.abi.json');  
final contract = DeployedContract(ContractAbi.fromJson(abi, "GuideRegistry"), EthereumAddress.fromHex(addressHex));  
final fn = contract.function('isGuideVerified');  
final result = await client.call(contract: contract, function: fn, params: [EthereumAddress.fromHex(myAddress)]);

## 10) Flowcharts / Diagrams (text)

Mermaid flow (paste to mermaid.live or compatible viewer):  
```mermaid  
flowchart LR  
 TouristApp[Pravass Flutter App] -->|reads| GuideRegistry[(GuideRegistry contract)]  
 TouristApp -->|reads| HandicraftNFT[(HandicraftNFT contract)]  
 TouristApp -->|creates booking + pays| Escrow[(Escrow contract)]  
 Guide -->|marks complete| Escrow  
 Tourist -->|confirms| Escrow  
 Escrow -->|releases funds| Guide  
```

ASCII flow:  
- Tourist App -> Query GuideRegistry.isGuideVerified(address)  
- Tourist App -> Query HandicraftNFT.tokenURI(tokenId)  
- Tourist App -> createBooking(guide) + send ETH -> Escrow holds funds  
- Guide -> markCompleted(bookingId)  
- Tourist -> releaseToGuide(bookingId) -> Escrow sends ETH to guide

## 11) Where & How to Run (end-to-end)

Step-by-step run checklist:  
1. Install Flutter & run `flutter doctor` and set up emulator or connect a real device.  
2. Create a new Flutter project and replace files with the ones provided in this doc. Add ABI JSON into assets/abi/ and update pubspec.yaml with assets.  
3. Create a `.env` with SEPOLIA\_RPC, PRIVATE\_KEY (demo/test key), contract addresses (GUIDE\_REGISTRY\_ADDR, HANDICRAFT\_NFT\_ADDR, ESCROW\_ADDR), CHAIN\_ID=11155111.  
4. Deploy contracts via Remix and MetaMask on Sepolia; copy addresses to .env.  
5. `flutter pub get`  
6. `flutter run` on emulator/device. Use forms to register guide, mint handicraft, create bookings and run escrow flows.  
7. Check transactions on Sepolia Etherscan (use deployed addresses or tx hashes).  
Notes:  
- For WalletConnect flows, use MetaMask Mobile and implement WC code instead of private-key signing.  
- If you get RPC rate limits, rotate Infura/Alchemy keys or use different provider.

## 12) Step-by-step explanation (from scratch)

A more finely grained step sequence to create a working mini-demo:  
1) Get developer tools installed (Flutter, VSCode, MetaMask).  
2) Create MetaMask Sepolia account and get test ETH from faucet.  
3) Open Remix and paste GuideRegistry.sol; compile; deploy using Injected Web3 (MetaMask). Repeat for HandicraftNFT and Escrow.  
4) Copy contract ABIs into assets/abi/ and deployed addresses into `.env`.  
5) Create Flutter app, add dependencies and assets in pubspec.yaml.  
6) Implement BlockchainService (shown above) to connect to the RPC and sign transactions.  
7) Implement simple UI screens (Home -> Guide/Handicraft/Payment) to interact with contract functions.  
8) Run the app; perform demo flows and verify on Sepolia Etherscan.  
9) Swap the private-key approach with WalletConnect for real users.

## 13) Security Considerations (detailed)

Critical security notes BEFORE you go to production:  
- NEVER store private keys or mnemonics in app code or repository.  
- Use WalletConnect or redirect to user wallets so keys never leave the user's device.  
- Admin keys: for owner-only functions (e.g., verifyGuide or mintAuthenticated) use a multisig wallet (Gnosis Safe) or hardware wallet; never keep owner key in a hot wallet.  
- Reentrancy & safety: use OpenZeppelin contracts and `ReentrancyGuard` (used in Escrow). Consider withdraw-patterns for funds rather than pushing funds directly in complex flows.  
- Validate all inputs (strings, addresses) and limit gas usage in loops. Prefer mappings to arrays for storage efficiency.  
- Metadata immutability: upload token metadata to IPFS and reference immutable CID in tokenURI. Pin files using a pinning service if necessary.  
- Audit smart contracts before deploying to mainnet. Use testnets extensively.

## 14) Troubleshooting & Common Errors

Examples:  
- 'Gas estimation failed': ensure the account has test ETH and the contract ABI/method params match.  
- 'Transaction pending' or 'nonce too low': check pending transactions, use correct chainId, and ensure RPC provider is healthy.  
- 'Import error' in Remix for OpenZeppelin: use exact GitHub tag release and correct import paths.  
- 'Web3Client connection issues': ensure SEPOLIA\_RPC is set correctly and provider allows Sepolia RPC.

## 15) Extras & Recommendations

- Use IPFS (nft.storage or Pinata) for token metadata/photos.  
- Consider adding an off-chain verification backend for additional checks (e.g., KYC) and only finalize verification on-chain after offline validation.  
- For full production flows: add a dispute resolution mechanism, timeouts, and penalty rules in Escrow contract.  
- To save gas, store minimal data on-chain and more details off-chain with signed references.

## Appendix A — Example minimal ABI files

Example: GuideRegistry.abi.json (partial, minimal):  
[  
 {  
 "inputs":[{"internalType":"address","name":"guideWallet","type":"address"}],  
 "name":"isGuideVerified",  
 "outputs":[{"internalType":"bool","name":"","type":"bool"}],  
 "stateMutability":"view",  
 "type":"function"  
 },  
 {  
 "inputs":[{"internalType":"string","name":"name","type":"string"},{"internalType":"string","name":"nationalId","type":"string"},{"internalType":"string","name":"metadataURI","type":"string"}],  
 "name":"registerGuide",  
 "outputs":[],  
 "stateMutability":"nonpayable",  
 "type":"function"  
 }  
]

## Appendix - Full code files

### GuideRegistry.sol

// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.19;  
  
import "@openzeppelin/contracts/access/Ownable.sol";  
  
/\*\*  
 \* @title GuideRegistry  
 \* @dev Register guides (their wallet addresses) and allow contract owner (admin) to verify them.  
 \* Guides register themselves by calling registerGuide() from their wallet.  
 \* Admin (owner) calls verifyGuide() to mark them as verified.  
 \*/  
contract GuideRegistry is Ownable {  
 struct Guide {  
 address wallet;  
 string name;  
 string nationalId; // can be IPFS hash or encrypted reference in prod  
 string metadataURI; // off-chain profile (IPFS/HTTPS)  
 bool verified;  
 }  
  
 uint256 public guideCount;  
 mapping(address => uint256) public guideIndex;  
 mapping(uint256 => Guide) public guides;  
  
 event GuideRegistered(uint256 indexed id, address wallet, string name);  
 event GuideVerified(uint256 indexed id, bool verified);  
  
 /\*\*   
 \* @dev Guides call this to create their profile on-chain.  
 \*/  
 function registerGuide(string calldata name, string calldata nationalId, string calldata metadataURI) external {  
 require(guideIndex[msg.sender] == 0, "already registered");  
 guideCount++;  
 guideIndex[msg.sender] = guideCount;  
 guides[guideCount] = Guide({ wallet: msg.sender, name: name, nationalId: nationalId, metadataURI: metadataURI, verified: false });  
 emit GuideRegistered(guideCount, msg.sender, name);  
 }  
  
 /\*\*  
 \* @dev Owner verifies a guide after offline checks.  
 \*/  
 function verifyGuide(address guideWallet, bool verified) external onlyOwner {  
 uint256 idx = guideIndex[guideWallet];  
 require(idx != 0, "guide not found");  
 guides[idx].verified = verified;  
 emit GuideVerified(idx, verified);  
 }  
  
 /\*\*  
 \* @dev Check if a given guide wallet is verified.  
 \*/  
 function isGuideVerified(address guideWallet) external view returns (bool) {  
 uint256 idx = guideIndex[guideWallet];  
 if (idx == 0) return false;  
 return guides[idx].verified;  
 }  
  
 /\*\*  
 \* @dev Returns full guide struct by wallet.  
 \*/  
 function getGuideByAddress(address guideWallet) external view returns (Guide memory) {  
 uint256 idx = guideIndex[guideWallet];  
 require(idx != 0, "guide not found");  
 return guides[idx];  
 }  
}

### HandicraftNFT.sol

// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.19;  
  
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";  
import "@openzeppelin/contracts/access/Ownable.sol";  
import "@openzeppelin/contracts/utils/Counters.sol";  
  
/\*\*  
 \* @title HandicraftNFT  
 \* @dev Mint ERC-721 tokens for authenticated handicrafts. Admin mints after offline verification.  
 \*/  
contract HandicraftNFT is ERC721URIStorage, Ownable {  
 using Counters for Counters.Counter;  
 Counters.Counter private \_tokenIds;  
  
 event HandicraftRegistered(uint256 tokenId, address owner, string tokenURI);  
  
 constructor() ERC721("PravassHandicraft", "PRVHC") {}  
  
 /\*\*  
 \* @dev Owner-only mint function. `tokenURI` should point to IPFS/HTTPS metadata JSON describing the craft.  
 \*/  
 function mintAuthenticated(address to, string memory tokenURI) external onlyOwner returns (uint256) {  
 \_tokenIds.increment();  
 uint256 newId = \_tokenIds.current();  
 \_safeMint(to, newId);  
 \_setTokenURI(newId, tokenURI);  
 emit HandicraftRegistered(newId, to, tokenURI);  
 return newId;  
 }  
}

### Escrow.sol

// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.19;  
  
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";  
import "@openzeppelin/contracts/access/Ownable.sol";  
  
/\*\*  
 \* @title Escrow  
 \* @dev Simple booking escrow: tourist deposits ETH -> guide marks complete -> tourist releases funds.  
 \* Owner can refund in emergencies (for demo only); in production use a proper dispute resolution.  
 \*/  
contract Escrow is ReentrancyGuard, Ownable {  
 struct Booking {  
 address tourist;  
 address guide;  
 uint256 amountWei;  
 bool completedByGuide;  
 bool released;  
 }  
  
 uint256 public bookingCount;  
 mapping(uint256 => Booking) public bookings;  
  
 event BookingCreated(uint256 indexed bookingId, address tourist, address guide, uint256 amount);  
 event MarkedCompleted(uint256 indexed bookingId, address guide);  
 event Released(uint256 indexed bookingId, address guide, uint256 amount);  
 event Refunded(uint256 indexed bookingId, address tourist, uint256 amount);  
  
 // Tourist creates booking and sends ETH  
 function createBooking(address guide) external payable returns (uint256) {  
 require(msg.value > 0, "deposit required");  
 bookingCount++;  
 bookings[bookingCount] = Booking({ tourist: msg.sender, guide: guide, amountWei: msg.value, completedByGuide: false, released: false });  
 emit BookingCreated(bookingCount, msg.sender, guide, msg.value);  
 return bookingCount;  
 }  
  
 // Guide marks as completed  
 function markCompleted(uint256 bookingId) external {  
 Booking storage b = bookings[bookingId];  
 require(b.guide == msg.sender, "only guide");  
 require(!b.completedByGuide, "already marked");  
 b.completedByGuide = true;  
 emit MarkedCompleted(bookingId, msg.sender);  
 }  
  
 // Tourist confirms and releases funds  
 function releaseToGuide(uint256 bookingId) external nonReentrant {  
 Booking storage b = bookings[bookingId];  
 require(msg.sender == b.tourist, "only tourist can confirm");  
 require(b.completedByGuide, "guide hasn't marked completed");  
 require(!b.released, "already released");  
 b.released = true;  
 (bool ok, ) = b.guide.call{value: b.amountWei}("");  
 require(ok, "transfer failed");  
 emit Released(bookingId, b.guide, b.amountWei);  
 }  
  
 // Admin refund (demo only)  
 function refund(uint256 bookingId) external onlyOwner nonReentrant {  
 Booking storage b = bookings[bookingId];  
 require(!b.released, "already released");  
 b.released = true;  
 (bool ok, ) = b.tourist.call{value: b.amountWei}("");  
 require(ok, "refund failed");  
 emit Refunded(bookingId, b.tourist, b.amountWei);  
 }  
}

### blockchain\_service.dart

import 'dart:convert';  
import 'package:http/http.dart';  
import 'package:web3dart/web3dart.dart';  
import 'package:flutter\_dotenv/flutter\_dotenv.dart';  
import 'package:flutter/services.dart' show rootBundle;  
  
class BlockchainService {  
 final Web3Client client;  
 final Credentials credentials;  
 final EthereumAddress myAddress;  
  
 late DeployedContract guideContract;  
 late DeployedContract nftContract;  
 late DeployedContract escrowContract;  
  
 BlockchainService.\_(this.client, this.credentials, this.myAddress);  
  
 static Future<BlockchainService> create() async {  
 await dotenv.load();  
  
 final rpcUrl = dotenv.env['SEPOLIA\_RPC']!;  
 final privateKey = dotenv.env['PRIVATE\_KEY']!;  
 final httpClient = Client();  
 final client = Web3Client(rpcUrl, httpClient);  
  
 final credentials = EthPrivateKey.fromHex(privateKey);  
 final myAddress = await credentials.extractAddress();  
  
 final instance = BlockchainService.\_(client, credentials, myAddress);  
  
 final guideAbi = await rootBundle.loadString('assets/abi/GuideRegistry.abi.json');  
 final nftAbi = await rootBundle.loadString('assets/abi/HandicraftNFT.abi.json');  
 final escrowAbi = await rootBundle.loadString('assets/abi/Escrow.abi.json');  
  
 final guideAddr = EthereumAddress.fromHex(dotenv.env['GUIDE\_REGISTRY\_ADDR']!);  
 final nftAddr = EthereumAddress.fromHex(dotenv.env['HANDICRAFT\_NFT\_ADDR']!);  
 final escrowAddr = EthereumAddress.fromHex(dotenv.env['ESCROW\_ADDR']!);  
  
 instance.guideContract = DeployedContract(ContractAbi.fromJson(guideAbi, "GuideRegistry"), guideAddr);  
 instance.nftContract = DeployedContract(ContractAbi.fromJson(nftAbi, "HandicraftNFT"), nftAddr);  
 instance.escrowContract = DeployedContract(ContractAbi.fromJson(escrowAbi, "Escrow"), escrowAddr);  
  
 return instance;  
 }  
  
 Future<bool> isGuideVerified(EthereumAddress guideWallet) async {  
 final fn = guideContract.function('isGuideVerified');  
 final res = await client.call(contract: guideContract, function: fn, params: [guideWallet]);  
 return res.first as bool;  
 }  
  
 Future<String> registerGuide(String name, String nationalId, String metadataURI) async {  
 final fn = guideContract.function('registerGuide');  
 final tx = Transaction.callContract(  
 contract: guideContract,  
 function: fn,  
 parameters: [name, nationalId, metadataURI],  
 from: myAddress,  
 );  
 return client.sendTransaction(credentials, tx, chainId: int.parse(dotenv.env['CHAIN\_ID'] ?? '11155111'));  
 }  
  
 Future<String> verifyGuide(EthereumAddress guideWallet, bool verified) async {  
 final fn = guideContract.function('verifyGuide');  
 final tx = Transaction.callContract(contract: guideContract, function: fn, parameters: [guideWallet, verified], from: myAddress);  
 return client.sendTransaction(credentials, tx, chainId: int.parse(dotenv.env['CHAIN\_ID'] ?? '11155111'));  
 }  
  
 Future<String> mintHandicraft(EthereumAddress to, String tokenURI) async {  
 final fn = nftContract.function('mintAuthenticated');  
 final tx = Transaction.callContract(contract: nftContract, function: fn, parameters: [to, tokenURI], from: myAddress);  
 return client.sendTransaction(credentials, tx, chainId: int.parse(dotenv.env['CHAIN\_ID'] ?? '11155111'));  
 }  
  
 Future<String> createBooking(EthereumAddress guideAddr, EtherAmount amount) async {  
 final fn = escrowContract.function('createBooking');  
 final tx = Transaction.callContract(  
 contract: escrowContract,  
 function: fn,  
 parameters: [guideAddr],  
 value: amount,  
 from: myAddress,  
 );  
 return client.sendTransaction(credentials, tx, chainId: int.parse(dotenv.env['CHAIN\_ID'] ?? '11155111'));  
 }  
  
 Future<String> markCompleted(BigInt bookingId) async {  
 final fn = escrowContract.function('markCompleted');  
 final tx = Transaction.callContract(contract: escrowContract, function: fn, parameters: [bookingId], from: myAddress);  
 return client.sendTransaction(credentials, tx, chainId: int.parse(dotenv.env['CHAIN\_ID'] ?? '11155111'));  
 }  
  
 Future<String> releaseToGuide(BigInt bookingId) async {  
 final fn = escrowContract.function('releaseToGuide');  
 final tx = Transaction.callContract(contract: escrowContract, function: fn, parameters: [bookingId], from: myAddress);  
 return client.sendTransaction(credentials, tx, chainId: int.parse(dotenv.env['CHAIN\_ID'] ?? '11155111'));  
 }  
}

### main.dart

import 'package:flutter/material.dart';  
import 'screens/home\_screen.dart';  
  
void main() async {  
 WidgetsFlutterBinding.ensureInitialized();  
 runApp(const PravassApp());  
}  
  
class PravassApp extends StatelessWidget {  
 const PravassApp({Key? key}) : super(key: key);  
  
 @override  
 Widget build(BuildContext context) {  
 return MaterialApp(  
 title: 'Pravass Demo',  
 theme: ThemeData(primarySwatch: Colors.teal),  
 home: const HomeScreen(),  
 );  
 }  
}

### home\_screen.dart

import 'package:flutter/material.dart';  
import 'guide\_screen.dart';  
import 'handicraft\_screen.dart';  
import 'payment\_screen.dart';  
  
class HomeScreen extends StatelessWidget {  
 const HomeScreen({Key? key}) : super(key: key);  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(title: const Text('Pravass Demo')),  
 body: Center(  
 child: Column(mainAxisSize: MainAxisSize.min, children: [  
 ElevatedButton(onPressed: () => Navigator.push(context, MaterialPageRoute(builder: (\_) => GuideScreen())), child: const Text('Guide Verification')),  
 ElevatedButton(onPressed: () => Navigator.push(context, MaterialPageRoute(builder: (\_) => HandicraftScreen())), child: const Text('Handicraft (NFT)')),  
 ElevatedButton(onPressed: () => Navigator.push(context, MaterialPageRoute(builder: (\_) => PaymentScreen())), child: const Text('Secure Payment (Escrow)')),  
 ]),  
 ),  
 );  
 }  
}

### guide\_screen.dart

import 'package:flutter/material.dart';  
import '../services/blockchain\_service.dart';  
import 'package:web3dart/web3dart.dart';  
  
class GuideScreen extends StatefulWidget {  
 const GuideScreen({Key? key}) : super(key: key);  
  
 @override  
 State<GuideScreen> createState() => \_GuideScreenState();  
}  
  
class \_GuideScreenState extends State<GuideScreen> {  
 final \_nameCtrl = TextEditingController();  
 final \_idCtrl = TextEditingController();  
 final \_metaCtrl = TextEditingController();  
 final \_checkCtrl = TextEditingController();  
  
 String \_status = '';  
 bool \_loading = false;  
  
 void \_show(String s) => setState(() => \_status = s);  
  
 Future<void> \_register() async {  
 setState(() => \_loading = true);  
 final svc = await BlockchainService.create();  
 try {  
 final tx = await svc.registerGuide(\_nameCtrl.text, \_idCtrl.text, \_metaCtrl.text);  
 \_show('Register tx sent: ' + tx);  
 } catch (e) {  
 \_show('Error: $e');  
 }  
 setState(() => \_loading = false);  
 }  
  
 Future<void> \_check() async {  
 setState(() => \_loading = true);  
 final svc = await BlockchainService.create();  
 try {  
 final addr = EthereumAddress.fromHex(\_checkCtrl.text);  
 final ok = await svc.isGuideVerified(addr);  
 \_show('Verified: $ok');  
 } catch (e) {  
 \_show('Error: $e');  
 }  
 setState(() => \_loading = false);  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(title: const Text('Guide Verification')),  
 body: Padding(  
 padding: const EdgeInsets.all(12.0),  
 child: SingleChildScrollView(  
 child: Column(children: [  
 TextField(controller: \_nameCtrl, decoration: const InputDecoration(labelText: 'Guide Name')),  
 TextField(controller: \_idCtrl, decoration: const InputDecoration(labelText: 'National ID (or reference)')),  
 TextField(controller: \_metaCtrl, decoration: const InputDecoration(labelText: 'Metadata URI (IPFS/HTTP)')),  
 const SizedBox(height: 8),  
 ElevatedButton(onPressed: \_register, child: \_loading ? const CircularProgressIndicator() : const Text('Register Guide')),  
 const Divider(),  
 TextField(controller: \_checkCtrl, decoration: const InputDecoration(labelText: 'Guide Wallet to Check (0x...)')),  
 ElevatedButton(onPressed: \_check, child: const Text('Check Verified')),  
 const SizedBox(height: 12),  
 Text(\_status),  
 ]),  
 ),  
 ),  
 );  
 }  
}

### handicraft\_screen.dart

import 'package:flutter/material.dart';  
import '../services/blockchain\_service.dart';  
import 'package:web3dart/web3dart.dart';  
  
class HandicraftScreen extends StatefulWidget {  
 const HandicraftScreen({Key? key}) : super(key: key);  
  
 @override  
 State<HandicraftScreen> createState() => \_HandicraftScreenState();  
}  
  
class \_HandicraftScreenState extends State<HandicraftScreen> {  
 final \_addrCtrl = TextEditingController();  
 final \_uriCtrl = TextEditingController();  
 String \_status = '';  
  
 void \_show(String s) => setState(() => \_status = s);  
  
 Future<void> \_mint() async {  
 final svc = await BlockchainService.create();  
 try {  
 final to = EthereumAddress.fromHex(\_addrCtrl.text);  
 final tx = await svc.mintHandicraft(to, \_uriCtrl.text);  
 \_show('Mint tx: ' + tx);  
 } catch (e) {  
 \_show('Error: $e');  
 }  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(title: const Text('Handicraft NFT')),  
 body: Padding(  
 padding: const EdgeInsets.all(12.0),  
 child: Column(children: [  
 TextField(controller: \_addrCtrl, decoration: const InputDecoration(labelText: 'Artisan Wallet (0x...)')),  
 TextField(controller: \_uriCtrl, decoration: const InputDecoration(labelText: 'Token Metadata URI (IPFS/HTTP)')),  
 const SizedBox(height: 8),  
 ElevatedButton(onPressed: \_mint, child: const Text('Mint Authenticated NFT')),  
 const SizedBox(height: 12),  
 Text(\_status),  
 ]),  
 ),  
 );  
 }  
}

### payment\_screen.dart

import 'package:flutter/material.dart';  
import '../services/blockchain\_service.dart';  
import 'package:web3dart/web3dart.dart';  
  
class PaymentScreen extends StatefulWidget {  
 const PaymentScreen({Key? key}) : super(key: key);  
  
 @override  
 State<PaymentScreen> createState() => \_PaymentScreenState();  
}  
  
class \_PaymentScreenState extends State<PaymentScreen> {  
 final \_guideCtrl = TextEditingController();  
 final \_amountCtrl = TextEditingController();  
 final \_bookingCtrl = TextEditingController();  
 String \_status = '';  
  
 void \_show(String s) => setState(() => \_status = s);  
  
 Future<void> \_createBooking() async {  
 final svc = await BlockchainService.create();  
 try {  
 final guide = EthereumAddress.fromHex(\_guideCtrl.text);  
 final amountEth = double.parse(\_amountCtrl.text);  
 final amountWei = EtherAmount.fromUnitAndValue(EtherUnit.ether, amountEth);  
 final tx = await svc.createBooking(guide, amountWei);  
 \_show('Booking tx: ' + tx);  
 } catch (e) {  
 \_show('Error: $e');  
 }  
 }  
  
 Future<void> \_markCompleted() async {  
 final svc = await BlockchainService.create();  
 try {  
 final bookingId = BigInt.parse(\_bookingCtrl.text);  
 final tx = await svc.markCompleted(bookingId);  
 \_show('Marked complete tx: ' + tx);  
 } catch (e) {  
 \_show('Error: $e');  
 }  
 }  
  
 Future<void> \_release() async {  
 final svc = await BlockchainService.create();  
 try {  
 final bookingId = BigInt.parse(\_bookingCtrl.text);  
 final tx = await svc.releaseToGuide(bookingId);  
 \_show('Release tx: ' + tx);  
 } catch (e) {  
 \_show('Error: $e');  
 }  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(title: const Text('Escrow Payment')),  
 body: Padding(  
 padding: const EdgeInsets.all(12.0),  
 child: SingleChildScrollView(  
 child: Column(children: [  
 TextField(controller: \_guideCtrl, decoration: const InputDecoration(labelText: 'Guide Wallet (0x...)')),  
 TextField(controller: \_amountCtrl, decoration: const InputDecoration(labelText: 'Amount (ETH)'), keyboardType: TextInputType.number),  
 const SizedBox(height: 8),  
 ElevatedButton(onPressed: \_createBooking, child: const Text('Create Booking (deposit)')),  
 const Divider(),  
 TextField(controller: \_bookingCtrl, decoration: const InputDecoration(labelText: 'Booking ID (number)')),  
 ElevatedButton(onPressed: \_markCompleted, child: const Text('Guide: Mark Completed')),  
 ElevatedButton(onPressed: \_release, child: const Text('Tourist: Release to Guide')),  
 const SizedBox(height: 12),  
 Text(\_status),  
 ]),  
 ),  
 ),  
 );  
 }  
}

### pubspec.yaml

name: pravass\_blockchain\_demo  
description: A Flutter demo integrating Ethereum smart contracts (Sepolia) using web3dart  
publish\_to: "none"  
version: 0.0.1  
  
environment:  
 sdk: ">=2.18.0 <4.0.0"  
  
dependencies:  
 flutter:  
 sdk: flutter  
 http: ^0.13.6  
 web3dart: ^2.6.0  
 flutter\_dotenv: ^5.0.2  
 walletconnect\_dart: ^0.0.11 # optional for WalletConnect  
  
dev\_dependencies:  
 flutter\_test:  
 sdk: flutter  
  
flutter:  
 uses-material-design: true  
 assets:  
 - assets/abi/GuideRegistry.abi.json  
 - assets/abi/HandicraftNFT.abi.json  
 - assets/abi/Escrow.abi.json