**توثيق واجهات برمجة التطبيقات (APIs)**

**Flow: Trader submit lands details by default created an auction for it.**

**I have:**

**Lands Collec. That has auctions**

**واجهة المستخدم (Users API)**

**المسار: /api/users**

**العمليات المتاحة:**

* **GET**: جلب جميع المستخدمين المسجلين.
* **POST**: إنشاء مستخدم جديد.

**البيانات المطلوبة في الطلب (POST):**

{

"username": "string",

"role": "TRADER | BROWSER1 | BROWSER2 | ADMIN"

}

**الاستجابة المتوقعة:**

{

"id": "string",

"username": "string",

"role": "TRADER | BROWSER1 | BROWSER2 | ADMIN",

"bids": [],

"lands": []

}

**الأخطاء المحتملة:**

* 400: في حالة فقدان أي بيانات مطلوبة.
* 409: إذا كان اسم المستخدم موجودًا بالفعل.

**واجهة الأحواض (Basins API)**

**المسار: /api/basins**

**العمليات المتاحة:**

* **GET**: جلب جميع الأحواض.
* **POST**: إنشاء حوض جديد.

**البيانات المطلوبة في الطلب (POST):**

{

"name": "string"

}

**الاستجابة المتوقعة:**

{

"id": "string",

"name": "string",

"lands": []

}

**الأخطاء المحتملة:**

* 400: عند فقدان اسم الحوض.
* 409: إذا كان اسم الحوض موجودًا بالفعل.

**واجهة الأراضي (Lands API)**

**المسار: /api/lands**

**العمليات المتاحة:**

* **GET**: جلب جميع الأراضي.
* **POST**: إنشاء أرض جديدة.
* **DELETE**: حذف أرض وجميع البيانات المرتبطة بها.

**البيانات المطلوبة في الطلب (POST):**

{

"basinId": "string",

"traderId": "string",

"name": "string",

"neighborhood": "string",

"location": "string",

"priceBefore": "float",

"priceAfter": "float",

"ownerName": "string",

"ownerPhone": "string"

}

**الاستجابة المتوقعة:**

{

"id": "string",

"basinId": "string",

"traderId": "string",

"name": "string",

"neighborhood": "string",

"location": "string",

"priceBefore": "float",

"priceAfter": "float",

"ownerName": "string",

"ownerPhone": "string",

"auctions": []

}

**الأخطاء المحتملة:**

* 400: عند فقدان أي بيانات مطلوبة.
* 404: في حال عدم العثور على basinId أو traderId.
* 500: خطأ داخلي في الخادم.

**واجهة المزادات (Auctions API)**

**المسار: /api/auctions**

**العمليات المتاحة:**

* **GET**: جلب جميع المزادات.
* **POST**: إنشاء مزاد جديد.
* **DELETE**: حذف مزاد وجميع البيانات المرتبطة به.

**البيانات المطلوبة في الطلب (POST):**

{

"landId": "string",

"startPrice": "float",

"endAt": "DateTime"

}

**الاستجابة المتوقعة:**

{

"id": "string",

"landId": "string",

"startPrice": "float",

"bids": [],

"endAt": "DateTime"

}

**الأخطاء المحتملة:**

* 400: عند فقدان أي بيانات مطلوبة.
* 404: إذا لم يتم العثور على الأرض (landId).
* 500: خطأ داخلي في الخادم.

**واجهة المزايدات (Bids API)**

**المسار: /api/bids**

**العمليات المتاحة:**

* **GET**: جلب جميع المزايدات.
* **POST**: تقديم مزايدة جديدة.

**البيانات المطلوبة في الطلب (POST):**

{

"userId": "string",

"auctionId": "string",

"amount": "float"

}

**الاستجابة المتوقعة:**

{

"id": "string",

"userId": "string",

"auctionId": "string",

"amount": "float"

}

**الأخطاء المحتملة:**

* 400: عند فقدان أي بيانات مطلوبة.
* 404: إذا لم يتم العثور على المستخدم أو المزاد.
* 409: إذا كان العرض المقدم أقل من السعر الحالي للمزاد.

**الإشعارات (Notifications)**

* يتم إرسال إشعارات في الوقت الفعلي عند تقديم عروض جديدة أو عند انتهاء المزادات.

**معالجة الأخطاء (Error Handling)**

* عند فقدان بيانات مطلوبة، يتم إرجاع 400 Bad Request.
* عند محاولة الوصول إلى موارد غير موجودة، يتم إرجاع 404 Not Found.
* عند محاولة الوصول إلى موارد بدون صلاحية، يتم إرجاع 403 Forbidden.
* أي خطأ داخلي في الخادم سيتم التعامل معه عبر 500 Internal Server Error.

**الاختبار (Testing)**

* يمكن اختبار جميع الـ APIs باستخدام أدوات مثل **Postman** أو **cURL** عبر إرسال الطلبات إلى النقاط النهائية (endpoints).
* يجب اختبار السيناريوهات المختلفة، بما في ذلك:
  + طلبات ناجحة.
  + بيانات مفقودة أو غير صحيحة.
  + حذف العناصر وما يترتب عليها من حذف بيانات مرتبطة.

**روابط الاختبار**

يمكنك تجربة الـ APIs عبر إرسال طلبات HTTP إلى الروابط التالية:

* **جلب جميع المستخدمين**: GET /api/users
* **إنشاء مستخدم جديد**: POST /api/users
* **جلب جميع الأراضي**: GET /api/lands
* **إضافة أرض جديدة**: POST /api/lands
* **جلب جميع الأحواض**: GET /api/basins
* **إضافة حوض جديد**: POST /api/basins
* **جلب جميع المزادات**: GET /api/auctions
* **إضافة مزاد جديد**: POST /api/auctions
* **جلب جميع المزايدات**: GET /api/bids
* **إضافة مزايدة جديدة**: POST /api/bids

**استخدام الوقت الحالي في Flutter**

**Ex**

**pubspec.yaml**

**dependencies:**

**socket\_io\_client: ^2.0.3+1**

**------------------------------------------------------------------------------------------------------------**

// lib/services/socket\_service.dart

import 'package:socket\_io\_client/socket\_io\_client.dart' as IO;

import 'package:flutter/foundation.dart';

class SocketService {

static SocketService? \_instance;

late IO.Socket socket;

final String userId;

// Callback for notifications

final Function(Map<String, dynamic>) onNotification;

SocketService.\_({required this.userId, required this.onNotification}) {

\_initSocket();

}

static SocketService getInstance({

required String userId,

required Function(Map<String, dynamic>) onNotification,

}) {

\_instance ??= SocketService.\_(userId: userId, onNotification: onNotification);

return \_instance!;

}

void \_initSocket() {

socket = IO.io('YOUR\_SERVER\_URL', <String, dynamic>{

'transports': ['websocket'],

'autoConnect': false,

});

// Connect to socket

socket.connect();

// Listen for connection

socket.onConnect((\_) {

debugPrint('Socket connected');

// Authenticate user

socket.emit('authenticate', userId);

});

// Listen for notifications

socket.on('notification', (data) {

onNotification(data);

});

// Listen for specific auction events

socket.on('AUCTION\_WON', (data) {

onNotification({

'type': 'AUCTION\_WON',

...data,

});

});

socket.on('AUCTION\_ENDED', (data) {

onNotification({

'type': 'AUCTION\_ENDED',

...data,

});

});

// Listen for connection errors

socket.onError((error) => debugPrint('Socket error: $error'));

socket.onDisconnect((\_) => debugPrint('Socket disconnected'));

}

void disconnect() {

socket.disconnect();

}

// Method to emit new bid event

void emitNewBid(Map<String, dynamic> bidData) {

socket.emit('new\_bid', bidData);

}

// Method to emit auction end event

void emitAuctionEnd(Map<String, dynamic> auctionData) {

socket.emit('auction\_end', auctionData);

}

}

// lib/services/notification\_service.dart

import 'package:flutter\_local\_notifications/flutter\_local\_notifications.dart';

class NotificationService {

static final NotificationService \_instance = NotificationService.\_();

final FlutterLocalNotificationsPlugin \_notifications = FlutterLocalNotificationsPlugin();

NotificationService.\_() {

\_initNotifications();

}

static NotificationService get instance => \_instance;

Future<void> \_initNotifications() async {

const androidSettings = AndroidInitializationSettings('@mipmap/ic\_launcher');

const iosSettings = DarwinInitializationSettings();

const initSettings = InitializationSettings(

android: androidSettings,

iOS: iosSettings,

);

await \_notifications.initialize(initSettings);

}

Future<void> showNotification({

required String title,

required String body,

String? payload,

}) async {

const androidDetails = AndroidNotificationDetails(

'auction\_notifications',

'Auction Notifications',

importance: Importance.high,

priority: Priority.high,

);

const iosDetails = DarwinNotificationDetails();

const details = NotificationDetails(

android: androidDetails,

iOS: iosDetails,

);

await \_notifications.show(

DateTime.now().millisecond,

title,

body,

details,

payload: payload,

);

}

}