**injection\_container.dart**

import "package:cloud\_firestore/cloud\_firestore.dart";  
import "package:firebase\_auth/firebase\_auth.dart";  
import "package:get\_it/get\_it.dart";  
import "package:hall\_sync/features/data/data\_sources/remote/firebase\_remote\_data\_source.dart";  
import "package:hall\_sync/features/data/data\_sources/remote/firebase\_remote\_data\_source\_impl.dart";  
import "package:hall\_sync/features/data/repositories/slot\_repository\_impl.dart";  
import "package:hall\_sync/features/data/repositories/user\_repository\_impl.dart";  
import "package:hall\_sync/features/domain/repositories/slot\_repository.dart";  
import "package:hall\_sync/features/domain/repositories/user\_repository.dart";  
import "package:hall\_sync/features/domain/usecases/book\_slot\_usecase.dart";  
import "package:hall\_sync/features/domain/usecases/delete\_slot\_usecase.dart";  
import "package:hall\_sync/features/domain/usecases/get\_all\_slots\_usecase.dart";  
import "package:hall\_sync/features/domain/usecases/get\_approved\_slots\_usecase.dart";  
import "package:hall\_sync/features/domain/usecases/get\_create\_current\_user\_usecase.dart";  
import "package:hall\_sync/features/domain/usecases/get\_current\_uid\_usecase.dart";  
import "package:hall\_sync/features/domain/usecases/get\_role\_usecase.dart";  
import "package:hall\_sync/features/domain/usecases/get\_user\_slots\_usecase.dart";  
import "package:hall\_sync/features/domain/usecases/is\_sign\_in\_usecase.dart";  
import "package:hall\_sync/features/domain/usecases/sign\_in\_usecase.dart";  
import "package:hall\_sync/features/domain/usecases/sign\_out\_usecase.dart";  
import "package:hall\_sync/features/domain/usecases/sign\_up\_usecase.dart";  
import "package:hall\_sync/features/domain/usecases/toggle\_slot\_approval\_usecase.dart";  
import "package:hall\_sync/features/domain/usecases/update\_slot\_usecase.dart";  
import "package:hall\_sync/features/presentation/cubit/auth/auth\_cubit.dart";  
import "package:hall\_sync/features/presentation/cubit/slot/slot\_cubit.dart";  
import "package:hall\_sync/features/presentation/cubit/user/user\_cubit.dart";  
  
GetIt s1 = GetIt.instance;  
  
Future<void> initializeDependencies() async{  
 s1.registerFactory(() => AuthCubit(  
 isSignInUsecase: s1.call(),   
 getCurrentUIdUsecase: s1.call(),   
 signOutUsecase: s1.call(),  
 getRoleUsecase: s1.call()  
 ));  
  
 s1.registerFactory(() => UserCubit(  
 signInUsecase: s1.call(),   
 signUpUsecase: s1.call(),   
 getCreateCurrentUserUsecase: s1.call()  
 ));  
  
 s1.registerFactory(() => SlotCubit(  
 bookSlotUsecase: s1.call(),   
 toggleSlotApprovalUsecase: s1.call(),   
 deleteSlotUsecase: s1.call(),   
 getAllSlotsUsecase: s1.call(),   
 getApprovedSlotsUsecase: s1.call(),   
 getUserSlotsUsecase: s1.call(),   
 updateSlotUsecase: s1.call()  
 ));  
  
 s1.registerLazySingleton<BookSlotUsecase>(() => BookSlotUsecase(repository: s1.call()));  
 s1.registerLazySingleton<DeleteSlotUsecase>(() => DeleteSlotUsecase(repository: s1.call()));  
 s1.registerLazySingleton<GetAllSlotsUsecase>(() => GetAllSlotsUsecase(repository: s1.call()));  
 s1.registerLazySingleton<GetApprovedSlotsUsecase>(() => GetApprovedSlotsUsecase(repository: s1.call()));  
 s1.registerLazySingleton<GetCreateCurrentUserUsecase>(() => GetCreateCurrentUserUsecase(repository: s1.call()));   
 s1.registerLazySingleton<GetCurrentUIdUsecase>(() => GetCurrentUIdUsecase(repository: s1.call()));  
 s1.registerLazySingleton<GetRoleUsecase>(() => GetRoleUsecase(repository: s1.call()));  
 s1.registerLazySingleton<GetUserSlotsUsecase>(() => GetUserSlotsUsecase(repository: s1.call()));  
 s1.registerLazySingleton<IsSignInUsecase>(() => IsSignInUsecase(repository: s1.call()));  
 s1.registerLazySingleton<SignInUsecase>(() => SignInUsecase(repository: s1.call()));  
 s1.registerLazySingleton<SignOutUsecase>(() => SignOutUsecase(repository: s1.call()));  
 s1.registerLazySingleton<SignUpUsecase>(() => SignUpUsecase(repository: s1.call()));  
 s1.registerLazySingleton<ToggleSlotApprovalUsecase>(() => ToggleSlotApprovalUsecase(repository: s1.call()));  
 s1.registerLazySingleton<UpdateSlotUsecase>(() => UpdateSlotUsecase(repository: s1.call()));  
   
 s1.registerLazySingleton<UserRepository>(() => UserRepositoryImpl(remoteDataSource: s1.call()));  
 s1.registerLazySingleton<SlotRepository>(() => SlotRepositoryImpl(remoteDataSource: s1.call()));  
 s1.registerLazySingleton<FirebaseRemoteDataStore>(() => FirebaseRemoteDataSourceImpl(auth: s1.call(), firestore: s1.call()));  
  
 final auth = FirebaseAuth.instance;  
 final fireStore = FirebaseFirestore.instance;  
  
 s1.registerLazySingleton(() => auth);  
 s1.registerLazySingleton(() => fireStore);  
  
}

**main.dart**

import 'package:flutter/material.dart';  
import 'package:flutter\_bloc/flutter\_bloc.dart';  
import 'package:hall\_sync/config/firebase\_options.dart';  
import 'package:hall\_sync/config/routes/routes.dart';  
import 'package:hall\_sync/features/presentation/cubit/auth/auth\_cubit.dart';  
import 'package:hall\_sync/features/presentation/cubit/slot/slot\_cubit.dart';  
import 'package:hall\_sync/features/presentation/cubit/user/user\_cubit.dart';  
import 'package:hall\_sync/features/presentation/pages/admin\_home\_page.dart';  
import 'package:hall\_sync/features/presentation/pages/home\_page.dart';  
import 'package:hall\_sync/features/presentation/pages/sign\_in\_page.dart';  
import 'injection\_container.dart' as di;  
import 'package:firebase\_core/firebase\_core.dart';  
  
void main() async {  
 WidgetsFlutterBinding.ensureInitialized();  
 await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);  
 await di.initializeDependencies();  
   
 runApp(const MyApp());  
}  
  
class MyApp extends StatelessWidget {  
 const MyApp({super.key});  
  
 @override  
 Widget build(BuildContext context) {  
 return MultiBlocProvider(  
 providers: [  
 BlocProvider<AuthCubit>(create: (\_)=> di.s1<AuthCubit>()..appStarted()),  
 BlocProvider<UserCubit>(create: (\_)=> di.s1<UserCubit>()),  
 BlocProvider<SlotCubit>(create: (\_)=> di.s1<SlotCubit>())  
 ],   
 child: MaterialApp(  
 title: "Hall Sync",  
 onGenerateRoute: onGenerateRoute.route,  
 initialRoute: '/',  
 routes: {  
 '/': (context) {  
 return BlocBuilder<AuthCubit, AuthState>(builder: (context, authState){  
   
 if(authState is Authenticated){  
 if(authState.role=='viewer') return HomePage(uid: authState.uid);  
 if(authState.role=='admin') return AdminHomePage(uid: authState.uid);  
 return const ErrorPage();  
 }  
 if(authState is UnAuthenticated){  
 return const SignInPage();  
 }  
 return const CircularProgressIndicator();  
 });  
 }  
 },  
 ));  
 }  
}

**firebase\_options.dart**

// File generated by FlutterFire CLI.  
// ignore\_for\_file: lines\_longer\_than\_80\_chars, avoid\_classes\_with\_only\_static\_members  
import 'package:firebase\_core/firebase\_core.dart' show FirebaseOptions;  
import 'package:flutter/foundation.dart'  
 show defaultTargetPlatform, kIsWeb, TargetPlatform;  
  
/// Default [FirebaseOptions] for use with your Firebase apps.  
///  
/// Example:  
/// ```dart  
/// import 'firebase\_options.dart';  
/// // ...  
/// await Firebase.initializeApp(  
/// options: DefaultFirebaseOptions.currentPlatform,  
/// );  
/// ```  
class DefaultFirebaseOptions {  
 static FirebaseOptions get currentPlatform {  
 if (kIsWeb) {  
 return web;  
 }  
 switch (defaultTargetPlatform) {  
 case TargetPlatform.android:  
 return android;  
 case TargetPlatform.iOS:  
 return ios;  
 case TargetPlatform.macOS:  
 return macos;  
 case TargetPlatform.windows:  
 throw UnsupportedError(  
 'DefaultFirebaseOptions have not been configured for windows - '  
 'you can reconfigure this by running the FlutterFire CLI again.',  
 );  
 case TargetPlatform.linux:  
 throw UnsupportedError(  
 'DefaultFirebaseOptions have not been configured for linux - '  
 'you can reconfigure this by running the FlutterFire CLI again.',  
 );  
 default:  
 throw UnsupportedError(  
 'DefaultFirebaseOptions are not supported for this platform.',  
 );  
 }  
 }  
  
 static const FirebaseOptions web = FirebaseOptions(  
 apiKey: 'AIzaSyAsI3rEFYtmrS2Og3UxBXWblK3UTyPs-94',  
 appId: '1:57601764622:web:15ab60217bb39d310b3b5f',  
 messagingSenderId: '57601764622',  
 projectId: 'hall-sync',  
 authDomain: 'hall-sync.firebaseapp.com',  
 storageBucket: 'hall-sync.appspot.com',  
 measurementId: 'G-QP31LW8JHB',  
 );  
  
 static const FirebaseOptions android = FirebaseOptions(  
 apiKey: 'AIzaSyCjX5vkVxjoOjkGy3tcQCU9O0In1rdmtlQ',  
 appId: '1:57601764622:android:d6163d9b889e57520b3b5f',  
 messagingSenderId: '57601764622',  
 projectId: 'hall-sync',  
 storageBucket: 'hall-sync.appspot.com',  
 );  
  
 static const FirebaseOptions ios = FirebaseOptions(  
 apiKey: 'AIzaSyBzKp9nwRa1mzjyXYhUOy90DSl1i99IOs4',  
 appId: '1:57601764622:ios:e60fe25f37f9b8f10b3b5f',  
 messagingSenderId: '57601764622',  
 projectId: 'hall-sync',  
 storageBucket: 'hall-sync.appspot.com',  
 iosBundleId: 'com.hariharanvj.hallSync',  
 );  
  
 static const FirebaseOptions macos = FirebaseOptions(  
 apiKey: 'AIzaSyBzKp9nwRa1mzjyXYhUOy90DSl1i99IOs4',  
 appId: '1:57601764622:ios:3d48d7c8d2ebd8be0b3b5f',  
 messagingSenderId: '57601764622',  
 projectId: 'hall-sync',  
 storageBucket: 'hall-sync.appspot.com',  
 iosBundleId: 'com.hariharanvj.hallSync.RunnerTests',  
 );  
}

**routes.dart**

import 'package:flutter/material.dart';  
import 'package:hall\_sync/config/utils/constants.dart';  
import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:hall\_sync/features/presentation/pages/admin\_home\_page.dart';  
import 'package:hall\_sync/features/presentation/pages/home\_page.dart';  
import 'package:hall\_sync/features/presentation/pages/sign\_in\_page.dart';  
import 'package:hall\_sync/features/presentation/pages/sign\_up\_page.dart';  
import 'package:hall\_sync/features/presentation/pages/update\_slot\_page.dart';  
  
class onGenerateRoute{  
 static Route<dynamic> route(RouteSettings settings){  
 final arg = settings.arguments;  
  
 switch(settings.name){  
 case PageConstants.signupPage:  
 return materialBuilder(widget: const SignUpPage());   
   
 case PageConstants.signinPage:  
 return materialBuilder(widget: const SignInPage());  
  
 case PageConstants.homePage:  
 if(arg is String){  
 return materialBuilder(widget: HomePage(uid:arg,));  
 }  
 else{  
 return materialBuilder(widget: const ErrorPage());  
 }  
 case PageConstants.updateSlotPage:  
 if(arg is SlotEntity){  
 return materialBuilder(widget: UpdateSlotPage(slot:arg,));  
 }  
 else{  
 return materialBuilder(widget: const ErrorPage());  
 }  
 case PageConstants.adminHomePage:  
 if(arg is String){  
 return materialBuilder(widget: AdminHomePage(uid:arg,));  
 }  
 else{  
 return materialBuilder(widget: const ErrorPage());  
 }  
 default:  
 return materialBuilder(widget: const ErrorPage());  
 }  
 }  
}  
  
class ErrorPage extends StatelessWidget {  
 const ErrorPage({super.key});  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: const Text("Error")  
 ),  
 body: const Center(  
 child: Text("Error"),  
 ),  
 );  
 }  
}  
  
MaterialPageRoute materialBuilder({required Widget widget}){  
 return MaterialPageRoute(builder: (\_)=>widget);  
}

**constants.dart**

class AppConstants{  
  
}  
  
class PageConstants{  
 static const String signupPage = 'signup';  
 static const String signinPage = 'signin';  
 static const String homePage = 'homepage';  
 static const String adminHomePage = 'adminHomepage';  
 static const String updateSlotPage = 'updateSlotPage';  
}

**firebase\_remote\_data\_source.dart**

import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:hall\_sync/features/domain/entities/user\_entity.dart';  
  
abstract class FirebaseRemoteDataStore{  
 Future<bool> isSignIn();  
 Future<void> signIn(UserEntity user);  
 Future<void> signUp(UserEntity user);  
 Future<void> signOut();  
 Future<String> getCurrentUId();  
 Future<String> getRole(String uid);  
 Future<void> getCreateCurrentUser(UserEntity user);  
 Future<void> bookSlot(SlotEntity slot);  
 Future<void> updateSlot(SlotEntity slot);  
 Future<void> deleteSlot(SlotEntity slot);  
 Future<void> toggleSlotApproval(SlotEntity slot);  
 Stream<List<SlotEntity>> getUserSlots(String uid);  
 Stream<List<SlotEntity>> getAllSlots();  
 Stream<List<SlotEntity>> getApprovedSlots();  
}

**firebase\_remote\_data\_source\_impl.dart**

import 'package:hall\_sync/features/data/data\_sources/remote/firebase\_remote\_data\_source.dart';  
import 'package:firebase\_auth/firebase\_auth.dart';  
import 'package:cloud\_firestore/cloud\_firestore.dart';  
import 'package:hall\_sync/features/data/models/slot\_model.dart';  
import 'package:hall\_sync/features/data/models/user\_model.dart';  
import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:hall\_sync/features/domain/entities/user\_entity.dart';  
  
class FirebaseRemoteDataSourceImpl implements FirebaseRemoteDataStore{  
 final FirebaseAuth auth;  
 final FirebaseFirestore firestore;  
   
 const FirebaseRemoteDataSourceImpl({required this.auth,required this.firestore});  
  
 static String slotCollectionPath = 'slots';  
 static String userCollectionPath = 'users';  
  
 @override  
 Future<void> toggleSlotApproval(SlotEntity slot) async {  
 Map<String, dynamic> slotMap = {};  
 final slotCollectionRef = firestore.collection(slotCollectionPath);  
 if(slot.isApproved!=null) slotMap['isApproved'] = !(slot.isApproved?? false) ;  
 slotCollectionRef.doc(slot.id!).update(slotMap);  
 }  
  
  
 @override  
 Future<void> bookSlot(SlotEntity slot) async {  
 final slotCollectionRef = firestore.collection(slotCollectionPath);  
 final querySnapshot = await slotCollectionRef  
 .where('startTime', isGreaterThanOrEqualTo: slot.startTime)  
 .get();  
  
 final filteredSnapshots = querySnapshot.docs.where((doc) {  
 final endTime = doc.data()['endTime'];  
 final isApproved = doc.data()['isApproved'];  
 return endTime <= slot.endTime && isApproved;  
 }).toList();  
  
 if (filteredSnapshots.isNotEmpty) {  
 throw Exception('Overlapping slot found. Booking failed.');  
 }  
  
 final slotId = slotCollectionRef.doc().id;  
  
 slotCollectionRef.doc(slotId).get().then((\_slot){  
 final newSlot = SlotModel(  
 id: slotId,  
 name: slot.name,  
 startTime: slot.startTime,  
 endTime: slot.endTime,  
 description: slot.description,  
 isApproved: false,  
 inchargeId: slot.inchargeId  
 ).toDocument();  
  
 if(!\_slot.exists) slotCollectionRef.doc(slotId).set(newSlot);  
 });  
  
 }  
  
  
 @override  
 Future<void> deleteSlot(SlotEntity slot) async {  
 final slotCollectionRef = firestore.collection(slotCollectionPath);  
 slotCollectionRef.doc(slot.id).get().then((slot) => {  
 if(slot.exists)  
 slotCollectionRef.doc(slot.id).delete()   
 });  
 return;  
 }  
  
 @override  
 Stream<List<SlotEntity>> getAllSlots() {  
 final slotCollectionRef = firestore.collection(slotCollectionPath);  
 return slotCollectionRef.snapshots().map((querySnaps) {  
 return querySnaps.docs.map((docSnap) => SlotModel.fromSnapshot(docSnap)).toList();  
 });  
 }  
  
 @override  
 Future<void> getCreateCurrentUser(UserEntity user) async {  
 final userCollectionRef = firestore.collection(userCollectionPath);  
 final uid = await getCurrentUId();  
  
 userCollectionRef.doc(uid).get().then((\_user){  
 final newUser = UserModel(uid:uid, name: user.name, type:"member").toDocument();  
  
 if(\_user.exists){  
 userCollectionRef.doc(uid).set(newUser);  
 }  
 return;  
 });   
 }  
  
 @override  
 Future<String> getCurrentUId() async => auth.currentUser!.uid;  
  
 @override  
 Stream<List<SlotEntity>> getUserSlots(String uid) {  
 final slotCollectionRef = firestore.collection(slotCollectionPath);  
 return slotCollectionRef.where('inchargeId',isEqualTo: uid).snapshots().map((querySnapshot){  
 return querySnapshot.docs.map((docSnap) => SlotModel.fromSnapshot(docSnap)).toList();  
 });  
 }  
  
 @override  
 Future<String> getRole(String uid) async{  
 final userCollectionRef = firestore.collection(userCollectionPath);  
 String type="viewer";  
 userCollectionRef.doc(uid).get().then((user) => {  
 if(user.exists){  
 type = UserModel.fromDocument(user).type!  
 }  
 });  
 return type;  
 }  
  
 @override  
 Stream<List<SlotEntity>> getApprovedSlots() {  
 final slotCollectionRef = firestore.collection(slotCollectionPath);  
 return slotCollectionRef.where('isApproved',isEqualTo: true).snapshots().map((querySnapshot){  
 return querySnapshot.docs.map((docSnap) => SlotModel.fromSnapshot(docSnap)).toList();  
 });  
 }  
  
 @override  
 Future<bool> isSignIn() async => auth.currentUser?.uid!=null;  
  
 @override  
 Future<void> signIn(UserEntity user) async => auth.signInWithEmailAndPassword(email:user.email!, password:user.password!);  
   
  
 @override  
 Future<void> signOut() async => auth.signOut();  
  
 @override  
 Future<void> signUp(UserEntity user) async => auth.createUserWithEmailAndPassword(email: user.email!, password: user.password!);  
  
 @override  
 Future<void> updateSlot(SlotEntity slot) async {  
 final slotCollectionRef = firestore.collection(slotCollectionPath);  
  
 slotCollectionRef.doc(slot.id).get().then((oldSlot){  
 if(oldSlot.exists) {  
 slotCollectionRef.doc(slot.id).update(  
 {  
 "name": slot.name,  
 "description": slot.description,  
 });  
 }  
 return;  
 });  
   
 }  
  
}

**slot\_model.dart**

import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:cloud\_firestore/cloud\_firestore.dart';  
  
class SlotModel extends SlotEntity{  
 final String ? id;  
 final String ? name;  
 final DateTime ? startTime;  
 final DateTime ? endTime;  
 final String ? description;  
 final String ? inchargeId;  
 final bool ? isApproved;  
   
 const SlotModel({this.id, this.name, this.startTime, this.endTime, this.description, this.inchargeId, this.isApproved});  
  
 factory SlotModel.fromSnapshot(DocumentSnapshot documentSnapshot){  
 return SlotModel(  
 id: documentSnapshot.get('id'),  
 name: documentSnapshot.get('name'),  
 startTime: DateTime.parse(documentSnapshot.get('startTime')),  
 endTime: DateTime.parse(documentSnapshot.get('endTime')),  
 description: documentSnapshot.get('description'),  
 inchargeId: documentSnapshot.get('inchargeId'),  
 isApproved: documentSnapshot.get('isApproved')  
 );  
 }  
  
 Map<String, dynamic> toDocument() {  
 return {  
 'id': id,  
 'name': name,  
 'startTime': startTime!.toIso8601String(),  
 'endTime': endTime!.toIso8601String(),  
 'isApproved': isApproved,   
 'description': description,  
 'inchargeId': inchargeId,  
 };  
 }  
}

**user\_model.dart**

import 'package:cloud\_firestore/cloud\_firestore.dart';  
import 'package:hall\_sync/features/domain/entities/user\_entity.dart';  
  
class UserModel extends UserEntity{  
 const UserModel({  
 String? uid,  
 String? name,  
 String? type,  
 String? email,  
 String? password,  
 String? profileUrl,  
 }):super(uid: uid, name: name, type: type, email: email, password: password, profileUrl: profileUrl);  
  
 factory UserModel.fromDocument(DocumentSnapshot documentSnapshot){  
 return UserModel(  
 uid: documentSnapshot.get('uid'),  
 name: documentSnapshot.get('name'),  
 type: documentSnapshot.get('type'),  
 email: documentSnapshot.get('email'),  
 profileUrl: documentSnapshot.get('profileUrl'),  
 );  
 }  
  
 Map<String, dynamic> toDocument(){  
 return {  
 "uid": uid,  
 "name": name,  
 "type": type,  
 "email": email,  
 "profileUrl": profileUrl  
 };  
 }  
}

**slot\_repository\_impl.dart**

import 'package:hall\_sync/features/data/data\_sources/remote/firebase\_remote\_data\_source.dart';  
import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:hall\_sync/features/domain/repositories/slot\_repository.dart';  
  
class SlotRepositoryImpl extends SlotRepository{  
 final FirebaseRemoteDataStore remoteDataSource;  
 SlotRepositoryImpl({required this.remoteDataSource});  
  
 @override  
 Future<void> bookSlot(SlotEntity slot) =>   
 remoteDataSource.bookSlot(slot);  
  
 @override  
 Future<void> deleteSlot(SlotEntity slot) =>   
 remoteDataSource.deleteSlot(slot);  
  
 @override  
 Stream<List<SlotEntity>> getAllSlots() =>   
 remoteDataSource.getAllSlots();  
  
 @override  
 Stream<List<SlotEntity>> getApprovedSlots() =>   
 remoteDataSource.getApprovedSlots();  
  
 @override  
 Stream<List<SlotEntity>> getUserSlots(String uid) =>   
 remoteDataSource.getUserSlots(uid);  
  
 @override  
 Future<void> toggleSlotApproval(SlotEntity slot) =>   
 remoteDataSource.toggleSlotApproval(slot);  
  
 @override  
 Future<void> updateSlot(SlotEntity slot) =>   
 remoteDataSource.updateSlot(slot);  
  
}

**user\_repository\_impl.dart**

import 'package:hall\_sync/features/data/data\_sources/remote/firebase\_remote\_data\_source.dart';  
import 'package:hall\_sync/features/domain/entities/user\_entity.dart';  
import 'package:hall\_sync/features/domain/repositories/user\_repository.dart';  
  
class UserRepositoryImpl extends UserRepository {  
 final FirebaseRemoteDataStore remoteDataSource;  
 UserRepositoryImpl({required this.remoteDataSource});  
   
 @override  
 Future<void> getCreateCurrentUser(UserEntity user) async =>  
 remoteDataSource.getCreateCurrentUser(user);  
   
 @override  
 Future<String> getRole(String uid) async =>   
 remoteDataSource.getRole(uid);  
  
 @override  
 Future<String> getCurrentUId() async =>  
 remoteDataSource.getCurrentUId();  
   
  
 @override  
 Future<bool> isSignedIn() =>  
 remoteDataSource.isSignIn();  
   
  
 @override  
 Future<void> login(UserEntity user) =>  
 remoteDataSource.signIn(user);  
   
  
 @override  
 Future<void> signOut() =>  
 remoteDataSource.signOut();  
  
 @override  
 Future<void> signup(UserEntity user) =>  
 remoteDataSource.signUp(user);  
   
}

**slot\_entity.dart**

import 'package:equatable/equatable.dart';  
  
class SlotEntity extends Equatable {  
 final String? id;  
 final String? name;  
 final DateTime? startTime;  
 final DateTime? endTime;  
 final String? description;  
 final String? inchargeId;  
 final bool? isApproved;  
  
 const SlotEntity({  
 this.id,  
 this.name,  
 this.startTime,  
 this.endTime,  
 this.description,  
 this.inchargeId,  
 this.isApproved  
 });  
   
 @override  
 List<Object?> get props {  
 return [id, name, startTime, endTime, description, inchargeId, isApproved];  
 }  
}

**user\_entity.dart**

import 'package:equatable/equatable.dart';  
  
class UserEntity extends Equatable{  
 final String? uid;  
 final String? name;  
 final String? type;  
 final String? email;  
 final String? password;  
 final String? profileUrl;  
  
 const UserEntity({this.uid, this.name, this.type, this.email, this.password, this.profileUrl});  
   
 @override  
 List<Object?> get props{  
 return [uid, name, type, email, password, profileUrl];  
 }  
  
}

**slot\_repository.dart**

import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
  
abstract class SlotRepository{  
 Future<void> bookSlot(SlotEntity slot);  
 Future<void> updateSlot(SlotEntity slot);  
 Future<void> deleteSlot(SlotEntity slot);  
 Future<void> toggleSlotApproval(SlotEntity slot);  
 Stream<List<SlotEntity>> getUserSlots(String uid);  
 Stream<List<SlotEntity>> getAllSlots();  
 Stream<List<SlotEntity>> getApprovedSlots();  
}

**user\_repository.dart**

import 'package:hall\_sync/features/domain/entities/user\_entity.dart';  
  
abstract class UserRepository{  
 Future<void> login(UserEntity user);  
 Future<void> signup(UserEntity user);  
 Future<void> signOut();  
 Future<bool> isSignedIn();  
 Future<String> getRole(String uid);  
 Future<String> getCurrentUId();  
 Future<void> getCreateCurrentUser(UserEntity user);  
}

**book\_slot\_usecase.dart**

import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:hall\_sync/features/domain/repositories/slot\_repository.dart';  
  
class BookSlotUsecase{  
 final SlotRepository repository;  
  
 BookSlotUsecase({required this.repository});  
  
 Future<void> call(SlotEntity slot){  
 return repository.bookSlot(slot);  
 }  
}

**delete\_slot\_usecase.dart**

import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:hall\_sync/features/domain/repositories/slot\_repository.dart';  
  
class DeleteSlotUsecase{  
 final SlotRepository repository;  
  
 DeleteSlotUsecase({required this.repository});  
  
 Future<void> call(SlotEntity slot){  
 return repository.deleteSlot(slot);  
 }  
}

**get\_all\_slots\_usecase.dart**

import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:hall\_sync/features/domain/repositories/slot\_repository.dart';  
  
class GetAllSlotsUsecase {  
 final SlotRepository repository;  
  
 GetAllSlotsUsecase({required this.repository});  
  
 Stream<List<SlotEntity>> call(){  
 return repository.getAllSlots();  
 }  
}

**get\_approved\_slots\_usecase.dart**

import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:hall\_sync/features/domain/repositories/slot\_repository.dart';  
  
class GetApprovedSlotsUsecase{  
 final SlotRepository repository;  
  
 GetApprovedSlotsUsecase({required this.repository});  
  
 Stream<List<SlotEntity>> call(){  
 return repository.getApprovedSlots();  
 }  
}

**get\_create\_current\_user\_usecase.dart**

import 'package:hall\_sync/features/domain/entities/user\_entity.dart';  
import 'package:hall\_sync/features/domain/repositories/user\_repository.dart';  
  
class GetCreateCurrentUserUsecase {  
  
 final UserRepository repository;  
  
 GetCreateCurrentUserUsecase({required this.repository});  
  
 Future<void> call(UserEntity user)async{  
 return repository.getCreateCurrentUser(user);  
 }  
}

**get\_current\_uid\_usecase.dart**

import 'package:hall\_sync/features/domain/repositories/user\_repository.dart';  
  
class GetCurrentUIdUsecase{  
  
 final UserRepository repository;  
   
 GetCurrentUIdUsecase({required this.repository});  
  
 Future<String> call(){  
 return repository.getCurrentUId();  
 }  
}

**get\_role\_usecase.dart**

import 'package:hall\_sync/features/domain/repositories/user\_repository.dart';  
  
class GetRoleUsecase{  
  
 final UserRepository repository;  
   
 GetRoleUsecase({required this.repository});  
  
 Future<String> call(String uid){  
 return repository.getRole(uid);  
 }  
}

**get\_user\_slots\_usecase.dart**

import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:hall\_sync/features/domain/repositories/slot\_repository.dart';  
  
class GetUserSlotsUsecase {  
 final SlotRepository repository;  
  
 GetUserSlotsUsecase({required this.repository});  
  
 Stream<List<SlotEntity>> call(String uid){  
 return repository.getUserSlots(uid);  
 }  
}

**is\_sign\_in\_usecase.dart**

import 'package:hall\_sync/features/domain/repositories/user\_repository.dart';  
  
class IsSignInUsecase{  
 final UserRepository repository;  
  
 IsSignInUsecase({required this.repository});  
  
 Future<bool> call(){  
 return repository.isSignedIn();  
 }  
  
}

**sign\_in\_usecase.dart**

import 'package:hall\_sync/features/domain/entities/user\_entity.dart';  
import 'package:hall\_sync/features/domain/repositories/user\_repository.dart';  
  
class SignInUsecase{  
 final UserRepository repository;  
  
 SignInUsecase({required this.repository});  
  
 Future<void> call(UserEntity user){  
 return repository.login(user);  
 }  
}

**sign\_out\_usecase.dart**

import 'package:hall\_sync/features/domain/repositories/user\_repository.dart';  
  
class SignOutUsecase{  
 final UserRepository repository;  
  
 SignOutUsecase({required this.repository});  
  
 Future<void> call(){  
 return repository.signOut();  
 }  
}

**sign\_up\_usecase.dart**

import 'package:hall\_sync/features/domain/entities/user\_entity.dart';  
import 'package:hall\_sync/features/domain/repositories/user\_repository.dart';  
  
class SignUpUsecase{  
 final UserRepository repository;  
  
 SignUpUsecase({required this.repository});  
  
 Future<void> call(UserEntity user){  
 return repository.signup(user);  
 }  
}

**toggle\_slot\_approval\_usecase.dart**

import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:hall\_sync/features/domain/repositories/slot\_repository.dart';  
  
class ToggleSlotApprovalUsecase{  
 final SlotRepository repository;  
  
 ToggleSlotApprovalUsecase({required this.repository});  
  
 Future<void> call(SlotEntity slot){  
 return repository.toggleSlotApproval(slot);  
 }  
}

**update\_slot\_usecase.dart**

import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:hall\_sync/features/domain/repositories/slot\_repository.dart';  
  
class UpdateSlotUsecase{  
 final SlotRepository repository;  
  
 UpdateSlotUsecase({required this.repository});  
  
 Future<void> call(SlotEntity slot){  
 return repository.updateSlot(slot);  
 }  
}

**auth\_cubit.dart**

import 'dart:io';  
  
import 'package:equatable/equatable.dart';  
import 'package:flutter\_bloc/flutter\_bloc.dart';  
import 'package:hall\_sync/features/domain/usecases/get\_current\_uid\_usecase.dart';  
import 'package:hall\_sync/features/domain/usecases/get\_role\_usecase.dart';  
import 'package:hall\_sync/features/domain/usecases/is\_sign\_in\_usecase.dart';  
import 'package:hall\_sync/features/domain/usecases/sign\_out\_usecase.dart';  
  
part "auth\_state.dart";  
  
class AuthCubit extends Cubit<AuthState>{  
 final IsSignInUsecase isSignInUsecase;  
 final GetCurrentUIdUsecase getCurrentUIdUsecase;  
 final GetRoleUsecase getRoleUsecase;  
 final SignOutUsecase signOutUsecase;  
 AuthCubit({required this.isSignInUsecase, required this.getCurrentUIdUsecase, required this.signOutUsecase, required this.getRoleUsecase}):super(AuthInitial());  
  
 Future<void> appStarted() async{  
 try{  
 final isSignIn = await isSignInUsecase.call();  
 if(isSignIn){  
 final uid = await getCurrentUIdUsecase.call();  
 final role = await getRoleUsecase.call(uid);  
 emit(Authenticated(uid: uid, role: role));  
 }  
 else{  
 emit(UnAuthenticated());  
 }  
 }  
 on SocketException catch(\_){  
 emit(UnAuthenticated());  
 }  
 }  
  
 Future<void> loggedIn() async{  
 try{  
 final uid = await getCurrentUIdUsecase.call();  
 final role = await getRoleUsecase.call(uid);  
 emit(Authenticated(uid: uid,role: role));  
 }  
 on SocketException catch(\_){  
 emit(UnAuthenticated());  
 }  
 }  
  
 Future<void> loggedOut() async{  
 try{  
 await signOutUsecase.call();  
 emit(UnAuthenticated());  
 }  
 on SocketException catch(\_){  
 emit(UnAuthenticated());  
 }  
 }  
}

**auth\_state.dart**

part of "auth\_cubit.dart";  
  
abstract class AuthState extends Equatable{  
 const AuthState();  
}  
  
class AuthInitial extends AuthState{  
 @override  
 List<Object?> get props => [];  
}  
  
class Authenticated extends AuthState{  
 final String uid;  
 final String role;  
 const Authenticated({required this.uid, required this.role});  
   
 @override  
 List<Object?> get props => [];  
}  
  
class UnAuthenticated extends AuthState{  
 @override  
 List<Object?> get props => [];  
}

**slot\_cubit.dart**

import 'dart:io';  
  
import 'package:equatable/equatable.dart';  
import 'package:flutter\_bloc/flutter\_bloc.dart';  
import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:hall\_sync/features/domain/usecases/book\_slot\_usecase.dart';  
import 'package:hall\_sync/features/domain/usecases/delete\_slot\_usecase.dart';  
import 'package:hall\_sync/features/domain/usecases/get\_all\_slots\_usecase.dart';  
import 'package:hall\_sync/features/domain/usecases/get\_approved\_slots\_usecase.dart';  
import 'package:hall\_sync/features/domain/usecases/get\_user\_slots\_usecase.dart';  
import 'package:hall\_sync/features/domain/usecases/toggle\_slot\_approval\_usecase.dart';  
import 'package:hall\_sync/features/domain/usecases/update\_slot\_usecase.dart';  
  
part 'slot\_state.dart';  
  
class SlotCubit extends Cubit<SlotState>{  
  
 final BookSlotUsecase bookSlotUsecase;  
 final ToggleSlotApprovalUsecase toggleSlotApprovalUsecase;  
 final UpdateSlotUsecase updateSlotUsecase;  
 final DeleteSlotUsecase deleteSlotUsecase;  
 final GetAllSlotsUsecase getAllSlotsUsecase;  
 final GetUserSlotsUsecase getUserSlotsUsecase;  
 final GetApprovedSlotsUsecase getApprovedSlotsUsecase;  
  
 SlotCubit({required this.bookSlotUsecase, required this.toggleSlotApprovalUsecase, required this.deleteSlotUsecase, required this.getAllSlotsUsecase, required this.getApprovedSlotsUsecase, required this.getUserSlotsUsecase, required this.updateSlotUsecase}): super(SlotInitial());  
  
 Future<void> bookSlot({required SlotEntity slot})async {  
 try{  
 await bookSlotUsecase(slot);  
 }  
 on SocketException catch(\_){  
 emit(SlotFailure());  
 }  
 catch(\_){  
 emit(SlotFailure());  
 }  
 }  
  
 Future<void> updateSlot({required SlotEntity slot})async {  
 try{  
 await updateSlotUsecase(slot);  
 }  
 on SocketException catch(\_){  
 emit(SlotFailure());  
 }  
 catch(\_){  
 emit(SlotFailure());  
 }  
 }  
  
 Future<void> deleteSlot({required SlotEntity slot})async {  
 try{  
 await deleteSlotUsecase(slot);  
 }  
 on SocketException catch(\_){  
 emit(SlotFailure());  
 }  
 catch(\_){  
 emit(SlotFailure());  
 }  
 }  
  
 Future<void> toggleApproval({required SlotEntity slot})async {  
 try{  
 await toggleSlotApprovalUsecase(slot);  
 }  
 on SocketException catch(\_){  
 emit(SlotFailure());  
 }  
 catch(\_){  
 emit(SlotFailure());  
 }  
 }  
  
 Future<void> getAllSlots()async {  
 emit(SlotLoading());  
 try{  
 getAllSlotsUsecase().listen((slots) {  
 emit(SlotLoaded(slots: slots));  
 });  
 }  
 on SocketException catch(\_){  
 emit(SlotFailure());  
 }  
 catch(\_){  
 emit(SlotFailure());  
 }  
 }  
  
 Future<void> getApprovedSlots()async {  
 emit(SlotLoading());  
 try{  
 getApprovedSlotsUsecase().listen((slots) {  
 emit(SlotLoaded(slots: slots));  
 });  
 }  
 on SocketException catch(\_){  
 emit(SlotFailure());  
 }  
 catch(\_){  
 emit(SlotFailure());  
 }  
 }  
  
 Future<void> getUserSlots({required String uid})async {  
 emit(SlotLoading());  
 try{  
 getUserSlotsUsecase(uid).listen((slots) {  
 emit(SlotLoaded(slots: slots));  
 });  
 }  
 on SocketException catch(\_){  
 emit(SlotFailure());  
 }  
 catch(\_){  
 emit(SlotFailure());  
 }  
 }  
  
}

**slot\_state.dart**

part of 'slot\_cubit.dart';  
  
abstract class SlotState extends Equatable{  
 const SlotState();  
}  
  
class SlotInitial extends SlotState{   
 @override  
 List<Object?> get props => [];  
}  
  
class SlotLoaded extends SlotState{  
 final List<SlotEntity> slots;   
 const SlotLoaded({required this.slots});  
   
 @override  
 List<Object?> get props => [];  
}  
class SlotLoading extends SlotState{   
 @override  
 List<Object?> get props => [];  
}  
  
class SlotFailure extends SlotState{   
 @override  
 List<Object?> get props => [];  
}

**user\_cubit.dart**

import 'dart:io';  
  
import 'package:equatable/equatable.dart';  
import 'package:flutter\_bloc/flutter\_bloc.dart';  
import 'package:hall\_sync/features/domain/entities/user\_entity.dart';  
import 'package:hall\_sync/features/domain/usecases/get\_create\_current\_user\_usecase.dart';  
import 'package:hall\_sync/features/domain/usecases/sign\_in\_usecase.dart';  
import 'package:hall\_sync/features/domain/usecases/sign\_up\_usecase.dart';  
  
part "user\_state.dart";  
  
class UserCubit extends Cubit<UserState>{  
 final SignInUsecase signInUsecase;  
 final SignUpUsecase signUpUsecase;  
 final GetCreateCurrentUserUsecase getCreateCurrentUserUsecase;  
  
 UserCubit({required this.signInUsecase, required this.signUpUsecase, required this.getCreateCurrentUserUsecase}):super(UserInitial());  
  
 Future<void> handleSignIn({required UserEntity user}) async {  
 emit(UserLoading());  
 try{  
 await signInUsecase.call(user);  
 emit(UserSuccess(user: user));  
 }  
 on SocketException catch(\_){  
 emit(UserFailure());  
 }  
 catch(\_){  
 emit(UserFailure());  
 }  
 }  
  
 Future<void> handleSignup({required UserEntity user}) async {  
 emit(UserLoading());  
 try{  
 await signUpUsecase(user);  
 await getCreateCurrentUserUsecase(user);  
 emit(UserSuccess(user: user));  
 }  
 on SocketException catch(\_){  
 emit(UserFailure());  
 }  
 catch(\_){  
 UserFailure();  
 }  
 }  
}

**user\_state.dart**

part of 'user\_cubit.dart';  
  
abstract class UserState extends Equatable{  
 const UserState();  
}  
  
class UserInitial extends UserState{  
 @override  
 List<Object?> get props => [];  
}  
  
class UserLoading extends UserState{  
 @override  
 List<Object?> get props => [];  
}  
  
class UserSuccess extends UserState{  
 final UserEntity user;  
 const UserSuccess({required this.user});  
   
 @override  
 List<Object?> get props => [];  
}  
  
class UserFailure extends UserState{  
 @override  
 List<Object?> get props => [];  
}

**admin\_home\_page.dart**

import 'package:flutter/material.dart';  
  
class AdminHomePage extends StatelessWidget {  
 final String uid;  
 const AdminHomePage({super.key, required this.uid});  
   
 @override  
 Widget build(BuildContext context) {  
 return const Text("Admin Homepage");  
 }  
}

**approve\_events\_page.dart**

import 'package:flutter/material.dart';  
  
class ApproveEventsPage extends StatelessWidget {  
 const ApproveEventsPage({super.key});  
  
 @override  
 Widget build(BuildContext context) {  
 return const Placeholder();  
 }  
}

**book\_events\_page.dart**

import 'package:flutter/material.dart';  
import 'package:flutter\_bloc/flutter\_bloc.dart';  
import 'package:hall\_sync/config/utils/constants.dart';  
import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:hall\_sync/features/presentation/cubit/auth/auth\_cubit.dart';  
import 'package:hall\_sync/features/presentation/cubit/slot/slot\_cubit.dart';  
import 'package:hall\_sync/features/presentation/widgets/common.dart';  
import 'package:intl/intl.dart';  
  
class BookEventsPage extends StatefulWidget {  
 final String uid;  
 const BookEventsPage({Key? key, required this.uid}) : super(key: key);  
  
 @override  
 \_BookEventsPageState createState() => \_BookEventsPageState();  
}  
  
class \_BookEventsPageState extends State<BookEventsPage> {  
 late TextEditingController \_nameController;  
 late TextEditingController \_descriptionController;  
 DateTime? \_startTime;  
 DateTime? \_endTime;  
  
 @override  
 void initState() {  
 \_nameController = TextEditingController();  
 \_descriptionController = TextEditingController();  
 super.initState();  
 }  
  
 @override  
 void dispose() {  
 \_nameController.dispose();  
 \_descriptionController.dispose();  
 super.dispose();  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: const Text('Book Slot'),  
 actions: [  
 IconButton(  
 onPressed: () {  
 BlocProvider.of<AuthCubit>(context).loggedOut();  
 },  
 icon: const Icon(Icons.logout),  
 )  
 ],  
 ),  
 body: SingleChildScrollView(  
 child: Padding(  
 padding: const EdgeInsets.all(16.0),  
 child: Column(  
 crossAxisAlignment: CrossAxisAlignment.start,  
 children: [  
 TextFormField(  
 controller: \_nameController,  
 decoration: InputDecoration(labelText: 'Name'),  
 ),  
 const SizedBox(height: 16),  
 TextFormField(  
 controller: \_descriptionController,  
 decoration: InputDecoration(labelText: 'Description'),  
 ),  
 const SizedBox(height: 16),  
 TextFormField(  
 onTap: () async {  
 DateTime? selectedTime = await showDatePicker(  
 context: context,  
 initialDate: DateTime.now(),  
 firstDate: DateTime.now(),  
 lastDate: DateTime.now().add(Duration(days: 365)),  
 );  
 setState(() {  
 \_startTime = selectedTime;  
 });  
 },  
 readOnly: true,  
 decoration: InputDecoration(  
 labelText: 'Start Time',  
 suffixIcon: Icon(Icons.calendar\_today),  
 ),  
 controller: TextEditingController(  
 text: \_startTime != null  
 ? DateFormat.yMd().add\_jm().format(\_startTime!)  
 : '',  
 ),  
 ),  
 const SizedBox(height: 16),  
 TextFormField(  
 onTap: () async {  
 DateTime? selectedTime = await showDatePicker(  
 context: context,  
 initialDate: DateTime.now(),  
 firstDate: DateTime.now(),  
 lastDate: DateTime.now().add(Duration(days: 365)),  
 );  
 setState(() {  
 \_endTime = selectedTime;  
 });  
 },  
 readOnly: true,  
 decoration: InputDecoration(  
 labelText: 'End Time',  
 suffixIcon: Icon(Icons.calendar\_today),  
 ),  
 controller: TextEditingController(  
 text: \_endTime != null  
 ? DateFormat.yMd().add\_jm().format(\_endTime!)  
 : '',  
 ),  
 ),  
 const SizedBox(height: 16),  
 ElevatedButton(  
 onPressed: () => \_saveSlot(context),  
 child: const Text('Book'),  
 ),  
 ],  
 ),  
 ),  
 ),  
 );  
 }  
  
 void \_saveSlot(BuildContext context) {  
 final String name = \_nameController.text.trim();  
 final String description = \_descriptionController.text.trim();  
  
 if (name.isEmpty || description.isEmpty || \_startTime == null || \_endTime == null) {  
 ScaffoldMessenger.of(context).showSnackBar(  
 const SnackBar(  
 content: Text('Please fill all fields.'),  
 ),  
 );  
 return;  
 }  
  
 final newSlot = SlotEntity(  
 id: UniqueKey().toString(),  
 name: name,  
 startTime: \_startTime!,  
 endTime: \_endTime!,  
 description: description,  
 inchargeId: widget.uid,  
 isApproved: false,  
 );  
  
 BlocProvider.of<SlotCubit>(context).bookSlotUsecase(newSlot);  
 Future.delayed(  
 Duration(seconds: 1),  
 () {  
 ScaffoldMessenger.of(context).showSnackBar(  
 const SnackBar(  
 content: Text('Slot booked successfully.'),  
 ),  
 );  
 },  
 );  
 }  
}

**home\_page.dart**

import 'package:flutter\_bloc/flutter\_bloc.dart';  
import 'package:hall\_sync/features/presentation/cubit/auth/auth\_cubit.dart';  
import 'package:hall\_sync/features/presentation/cubit/slot/slot\_cubit.dart';  
import 'package:hall\_sync/features/presentation/pages/book\_events\_page.dart';  
import 'package:hall\_sync/features/presentation/pages/view\_events\_page.dart';  
import 'package:hall\_sync/features/presentation/pages/view\_my\_events\_page.dart';  
import 'package:hall\_sync/features/presentation/widgets/bottom\_navigation\_bar\_widget.dart';  
import 'package:flutter/material.dart';  
  
class HomePage extends StatefulWidget {  
 final String uid;  
 const HomePage({Key? key, required this.uid}) : super(key: key);  
  
 @override  
 State<HomePage> createState() => \_HomePageState();  
}  
  
class \_HomePageState extends State<HomePage> {  
  
 GlobalKey<ScaffoldState> \_scaffoldKey = GlobalKey<ScaffoldState>();  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 key: \_scaffoldKey,  
 body: BottomNavBar(  
 navItems: [  
 NavigationItem(  
 widget: BookEventsPage(uid: widget.uid),   
 icon: Icons.book,   
 selectedIcon: Icons.book\_outlined,   
 label: "Book Events"  
 ),  
  
 NavigationItem(  
 widget: ViewEventsPage(uid: widget.uid,),   
 icon: Icons.event,   
 selectedIcon: Icons.event\_outlined,   
 label: "Events"  
 ),  
  
 NavigationItem(  
 widget: ViewMyEventsPage(uid: widget.uid),   
 icon: Icons.person,   
 selectedIcon: Icons.person\_outlined,   
 label: "My Events"  
 ),  
 ],  
 ),  
 );  
 }  
}

**sign\_in\_page.dart**

import 'package:flutter/material.dart';  
import 'package:flutter\_bloc/flutter\_bloc.dart';  
import 'package:hall\_sync/config/utils/constants.dart';  
import 'package:hall\_sync/features/domain/entities/user\_entity.dart';  
import 'package:hall\_sync/features/presentation/cubit/auth/auth\_cubit.dart';  
import 'package:hall\_sync/features/presentation/cubit/user/user\_cubit.dart';  
import 'package:hall\_sync/features/presentation/widgets/common.dart';  
  
import 'home\_page.dart';  
  
class SignInPage extends StatefulWidget {  
 const SignInPage({Key? key}) : super(key: key);  
  
 @override  
 \_SignInPageState createState() => \_SignInPageState();  
}  
  
class \_SignInPageState extends State<SignInPage> {  
  
  
 TextEditingController \_emailController = TextEditingController();  
 TextEditingController \_passwordController = TextEditingController();  
  
 GlobalKey<ScaffoldState> \_scaffoldGlobalKey = GlobalKey<ScaffoldState>();  
  
  
 @override  
 void dispose() {  
 \_emailController.dispose();  
 \_passwordController.dispose();  
 super.dispose();  
 }  
  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 key: \_scaffoldGlobalKey,  
 body: BlocConsumer<UserCubit,UserState>(  
 builder: (context,userState){  
  
 if (userState is UserSuccess){  
 return BlocBuilder<AuthCubit,AuthState>(builder:(context,authState){  
  
 if (authState is Authenticated){  
 return HomePage(uid: authState.uid,);  
 }else{  
 return \_bodyWidget();  
 }  
 });  
 }  
  
 return \_bodyWidget();  
 },  
 listener: (context,userState){  
 if (userState is UserSuccess){  
 BlocProvider.of<AuthCubit>(context).loggedIn();  
 }  
 if (userState is UserFailure){  
 snackBarError(msg: "invalid email",scaffoldState: \_scaffoldGlobalKey);  
 }  
 },  
 )  
 );  
 }  
  
 \_bodyWidget() {  
   
 return Container(  
 padding: EdgeInsets.all(25),  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: [  
 SizedBox(  
 height: 30,  
 ),  
 Container(  
 height: 50,  
 padding: EdgeInsets.symmetric(horizontal: 10),  
 decoration: BoxDecoration(  
 color: Colors.black.withOpacity(.1),  
 borderRadius: BorderRadius.all(Radius.circular(10)),  
 ),  
 child: TextField(  
 controller: \_emailController,  
 decoration: InputDecoration(  
 hintText: 'Enter your email', border: InputBorder.none),  
 ),  
 ),  
 SizedBox(  
 height: 10,  
 ),  
 Container(  
 height: 50,  
 padding: EdgeInsets.symmetric(horizontal: 10),  
 decoration: BoxDecoration(  
 color: Colors.black.withOpacity(.1),  
 borderRadius: BorderRadius.all(Radius.circular(10)),  
 ),  
 child: TextField(  
 controller: \_passwordController,  
 obscureText: true,  
 decoration: InputDecoration(  
 hintText: 'Enter your Password', border: InputBorder.none),  
 ),  
 ),  
 SizedBox(  
 height: 20,  
 ),  
 GestureDetector(  
 onTap: () {  
 submitSignIn();  
 },  
 child: Container(  
 height: 45,  
 alignment: Alignment.center,  
 width: MediaQuery  
 .of(context)  
 .size  
 .width / 2,  
 decoration: BoxDecoration(  
 color: Colors.deepOrange.withOpacity(.8),  
 borderRadius: BorderRadius.all(  
 Radius.circular(10),  
 ),  
 ),  
 child: Text(  
 "Login",  
 style: TextStyle(fontSize: 18, fontWeight: FontWeight.w600),  
 ),  
 ),  
 ),  
 SizedBox(  
 height: 10,  
 ),  
 GestureDetector(  
 onTap: () {  
 Navigator.pushNamedAndRemoveUntil(context, PageConstants.signupPage, (route) => false);  
 },  
 child: Container(  
 height: 45,  
 alignment: Alignment.center,  
 width: MediaQuery  
 .of(context)  
 .size  
 .width / 2,  
 decoration: BoxDecoration(  
 color: Colors.grey.withOpacity(.8),  
 borderRadius: BorderRadius.all(  
 Radius.circular(10),  
 ),  
 ),  
 child: Text(  
 "Sign Up",  
 style: TextStyle(fontSize: 18, fontWeight: FontWeight.w600),  
 ),  
 ),  
 ),  
 ],  
 ),  
 );  
 }  
  
 void submitSignIn() {  
 if (\_emailController.text.isNotEmpty &&  
 \_passwordController.text.isNotEmpty) {  
 BlocProvider.of<UserCubit>(context).handleSignIn(user: UserEntity(  
 email: \_emailController.text,  
 password: \_passwordController.text,  
 ));  
 }  
 }  
}

**sign\_up\_page.dart**

import 'package:flutter/material.dart';  
import 'package:flutter\_bloc/flutter\_bloc.dart';  
import 'package:hall\_sync/config/utils/constants.dart';  
import 'package:hall\_sync/features/domain/entities/user\_entity.dart';  
import 'package:hall\_sync/features/presentation/cubit/auth/auth\_cubit.dart';  
import 'package:hall\_sync/features/presentation/cubit/user/user\_cubit.dart';  
import 'package:hall\_sync/features/presentation/pages/admin\_home\_page.dart';  
import 'package:hall\_sync/features/presentation/widgets/common.dart';  
  
import 'home\_page.dart';  
  
class SignUpPage extends StatefulWidget {  
 const SignUpPage({Key? key}) : super(key: key);  
  
 @override  
 \_SignUpPageState createState() => \_SignUpPageState();  
}  
  
class \_SignUpPageState extends State<SignUpPage> {  
  
  
 TextEditingController \_usernameController = TextEditingController();  
 TextEditingController \_emailController = TextEditingController();  
 TextEditingController \_passwordController = TextEditingController();  
  
 GlobalKey<ScaffoldState> \_globalKey = GlobalKey<ScaffoldState>();  
  
  
  
 @override  
 void dispose() {  
 \_usernameController.dispose();  
 \_emailController.dispose();  
 \_passwordController.dispose();  
 super.dispose();  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 key: \_globalKey,  
 body: BlocConsumer<UserCubit,UserState>(  
 builder: (context,userState){  
  
 if (userState is UserSuccess){  
 return BlocBuilder<AuthCubit,AuthState>(builder:(context,authState){  
  
 if (authState is Authenticated){  
 if(authState.role=='admin'){  
 return AdminHomePage(uid: authState.uid,);  
 }  
 else{  
 return HomePage(uid: authState.uid);  
 }  
 }else{  
 return \_bodyWidget();  
 }  
 });  
 }  
  
 return \_bodyWidget();  
 },  
 listener: (context,userState){  
 if (userState is UserSuccess){  
 BlocProvider.of<AuthCubit>(context).loggedIn();  
 }  
 if (userState is UserFailure){  
 snackBarError(msg: "invalid email",scaffoldState: \_globalKey);  
 }  
 },  
 )  
 );  
 }  
  
 \_bodyWidget() {  
 return Container(  
 padding: EdgeInsets.all(25),  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: [  
 SizedBox(  
 height: 30,  
 ),  
 GestureDetector(  
 onTap: () {  
 Navigator.pushNamedAndRemoveUntil(context, PageConstants.signinPage, (route) => false);  
 },  
 child: Container(  
 height: 50,  
 width: 50,  
 alignment: Alignment.center,  
 decoration: BoxDecoration(  
 color: Colors.white,  
 border: Border.all(color: Colors.black.withOpacity(.6)),  
 shape: BoxShape.circle,  
 ),  
 child: Icon(Icons.arrow\_back\_ios),  
 ),  
 ),  
 SizedBox(  
 height: 30,  
 ),  
 Container(  
 height: 50,  
 padding: EdgeInsets.symmetric(horizontal: 10),  
 decoration: BoxDecoration(  
 color: Colors.black.withOpacity(.1),  
 borderRadius: BorderRadius.all(Radius.circular(10)),  
 ),  
 child: TextField(  
 controller: \_usernameController,  
 decoration: InputDecoration(  
 hintText: 'Username', border: InputBorder.none),  
 ),  
 ),  
 SizedBox(  
 height: 10,  
 ),  
 Container(  
 height: 50,  
 padding: EdgeInsets.symmetric(horizontal: 10),  
 decoration: BoxDecoration(  
 color: Colors.black.withOpacity(.1),  
 borderRadius: BorderRadius.all(Radius.circular(10)),  
 ),  
 child: TextField(  
 controller: \_emailController,  
 decoration: InputDecoration(  
 hintText: 'Enter your email', border: InputBorder.none),  
 ),  
 ),  
 SizedBox(  
 height: 10,  
 ),  
 Container(  
 height: 50,  
 padding: EdgeInsets.symmetric(horizontal: 10),  
 decoration: BoxDecoration(  
 color: Colors.black.withOpacity(.1),  
 borderRadius: BorderRadius.all(Radius.circular(10)),  
 ),  
 child: TextField(  
 obscureText: true,  
 controller: \_passwordController,  
 decoration: InputDecoration(  
 hintText: 'Enter your Password', border: InputBorder.none),  
 ),  
 ),  
 SizedBox(  
 height: 15,  
 ),  
 GestureDetector(  
 onTap: (){  
 submitSignIn();  
 },  
 child: Container(  
 height: 45,  
 alignment: Alignment.center,  
 width: MediaQuery.of(context).size.width / 2,  
 decoration: BoxDecoration(  
 color: Colors.deepOrange.withOpacity(.8),  
 borderRadius: BorderRadius.all(  
 Radius.circular(10),  
 ),  
 ),  
 child: Text(  
 "Create New Account",  
 style: TextStyle(fontSize: 18, fontWeight: FontWeight.w600),  
 ),  
 ),  
 ),  
 SizedBox(  
 height: 10,  
 ),  
 ],  
 ),  
 );  
 }  
  
 void submitSignIn() {  
 if (\_usernameController.text.isNotEmpty &&\_emailController.text.isNotEmpty &&  
 \_passwordController.text.isNotEmpty) {  
 BlocProvider.of<UserCubit>(context).handleSignup(user: UserEntity(  
 name: \_usernameController.text,  
 email: \_emailController.text,  
 password: \_passwordController.text,  
 type: "viewer"  
 ));  
 }  
 }  
}

**update\_slot\_page.dart**

import 'package:flutter/material.dart';  
import 'package:flutter\_bloc/flutter\_bloc.dart';  
import 'package:hall\_sync/config/utils/constants.dart';  
import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:hall\_sync/features/presentation/cubit/auth/auth\_cubit.dart';  
import 'package:hall\_sync/features/presentation/cubit/slot/slot\_cubit.dart';  
import 'package:hall\_sync/features/presentation/widgets/common.dart';  
import 'package:intl/intl.dart';  
  
class UpdateSlotPage extends StatefulWidget {  
 final SlotEntity slot;  
 const UpdateSlotPage({Key? key, required this.slot}) : super(key: key);  
  
 @override  
 \_UpdateSlotPageState createState() => \_UpdateSlotPageState();  
}  
  
class \_UpdateSlotPageState extends State<UpdateSlotPage> {  
 late TextEditingController \_nameController;  
 late TextEditingController \_descriptionController;  
 late DateTime \_startTime;  
 late DateTime \_endTime;  
  
 @override  
 void initState() {  
 \_nameController = TextEditingController(text: widget.slot.name);  
 \_descriptionController = TextEditingController(text: widget.slot.description);  
 \_startTime = widget.slot.startTime!;  
 \_endTime = widget.slot.endTime!;  
  
 super.initState();  
 }  
  
 @override  
 void dispose() {  
 \_nameController.dispose();  
 \_descriptionController.dispose();  
 super.dispose();  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: const Text('Update Slot'),  
 actions: [  
 IconButton(  
 onPressed: () {  
 BlocProvider.of<AuthCubit>(context).loggedOut();  
 },  
 icon: const Icon(Icons.logout),  
 )  
 ],  
 ),  
 body: Padding(  
 padding: const EdgeInsets.all(16.0),  
 child: Column(  
 crossAxisAlignment: CrossAxisAlignment.start,  
 children: [  
 TextFormField(  
 controller: \_nameController,  
 decoration: InputDecoration(labelText: 'Name'),  
 ),  
 const SizedBox(height: 16),  
 TextFormField(  
 controller: \_descriptionController,  
 decoration: InputDecoration(labelText: 'Description'),  
 ),  
 const SizedBox(height: 16),  
 TextFormField(  
 onTap: () async {  
 DateTime? selectedTime = await showDatePicker(  
 context: context,  
 initialDate: \_startTime,  
 firstDate: DateTime.now(),  
 lastDate: DateTime.now().add(Duration(days: 365)),  
 );  
 if (selectedTime != null) {  
 setState(() {  
 \_startTime = selectedTime;  
 });  
 }  
 },  
 readOnly: true,  
 decoration: InputDecoration(  
 labelText: 'Start Time',  
 suffixIcon: Icon(Icons.calendar\_today),  
 ),  
 controller: TextEditingController(  
 text: \_startTime != null  
 ? DateFormat.yMd().add\_jm().format(\_startTime)  
 : '',  
 ),  
 ),  
 const SizedBox(height: 16),  
 TextFormField(  
 onTap: () async {  
 DateTime? selectedTime = await showDatePicker(  
 context: context,  
 initialDate: \_endTime,  
 firstDate: DateTime.now(),  
 lastDate: DateTime.now().add(Duration(days: 365)),  
 );  
 if (selectedTime != null) {  
 setState(() {  
 \_endTime = selectedTime;  
 });  
 }  
 },  
 readOnly: true,  
 decoration: InputDecoration(  
 labelText: 'End Time',  
 suffixIcon: Icon(Icons.calendar\_today),  
 ),  
 controller: TextEditingController(  
 text: \_endTime != null  
 ? DateFormat.yMd().add\_jm().format(\_endTime)  
 : '',  
 ),  
 ),  
 const SizedBox(height: 16),  
 ElevatedButton(  
 onPressed: () => \_updateSlot(context),  
 child: const Text('Update'),  
 ),  
 ],  
 ),  
 ),  
 );  
 }  
  
 void \_updateSlot(BuildContext context) {  
 final String name = \_nameController.text.trim();  
 final String description = \_descriptionController.text.trim();  
  
 if (name.isEmpty || description.isEmpty || \_startTime == null || \_endTime == null) {  
 ScaffoldMessenger.of(context).showSnackBar(  
 const SnackBar(  
 content: Text('Please fill all fields.'),  
 ),  
 );  
 return;  
 }  
  
 final updatedSlot = SlotEntity(  
 id: widget.slot.id,  
 name: name,  
 startTime: \_startTime,  
 endTime: \_endTime,  
 description: description,  
 inchargeId: widget.slot.inchargeId,  
 isApproved: widget.slot.isApproved,  
 );  
  
 BlocProvider.of<SlotCubit>(context).updateSlotUsecase(updatedSlot);  
 Future.delayed(Duration(seconds: 2),(){  
 Navigator.pop(context);  
 });  
 }  
}

**view\_events\_page.dart**

import 'package:flutter/material.dart';  
import 'package:flutter\_bloc/flutter\_bloc.dart';  
import 'package:hall\_sync/config/utils/constants.dart';  
import 'package:hall\_sync/features/domain/entities/slot\_entity.dart';  
import 'package:hall\_sync/features/presentation/cubit/auth/auth\_cubit.dart';  
import 'package:hall\_sync/features/presentation/cubit/slot/slot\_cubit.dart';  
import 'package:intl/intl.dart';  
  
  
class ViewEventsPage extends StatefulWidget {  
 final String uid;  
 const ViewEventsPage({Key? key,required this.uid}) : super(key: key);  
  
 @override  
 \_ViewEventsPageState createState() => \_ViewEventsPageState();  
}  
  
class \_ViewEventsPageState extends State<ViewEventsPage> {  
  
 @override  
 void initState() {  
 BlocProvider.of<SlotCubit>(context).getAllSlots();  
 super.initState();  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: Text(  
 "Events",  
 style: TextStyle(fontSize: 20, fontWeight: FontWeight.w600),  
 ),  
 actions: [  
 IconButton(  
 onPressed: () {  
 BlocProvider.of<AuthCubit>(context).loggedOut();  
 },  
 icon: Icon(Icons.exit\_to\_app)),  
 ],  
 ),  
 body: BlocBuilder<SlotCubit,SlotState>(  
 builder: (context,slotState){  
  
 if (slotState is SlotLoaded){  
 return \_bodyWidget(slotState,context);  
 }  
  
  
 return const Center(child: CircularProgressIndicator());  
 },  
 ),  
 );  
 }  
  
 Widget \_noSlotsWidget(){  
 return const Center(  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: [  
 SizedBox(  
 height: 10,  
 ),  
 Text("No events yet"),  
 ],  
 ),  
 );  
 }  
  
  
 Widget \_bodyWidget(SlotLoaded slotLoadedState, BuildContext ctx) {  
 List<SlotEntity> filteredSlots = slotLoadedState.slots;  
  
 List<SlotEntity> \_filterEventsByDate(List<SlotEntity> slots, DateTime startDate, DateTime endDate) {  
 return slots.where((slot) => slot.startTime!.isAfter(startDate) && slot.startTime!.isBefore(endDate)).toList();  
 }  
 Widget \_filterButton(String text, VoidCallback onPressed) {  
 return ElevatedButton(  
 onPressed: onPressed,  
 child: Text(text),  
 );  
 }  
  
 return Column(  
 children: [  
 Row(  
 mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
 children: [  
 \_filterButton("Today", () {  
 DateTime now = DateTime.now();  
 DateTime startOfToday = DateTime(now.year, now.month, now.day);   
 DateTime startOfTomorrow = startOfToday.add(Duration(days: 1));   
  
 setState(() {  
 filteredSlots = \_filterEventsByDate(slotLoadedState.slots, startOfToday, startOfTomorrow);  
 });  
 }),  
 \_filterButton("Tomorrow", () {  
 DateTime now = DateTime.now();  
 DateTime startOfTomorrow = DateTime(now.year, now.month, now.day + 1);  
 DateTime startOfNextDay = startOfTomorrow.add(const Duration(days: 1));   
  
 setState(() {  
 filteredSlots = \_filterEventsByDate(slotLoadedState.slots, startOfTomorrow, startOfNextDay);  
 });  
 }),  
  
 \_filterButton("This Week", () {  
 DateTime startOfWeek = DateTime.now().subtract(Duration(days: DateTime.now().weekday - 1));  
 DateTime endOfWeek = startOfWeek.add(const Duration(days: 7));  
 setState(() {  
 filteredSlots = \_filterEventsByDate(slotLoadedState.slots, startOfWeek, endOfWeek);  
 });  
 }),  
 \_filterButton("This Month", () {  
 DateTime startOfMonth = DateTime(DateTime.now().year, DateTime.now().month, 1);  
 DateTime endOfMonth = DateTime(DateTime.now().year, DateTime.now().month + 1, 0);  
 setState(() {  
 filteredSlots = \_filterEventsByDate(slotLoadedState.slots, startOfMonth, endOfMonth);  
 });  
 }),  
 \_filterButton("All", () {  
 setState(() {  
 filteredSlots = slotLoadedState.slots;  
 });  
 }),  
 ],  
 ),  
 Expanded(  
 child: filteredSlots.isEmpty  
 ? \_noSlotsWidget()  
 : SingleChildScrollView(  
 child: GridView.builder(  
 itemCount: filteredSlots.length,  
 gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(  
 crossAxisCount: 2,  
 childAspectRatio: 1.2,  
 ),  
 itemBuilder: (\_, index) {  
 return GestureDetector(  
 child: Container(  
 decoration: BoxDecoration(  
 color: Colors.white,  
 borderRadius: BorderRadius.circular(8),  
 boxShadow: [  
 BoxShadow(  
 color: Colors.black.withOpacity(.2),  
 blurRadius: 2,  
 spreadRadius: 2,  
 offset: Offset(0, 1.5),  
 ),  
 ],  
 ),  
 padding: EdgeInsets.all(10),  
 margin: EdgeInsets.all(6),  
 child: Column(  
 crossAxisAlignment: CrossAxisAlignment.start,  
 mainAxisAlignment: MainAxisAlignment.spaceBetween,  
 children: [  
 Text(  
 "${filteredSlots[index].description}",  
 maxLines: 6,  
 overflow: TextOverflow.ellipsis,  
 ),  
 const SizedBox(height: 4),  
 Text(  
 "${DateFormat("dd MMM yyy hh:mm a").format(filteredSlots[index].startTime!)}",  
 ),  
 ],  
 ),  
 ),  
 );  
 },  
 ),  
 ),  
 ),  
 ],  
 );  
 }  
}

**view\_my\_events\_page.dart**

import 'package:flutter/material.dart';  
import 'package:flutter\_bloc/flutter\_bloc.dart';  
import 'package:hall\_sync/config/utils/constants.dart';  
import 'package:hall\_sync/features/presentation/cubit/auth/auth\_cubit.dart';  
import 'package:hall\_sync/features/presentation/cubit/slot/slot\_cubit.dart';  
import 'package:intl/intl.dart';  
  
  
class ViewMyEventsPage extends StatefulWidget {  
 final String uid;  
 const ViewMyEventsPage({Key? key,required this.uid}) : super(key: key);  
  
 @override  
 \_ViewMyEventsPageState createState() => \_ViewMyEventsPageState();  
}  
  
class \_ViewMyEventsPageState extends State<ViewMyEventsPage> {  
  
  
 @override  
 void initState() {  
 BlocProvider.of<SlotCubit>(context).getUserSlots(uid: widget.uid);  
 super.initState();  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: Text(  
 "Events",  
 style: TextStyle(fontSize: 20, fontWeight: FontWeight.w600),  
 ),  
 actions: [  
 IconButton(  
 onPressed: () {  
 BlocProvider.of<AuthCubit>(context).loggedOut();  
 },  
 icon: Icon(Icons.exit\_to\_app)),  
 ],  
 ),  
 body: BlocBuilder<SlotCubit,SlotState>(  
 builder: (context,slotState){  
  
 if (slotState is SlotLoaded){  
 return \_bodyWidget(slotState,context);  
 }  
  
  
 return const Center(child: CircularProgressIndicator());  
 },  
 ),  
 );  
 }  
  
 Widget \_noSlotsWidget(){  
 return const Center(  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: [  
 SizedBox(  
 height: 10,  
 ),  
 Text("No events yet"),  
 ],  
 ),  
 );  
 }  
  
 Widget \_bodyWidget(SlotLoaded slotLoadedState, BuildContext ctx) {  
 return Column(  
 children: [  
 Expanded(  
 child: slotLoadedState.slots.isEmpty?\_noSlotsWidget():GridView.builder(  
 itemCount: slotLoadedState.slots.length,  
 gridDelegate:  
 SliverGridDelegateWithFixedCrossAxisCount(  
 crossAxisCount: 2, childAspectRatio: 1.2),  
 itemBuilder: (\_, index) {  
 return GestureDetector(  
 onTap: () {  
 Navigator.pushNamed(  
 context, PageConstants.updateSlotPage,  
 arguments: slotLoadedState.slots[index]);  
 },  
 onLongPress: () {  
 showDialog(  
 context: context,  
 builder: (BuildContext context) {  
 return AlertDialog(  
 title: Text("Delete Note"),  
 content: Text("are you sure you want to delete this note."),  
 actions: [  
 TextButton(  
 child: Text("Delete"),  
 onPressed: () {  
 BlocProvider.of<SlotCubit>(context).deleteSlot(slot: slotLoadedState.slots[index]);  
 Navigator.pop(context);  
   
 Future.delayed(Duration(seconds: 2), (){  
 BlocProvider.of<SlotCubit>(ctx).getAllSlots();  
 });   
 },  
 ),  
 TextButton(  
 child: Text("No"),  
 onPressed: () {  
 Navigator.pop(context);  
 },  
 ),  
 ],  
 );  
 },  
 );  
 },  
 child: Container(  
 decoration: BoxDecoration(  
 color: Colors.white,  
 borderRadius: BorderRadius.circular(8),  
 boxShadow: [  
 BoxShadow(  
 color: Colors.black.withOpacity(.2),  
 blurRadius: 2,  
 spreadRadius: 2,  
 offset: Offset(0, 1.5))  
 ]),  
 padding: EdgeInsets.all(10),  
 margin: EdgeInsets.all(6),  
 child: Column(  
 crossAxisAlignment:  
 CrossAxisAlignment.start,  
 mainAxisAlignment:  
 MainAxisAlignment.spaceBetween,  
 children: [  
 Text(  
 "${slotLoadedState.slots[index].description}",  
 maxLines: 6,  
 overflow: TextOverflow.ellipsis,  
 ),  
 const SizedBox(  
 height: 4,  
 ),  
 Text(  
 "${DateFormat("dd MMM yyy hh:mm a").format(slotLoadedState.slots[index].startTime!)}")  
 ],  
 ),  
 ),  
 );  
 },  
 ),  
 ),  
 ],  
 );  
 }  
}

**bottom\_navigation\_bar\_widget.dart**

import 'package:flutter/material.dart';  
import 'package:flutter/widgets.dart';  
  
class NavigationItem{  
 final Widget widget;  
 final String label;  
 final IconData icon;  
 final IconData selectedIcon;  
 const NavigationItem({required this.widget, required this.label, required this.icon, required this.selectedIcon});  
}  
  
class BottomNavBar extends StatefulWidget {  
 final List<NavigationItem> navItems;  
 const BottomNavBar({super.key, required this.navItems});  
  
 @override  
 State<BottomNavBar> createState() => \_BottomNavBarState();  
}  
  
class \_BottomNavBarState extends State<BottomNavBar> {  
 int currentPageIndex = 0;  
  
 @override  
 Widget build(BuildContext context) {  
  
 return Scaffold(  
 bottomNavigationBar: NavigationBar(  
 onDestinationSelected: (int index) {  
 setState(() {  
 currentPageIndex = index;  
 });  
 },  
 indicatorColor: Colors.amber,  
 selectedIndex: currentPageIndex,  
 destinations: widget.navItems.map((navItem) =>   
 NavigationDestination(  
 selectedIcon: Icon(navItem.icon),  
 icon: Icon(navItem.icon),  
 label: navItem.label,  
 )  
 ).toList()  
 ),  
 body: widget.navItems[currentPageIndex].widget,  
 );  
 }  
}

**common.dart**

import 'package:flutter/material.dart';  
  
void snackBarError({String? msg, required GlobalKey<ScaffoldState> scaffoldState}) {  
   
 ScaffoldMessenger.of(scaffoldState.currentState!.context).showSnackBar(  
 SnackBar(  
 backgroundColor: Colors.red,  
 duration: Duration(seconds: 3),  
 content: Row(  
 mainAxisAlignment: MainAxisAlignment.spaceBetween,  
 children: [  
 Text("$msg"),  
 const Icon(Icons.error)  
 ],  
 ),  
 ),  
 );  
}