

<b>Experiment</b>	9&10
<b>Aim</b>	Capstone project with database connectivity
<b>Objective</b>	Build an application in flutter with database connectivity
<b>Name</b>	Vedant Onkar
<b>UCID</b>	2024510036
<b>Class</b>	FYMCA
<b>Batch</b>	B
<b>Date of Submission</b>	06/05/2025

<b>Technology used</b>	Flutter
<b>Task</b>	Build a flight booking management system in flutter with database connectivity
<b>Code with proper label</b>	<pre> import 'package:flutter/material.dart'; import 'package:provider/provider.dart'; import 'package:intl/intl.dart';  import 'package:google_fonts/google_fonts.dart';  import 'package:postgres/postgres.dart';  class DatabaseService {   static final DatabaseService _instance = DatabaseService._internal();   factory DatabaseService() =&gt; _instance;    late PostgreSQLConnection _connection;   bool _isConnected = false;    DatabaseService._internal();    Future&lt;void&gt; connect() async {     try {       _connection = PostgreSQLConnection(         '10.0.2.2', // For Android emulator         // 'localhost', // For physical device or web         5432,         'skyboard',         username: 'postgres',         password: 'root',       );        await _connection.open();       _isConnected = true; </pre>

```

print('Connected to PostgreSQL');

// Create tables if they don't exist
await _initializeDatabase();
} catch (e) {
    print('Failed to connect to PostgreSQL: $e');
    rethrow;
}
}

Future<void> _initializeDatabase() async {
    // Create users table
    await _connection.execute("""
CREATE TABLE IF NOT EXISTS users (
    id TEXT PRIMARY KEY,
    name TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,
    password TEXT NOT NULL,
    role TEXT NOT NULL
)
""");

    // Create flights table
    await _connection.execute("""
CREATE TABLE IF NOT EXISTS flights (
    id TEXT PRIMARY KEY,
    airline TEXT NOT NULL,
    flight_number TEXT NOT NULL,
    source TEXT NOT NULL,
    destination TEXT NOT NULL,
    departure_time TIMESTAMP NOT NULL,
    arrival_time TIMESTAMP NOT NULL,
    price DECIMAL NOT NULL,
    seats_available INTEGER NOT NULL
)
""");

    // Create bookings table
    await _connection.execute("""
CREATE TABLE IF NOT EXISTS bookings (
    id TEXT PRIMARY KEY,
    user_id TEXT NOT NULL,
    flight_id TEXT NOT NULL,
    passenger_name TEXT NOT NULL,
    passenger_age INTEGER NOT NULL,
    seat_number TEXT NOT NULL,
    booking_time TIMESTAMP NOT NULL,

```

```

status TEXT NOT NULL,
FOREIGN KEY (user_id) REFERENCES users (id),
FOREIGN KEY (flight_id) REFERENCES flights (id)
)
""");

// Create default admin user if not exists
final adminExists = await _connection.query(
  'SELECT COUNT(*) FROM users WHERE email = @email',
  substitutionValues: {'email': 'admin@example.com'},
);

if ((adminExists.first.first as int) == 0) {
  await _connection.execute("""
  INSERT INTO users (id, name, email, password, role)
  VALUES ('a1', 'Admin User', 'admin@example.com', 'admin123', 'admin')
  """);
}

// Add sample flights if the table is empty
final flightsCount =
  await _connection.query('SELECT COUNT(*) FROM flights');
if ((flightsCount.first.first as int) == 0) {
  await _addSampleFlights();
}
}

Future<void> _addSampleFlights() async {
  final now = DateTime.now();

  await _connection.execute("""
  INSERT INTO flights (id, airline, flight_number, source, destination,
departure_time, arrival_time, price, seats_available)
  VALUES
  ('1', 'Air India', 'AI101', 'Mumbai', 'Delhi', @dep1, @arr1, 4500, 45),
  ('2', 'IndiGo', 'IN202', 'Pune', 'Bangalore', @dep2, @arr2, 3200, 32),
  ('3', 'Vistara', 'VS303', 'Hyderabad', 'Chennai', @dep3, @arr3, 4000, 20),
  ('4', 'SpiceJet', 'SJ404', 'Mumbai', 'Goa', @dep4, @arr4, 2800, 15)
  """, substitutionValues: {
    'dep1': now.add(const Duration(days: 2, hours: 10)),
    'arr1': now.add(const Duration(days: 2, hours: 12)),
    'dep2': now.add(const Duration(days: 3, hours: 8)),
    'arr2': now.add(const Duration(days: 3, hours: 10)),
    'dep3': now.add(const Duration(days: 4, hours: 14)),
    'arr3': now.add(const Duration(days: 4, hours: 16)),
    'dep4': now.add(const Duration(days: 5, hours: 7)),
    'arr4': now.add(const Duration(days: 5, hours: 8, minutes: 30)),
  });
}

```

```

    });
}

Future<void> close() async {
  if (_isConnected) {
    await _connection.close();
    _isConnected = false;
  }
}

// User Methods
Future<List<Map<String, dynamic>>> getUsers() async {
  final results = await _connection.query('SELECT * FROM users');
  return results
    .map((row) => {
      'id': row[0],
      'name': row[1],
      'email': row[2],
      'password': row[3],
      'role': row[4],
    })
    .toList();
}

Future<Map<String, dynamic>?> getUserByEmailAndPassword(
  String email, String password) async {
  final results = await _connection.query(
    'SELECT * FROM users WHERE email = @email AND password = @password',
    substitutionValues: {'email': email, 'password': password},
  );

  if (results.isEmpty) return null;

  final row = results.first;
  return {
    'id': row[0],
    'name': row[1],
    'email': row[2],
    'password': row[3],
    'role': row[4],
  };
}

Future<void> createUser(Map<String, dynamic> user) async {
  await _connection.execute(
    INSERT INTO users (id, name, email, password, role)

```

```

VALUES (@id, @name, @email, @password, @role)
'', substitutionValues: {
    'id': user['id'],
    'name': user['name'],
    'email': user['email'],
    'password': user['password'],
    'role': user['role'],
});
}

// Flight Methods
Future<List<Map<String, dynamic>>> getFlights() async {
    final results = await _connection.query('SELECT * FROM flights');
    return results
        .map((row) => {
            'id': row[0],
            'airline': row[1],
            'flightNumber': row[2],
            'source': row[3],
            'destination': row[4],
            'departureTime': row[5],
            'arrivalTime': row[6],
            'price': row[7],
            'seatsAvailable': row[8],
        })
        .toList();
}

Future<Map<String, dynamic>?> getFlightById(String id) async {
    final results = await _connection.query(
        'SELECT * FROM flights WHERE id = @id',
        substitutionValues: {'id': id},
    );

    if (results.isEmpty) return null;

    final row = results.first;
    return {
        'id': row[0],
        'airline': row[1],
        'flightNumber': row[2],
        'source': row[3],
        'destination': row[4],
        'departureTime': row[5],
        'arrivalTime': row[6],
        'price': row[7],
        'seatsAvailable': row[8],
    };
}

```

```

    };
}

Future<List<Map<String, dynamic>>> searchFlights({
    required String source,
    required String destination,
    required DateTime date,
}) async {
    final results = await _connection.query("""
    SELECT * FROM flights
    WHERE source = @source
    AND destination = @destination
    AND DATE(departure_time) = @date
    """, substitutionValues: {
        'source': source,
        'destination': destination,
        'date': DateTime(date.year, date.month, date.day),
    });

    return results
        .map((row) => {
            'id': row[0],
            'airline': row[1],
            'flightNumber': row[2],
            'source': row[3],
            'destination': row[4],
            'departureTime': row[5],
            'arrivalTime': row[6],
            'price': row[7],
            'seatsAvailable': row[8],
        })
        .toList();
}

Future<void> createFlight(Map<String, dynamic> flight) async {
    await _connection.execute("""
    INSERT INTO flights (id, airline, flight_number, source, destination,
    departure_time, arrival_time, price, seats_available)
    VALUES (@id, @airline, @flightNumber, @source, @destination,
    @departureTime, @arrivalTime, @price, @seatsAvailable)
    """, substitutionValues: {
        'id': flight['id'],
        'airline': flight['airline'],
        'flightNumber': flight['flightNumber'],
        'source': flight['source'],
        'destination': flight['destination'],
        'departureTime': flight['departureTime'],
    });
}

```

```

        'arrivalTime': flight['arrivalTime'],
        'price': flight['price'],
        'seatsAvailable': flight['seatsAvailable'],
    });
}

Future<void> deleteFlight(String id) async {
    await _connection.execute(
        'DELETE FROM flights WHERE id = @id',
        substitutionValues: {'id': id},
    );
}

Future<List<Map<String, dynamic>>> getUserBookings(String userId) async {
    final results = await _connection.query(
        'SELECT * FROM bookings WHERE user_id = @userId',
        substitutionValues: {'userId': userId},
    );

    return results
        .map((row) => {
            'id': row[0],
            'userId': row[1],
            'flightId': row[2],
            'passengerName': row[3],
            'passengerAge': row[4],
            'seatNumber': row[5],
            'bookingTime': row[6],
            'status': row[7],
        })
        .toList();
}

Future<void> updateFlight(Map<String, dynamic> flight) async {
    await _connection.execute(
        'UPDATE flights
        SET airline = @airline,
            flight_number = @flightNumber,
            source = @source,
            destination = @destination,
            departure_time = @departureTime,
            arrival_time = @arrivalTime,
            price = @price,
            seats_available = @seatsAvailable
        WHERE id = @id',
        substitutionValues: {
            'id': flight['id'],

```

```

        'airline': flight['airline'],
        'flightNumber': flight['flightNumber'],
        'source': flight['source'],
        'destination': flight['destination'],
        'departureTime': flight['departureTime'],
        'arrivalTime': flight['arrivalTime'],
        'price': flight['price'],
        'seatsAvailable': flight['seatsAvailable'],
    });
}

// Future<void> deleteFlight(String id) async {
//   await _connection.execute(
//     'DELETE FROM flights WHERE id = @id',
//     substitutionValues: {'id': id},
//   );
// }

// Booking Methods
Future<List<Map<String, dynamic>>> getBookings() async {
  final results = await _connection.query('SELECT * FROM bookings');
  return results
    .map((row) => {
      'id': row[0],
      'userId': row[1],
      'flightId': row[2],
      'passengerName': row[3],
      'passengerAge': row[4],
      'seatNumber': row[5],
      'bookingTime': row[6],
      'status': row[7],
    })
    .toList();
}

// Future<List<Map<String, dynamic>>> getUserBookings(String userId) async
{
  // final results = await _connection.query(
  //   'SELECT * FROM bookings WHERE user_id = @userId',
  //   substitutionValues: {'userId': userId},
  // );

  // return results.map((row) => {
  //   'id': row[0],
  //   'userId': row[1],
  //   'flightId': row[2],
  //   'passengerName': row[3],

```



```

// 'passengerAge': row[4],
// 'seatNumber': row[5],
// 'bookingTime': row[6],
// 'status': row[7],
// }).toList();
// }

Future<void> createBooking(Map<String, dynamic> booking) async {
  await _connection.execute("""
    INSERT INTO bookings (id, user_id, flight_id, passenger_name,
passenger_age, seat_number, booking_time, status)
    VALUES (@id, @userId, @flightId, @passengerName, @passengerAge,
@seatNumber, @bookingTime, @status)
    """, substitutionValues: {
      'id': booking['id'],
      'userId': booking['userId'],
      'flightId': booking['flightId'],
      'passengerName': booking['passengerName'],
      'passengerAge': booking['passengerAge'],
      'seatNumber': booking['seatNumber'],
      'bookingTime': booking['bookingTime'],
      'status': booking['status'],
    });

  // Update flight seats
  await _connection.execute("""
    UPDATE flights
    SET seats_available = seats_available - 1
    WHERE id = @flightId
    """, substitutionValues: {'flightId': booking['flightId']});
}

Future<void> cancelBooking(String bookingId) async {
  // Get flight ID from booking
  final booking = await _connection.query(
    'SELECT flight_id FROM bookings WHERE id = @id',
    substitutionValues: {'id': bookingId},
  );

  final flightId = booking.first[0] as String;

  // Update booking status
  await _connection.execute("""
    UPDATE bookings
    SET status = 'cancelled'
    WHERE id = @id
    """, substitutionValues: {'id': bookingId});
}

```

```

// Update flight seats
await _connection.execute("""
UPDATE flights
SET seats_available = seats_available + 1
WHERE id = @flightId
""", substitutionValues: {'flightId': flightId});
}

Future<int> getActiveBookingsCount() async {
  final result = await _connection
    .query("SELECT COUNT(*) FROM bookings WHERE status = 'booked'");
  return result.first[0] as int;
}
}

class AppTheme {
  // Brand Colors
  static const Color primaryColor = Color(0xFF1A73E8);
  static const Color secondaryColor = Color(0xFF34A853);
  static const Color accentColor = Color(0xFFFA7B17);
  static const Color backgroundColor = Color(0xFFF8F9FA);
  static const Color cardColor = Colors.white;
  static const Color errorColor = Color(0xFFEA4335);

  // Typography
  static final TextTheme textTheme = TextTheme(
    displayLarge: GoogleFonts.poppins(
      fontSize: 24,
      fontWeight: FontWeight.bold,
      color: Colors.black87,
    ),
    displayMedium: GoogleFonts.poppins(
      fontSize: 22,
      fontWeight: FontWeight.w600,
      color: Colors.black87,
    ),
    displaySmall: GoogleFonts.poppins(
      fontSize: 20,
      fontWeight: FontWeight.w600,
      color: Colors.black87,
    ),
    headlineMedium: GoogleFonts.poppins(
      fontSize: 18,
      fontWeight: FontWeight.w600,
      color: Colors.black87,
    ),
  ),

```

```

headlineSmall: GoogleFonts.poppins(
    fontSize: 16,
    fontWeight: FontWeight.w600,
    color: Colors.black87,
),
titleLarge: GoogleFonts.poppins(
    fontSize: 16,
    fontWeight: FontWeight.w600,
    color: Colors.black87,
),
titleMedium: GoogleFonts.poppins(
    fontSize: 14,
    fontWeight: FontWeight.w500,
    color: Colors.black87,
),
bodyLarge: GoogleFonts.poppins(
    fontSize: 14,
    fontWeight: FontWeight.normal,
    color: Colors.black87,
),
bodyMedium: GoogleFonts.poppins(
    fontSize: 12,
    fontWeight: FontWeight.normal,
    color: Colors.black54,
),
);

// Light Theme
static final ThemeData lightTheme = ThemeData(
    useMaterial3: true,
    primaryColor: primaryColor,
    scaffoldBackgroundColor: backgroundColor,
    colorScheme: ColorScheme.light(
        primary: primaryColor,
        secondary: secondaryColor,
        error: errorColor,
        surface: cardColor,
        background: backgroundColor,
    ),
    textTheme: textTheme,
    cardTheme: CardTheme(
        color: cardColor,
        elevation: 2,
        shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),
        margin: const EdgeInsets.symmetric(vertical: 8, horizontal: 0),
    ),
    appBarTheme: AppBarTheme(

```

```

backgroundColor: primaryColor,
foregroundColor: Colors.white,
elevation: 0,
centerTitle: true,
titleTextStyle: GoogleFonts.poppins(
  fontSize: 18,
  fontWeight: FontWeight.w600,
  color: Colors.white,
),
),
elevatedButtonTheme: ElevatedButtonThemeData(
  style: ElevatedButton.styleFrom(
    backgroundColor: primaryColor,
    foregroundColor: Colors.white,
    padding: const EdgeInsets.symmetric(horizontal: 24, vertical: 12),
    shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(12)),
    elevation: 1,
  ),
),
outlinedButtonTheme: OutlinedButtonThemeData(
  style: OutlinedButton.styleFrom(
    foregroundColor: primaryColor,
    side: const BorderSide(color: primaryColor),
    padding: const EdgeInsets.symmetric(horizontal: 24, vertical: 12),
    shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(12)),
  ),
),
textButtonTheme: TextButtonThemeData(
  style: TextButton.styleFrom(
    foregroundColor: primaryColor,
    padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 12),
  ),
),
inputDecorationTheme: InputDecorationTheme(
  filled: true,
  fillColor: Colors.white,
  border: OutlineInputBorder(
    borderRadius: BorderRadius.circular(12),
    borderSide: BorderSide(color: Colors.grey.shade300),
  ),
  enabledBorder: OutlineInputBorder(
    borderRadius: BorderRadius.circular(12),
    borderSide: BorderSide(color: Colors.grey.shade300),
  ),
  focusedBorder: OutlineInputBorder(
    borderRadius: BorderRadius.circular(12),
    borderSide: const BorderSide(color: primaryColor, width: 2),

```

```

    ),
    errorBorder: OutlineInputBorder(
      borderRadius: BorderRadius.circular(12),
      borderSide: const BorderSide(color: errorColor),
    ),
    contentPadding: const EdgeInsets.symmetric(horizontal: 16, vertical: 16),
  ),
  snackBarTheme: SnackBarThemeData(
    backgroundColor: primaryColor,
    contentTextStyle: GoogleFonts.poppins(color: Colors.white),
    behavior: SnackBarBehavior.floating,
    shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(8)),
  ),
);

// Dark Theme
static final ThemeData darkTheme = ThemeData.dark().copyWith(
  useMaterial3: true,
  primaryColor: primaryColor,
  scaffoldBackgroundColor: const Color(0xFF121212),
  colorScheme: const ColorScheme.dark(
    primary: primaryColor,
    secondary: secondaryColor,
    error: errorColor,
    surface: Color(0xFF1E1E1E),
    background: Color(0xFF121212),
  ),
  textTheme: TextTheme(
    displayLarge: textTheme.displayLarge!.copyWith(color: Colors.white),
    displayMedium: textTheme.displayMedium!.copyWith(color: Colors.white),
    displaySmall: textTheme.displaySmall!.copyWith(color: Colors.white),
    headlineMedium: textTheme.headlineMedium!.copyWith(color:
Colors.white),
    headlineSmall: textTheme.headlineSmall!.copyWith(color: Colors.white),
    titleLarge: textTheme.titleLarge!.copyWith(color: Colors.white),
    titleMedium: textTheme.titleMedium!.copyWith(color: Colors.white),
    bodyLarge: textTheme.bodyLarge!.copyWith(color: Colors.white),
    bodyMedium: textTheme.bodyMedium!.copyWith(color: Colors.white70),
  ),
  cardTheme: CardTheme(
    color: const Color(0xFF1E1E1E),
    elevation: 2,
    shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),
    margin: const EdgeInsets.symmetric(vertical: 8, horizontal: 0),
  ),
  appBarTheme: AppBarTheme(
    backgroundColor: const Color(0xFF1E1E1E),

```

```

        foregroundColor: Colors.white,
        elevation: 0,
        centerTitle: true,
        titleTextStyle: GoogleFonts.poppins(
          fontSize: 18,
          fontWeight: FontWeight.w600,
          color: Colors.white,
        ),
      ),
    ),
    inputDecorationTheme: InputDecorationTheme(
      filled: true,
      fillColor: const Color(0xFF2A2A2A),
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(12),
        borderSide: BorderSide(color: Colors.grey.shade800),
      ),
      enabledBorder: OutlineInputBorder(
        borderRadius: BorderRadius.circular(12),
        borderSide: BorderSide(color: Colors.grey.shade800),
      ),
      focusedBorder: OutlineInputBorder(
        borderRadius: BorderRadius.circular(12),
        borderSide: const BorderSide(color: primaryColor, width: 2),
      ),
      errorBorder: OutlineInputBorder(
        borderRadius: BorderRadius.circular(12),
        borderSide: const BorderSide(color: errorColor),
      ),
      contentPadding: const EdgeInsets.symmetric(horizontal: 16, vertical: 16),
    ),
    snackBarTheme: SnackBarThemeData(
      backgroundColor: primaryColor,
      contentTextStyle: GoogleFonts.poppins(color: Colors.white),
      behavior: SnackBarBehavior.floating,
      shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(8)),
    ),
  );
}

void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  // Connect to database
  try {
    final dbService = DatabaseService();
    await dbService.connect();
    print('Database connected successfully');
  }
}

```

```

    } catch (e) {
        print('Failed to connect to database: $e');
        // Consider showing a user-friendly error or fallback to local storage
    }

    runApp(const MyApp());
}

/// USER MODEL
/// USER MODEL
class User {
    final String id;
    final String name;
    final String email;
    final String role; // "user" or "admin"
    final String password; // In a real app, this would be hashed

    User({
        required this.id,
        required this.name,
        required this.email,
        required this.role,
        required this.password,
    });
}

/// FLIGHT MODEL
class Flight {
    final String id;
    final String airline;
    final String flightNumber;
    final String source;
    final String destination;
    final DateTime departureTime;
    final DateTime arrivalTime;
    final double price;
    final int seatsAvailable;

    Flight({
        required this.id,
        required this.airline,
        required this.flightNumber,
        required this.source,
        required this.destination,
        required this.departureTime,
        required this.arrivalTime,
        required this.price,
    });
}

```

```

        required this.seatsAvailable,
    });
}

/// BOOKING MODEL
class Booking {
    final String id;
    final String userId;
    final String flightId;
    final String passengerName;
    final int passengerAge;
    final String seatNumber;
    final DateTime bookingTime;
    final String status; // "booked" or "cancelled"

    Booking({
        required this.id,
        required this.userId,
        required this.flightId,
        required this.passengerName,
        required this.passengerAge,
        required this.seatNumber,
        required this.bookingTime,
        required this.status,
    });
}

/// AUTH PROVIDER
class AuthProvider with ChangeNotifier {
    User? _currentUser;
    bool get isAuthenticated => _currentUser != null;
    bool get isAdmin => _currentUser?.role == 'admin';
    User? get currentUser => _currentUser;

    final _dbService = DatabaseService();

    // Registration method
    Future<void> register(String name, String email, String password) async {
        try {
            // Check if email already exists
            final users = await _dbService.getUsers();
            if (users.any((user) => user['email'] == email)) {
                throw Exception('Email already registered');
            }

            // Create new user with generated ID
            final userId = 'u${DateTime.now().millisecondsSinceEpoch}';

```



```

        await _dbService.createUser({
          'id': userId,
          'name': name,
          'email': email,
          'password': password,
          'role': 'user',
        });

        notifyListeners();
      } catch (e) {
        rethrow;
      }
    }
  }

  // Login method
  Future<void> login(String email, String password) async {
    try {
      final userData =
        await _dbService.getUserByEmailAndPassword(email, password);

      if (userData == null) {
        throw Exception('Invalid email or password');
      }

      _currentUser = User(
        id: userData['id'],
        name: userData['name'],
        email: userData['email'],
        role: userData['role'],
        password: userData['password'],
      );

      notifyListeners();
    } catch (e) {
      rethrow;
    }
  }

  void logout() {
    _currentUser = null;
    notifyListeners();
  }
}

/// REGISTRATION SCREEN
class RegistrationScreen extends StatefulWidget {
  const RegistrationScreen({super.key});

```

```

@override
State<RegistrationScreen> createState() => _RegistrationScreenState();
}

class _RegistrationScreenState extends State<RegistrationScreen> {
  final _formKey = GlobalKey<FormState>();
  final _nameController = TextEditingController();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  final _confirmPasswordController = TextEditingController();
  bool _isLoading = false;
  String? _errorMessage;

  @override
  void dispose() {
    _nameController.dispose();
    _emailController.dispose();
    _passwordController.dispose();
    _confirmPasswordController.dispose();
    super.dispose();
  }

  Future<void> _register() async {
    if (!_formKey.currentState!.validate()) return;

    setState(() {
      _isLoading = true;
      _errorMessage = null;
    });

    try {
      final authProvider = Provider.of<AuthProvider>(context, listen: false);
      await authProvider.register(
        _nameController.text,
        _emailController.text,
        _passwordController.text,
      );

      if (!mounted) return;

      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Text('Registration successful! Please log in.'),
          backgroundColor: Colors.green,
        ),
      );
    }
  }
}

```

```

Navigator.pushReplacement(
  context,
  PageTransitions.fadeTransition(page: const LoginScreen()),
);
} catch (e) {
  setState(() {
    _errorMessage = e.toString();
  });
} finally {
  setState(() {
    _isLoading = false;
  });
}
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Center(
      child: SingleChildScrollView(
        padding: const EdgeInsets.all(24),
        child: Form(
          key: _formKey,
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            crossAxisAlignment: CrossAxisAlignment.stretch,
            children: [
              const Icon(
                Icons.flight_takeoff,
                size: 70,
                color: AppTheme.primaryColor,
              ),
              const SizedBox(height: 16),
              Text(
                'Create an Account',
                style: Theme.of(context).textTheme.headlineSmall?.copyWith(
                  fontWeight: FontWeight.bold,
                ),
                textAlign: TextAlign.center,
              ),
              const SizedBox(height: 8),
              Text(
                'Register to book flights and manage your journeys',
                style: Theme.of(context).textTheme.bodyMedium?.copyWith(
                  color: Colors.grey,
                ),
              ),
            ],
          ),
        ),
      ),
    ),
  );
}

```

```

        textAlign: TextAlign.center,
      ),
      const SizedBox(height: 32),
      if (_errorMessage != null)
        Container(
          padding: const EdgeInsets.all(8),
          margin: const EdgeInsets.only(bottom: 16),
          decoration: BoxDecoration(
            color: Colors.red.shade100,
            borderRadius: BorderRadius.circular(8),
          ),
          child: Text(
            _errorMessage!,
            style: const TextStyle(color: Colors.red),
            textAlign: TextAlign.center,
          ),
        ),
      TextFormField(
        controller: _nameController,
        decoration: const InputDecoration(
          labelText: 'Full Name',
          border: OutlineInputBorder(),
          prefixIcon: Icon(Icons.person),
        ),
        validator: (value) {
          if (value == null || value.isEmpty) {
            return 'Please enter your name';
          }
          return null;
        },
      ),
      const SizedBox(height: 16),
      TextFormField(
        controller: _emailController,
        decoration: const InputDecoration(
          labelText: 'Email',
          border: OutlineInputBorder(),
          prefixIcon: Icon(Icons.email),
        ),
        keyboardType: TextInputType.emailAddress,
        validator: (value) {
          if (value == null || value.isEmpty) {
            return 'Please enter your email';
          }
          // Simple email validation
          if (!value.contains('@') || !value.contains('.')) {
            return 'Please enter a valid email';
          }
        },
      ),
    ),
  ),
),

```

```

    }
    return null;
  },
),
const SizedBox(height: 16),
TextFormField(
  controller: _passwordController,
  decoration: const InputDecoration(
    labelText: 'Password',
    border: OutlineInputBorder(),
    prefixIcon: Icon(Icons.lock),
  ),
  obscureText: true,
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter a password';
    }
    if (value.length < 6) {
      return 'Password must be at least 6 characters';
    }
    return null;
  },
),
const SizedBox(height: 16),
TextFormField(
  controller: _confirmPasswordController,
  decoration: const InputDecoration(
    labelText: 'Confirm Password',
    border: OutlineInputBorder(),
    prefixIcon: Icon(Icons.lock_outline),
  ),
  obscureText: true,
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please confirm your password';
    }
    if (value != _passwordController.text) {
      return 'Passwords do not match';
    }
    return null;
  },
),
const SizedBox(height: 24),
ElevatedButton(
  onPressed: _isLoading ? null : _register,
  style: ElevatedButton.styleFrom(
    padding: const EdgeInsets.symmetric(vertical: 16),

```

```

    ),
    child: _isLoading
      ? const SizedBox(
        height: 20,
        width: 20,
        child: CircularProgressIndicator(
          strokeWidth: 2,
          color: Colors.white,
        ),
      )
      : const Text('Register'),
  ),
  const SizedBox(height: 24),
  Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      const Text('Already have an account?'),
      TextButton(
        onPressed: () {
          Navigator.pushReplacement(
            context,
            PageTransitions.fadeTransition(
              page: const LoginScreen(),
            ),
          );
        },
        child: const Text('Login'),
      ),
    ],
  ),
],
),
],
),
),
),
),
),
);
}
}

```

```

/// FLIGHT PROVIDER
class FlightProvider with ChangeNotifier {
  List<Flight> _flights = [];
  final _dbService = DatabaseService();

  FlightProvider() {
    _loadFlights();
  }
}

```

```

List<Flight> get flights => [..._flights];

Future<void> _loadFlights() async {
  try {
    final flightData = await _dbService.getFlights();
    print('Retrieved ${flightData.length} flights from database');

    _flights = flightData
      .map((data) => Flight(
        id: data['id'].toString(),
        airline: data['airline'].toString(),
        flightNumber: data['flightNumber'].toString(),
        source: data['source'].toString(),
        destination: data['destination'].toString(),
        departureTime: data['departureTime'] as DateTime,
        arrivalTime: data['arrivalTime'] as DateTime,
        price: double.parse(data['price'].toString()),
        seatsAvailable: int.parse(data['seatsAvailable'].toString()),
      ))
      .toList();

    print('Successfully converted ${_flights.length} flights');
    notifyListeners();
  } catch (e) {
    print('Error loading flights: $e');
    print('Stack trace: ${StackTrace.current}');
  }
}

List<String> get allSources {
  return _flights.map((flight) => flight.source).toSet().toList();
}

List<String> get allDestinations {
  return _flights.map((flight) => flight.destination).toSet().toList();
}

Flight findById(String id) {
  return _flights.firstWhere((flight) => flight.id == id);
}

Future<List<Flight>> searchFlights({
  required String source,
  required String destination,
  required DateTime date,
}) async {
  try {

```

```

final searchResults = await _dbService.searchFlights(
    source: source,
    destination: destination,
    date: date,
);

print('Found ${searchResults.length} flights matching search criteria');

return searchResults
    .map((data) => Flight(
        id: data['id'].toString(),
        airline: data['airline'].toString(),
        flightNumber: data['flightNumber'].toString(),
        source: data['source'].toString(),
        destination: data['destination'].toString(),
        departureTime: data['departureTime'] as DateTime,
        arrivalTime: data['arrivalTime'] as DateTime,
        price: double.parse(data['price'].toString()),
        seatsAvailable: int.parse(data['seatsAvailable'].toString()),
    ))
    .toList();
} catch (e) {
    print('Error searching flights: $e');
    print('Stack trace: ${StackTrace.current}');
    return [];
}
}

Future<void> addFlight(Flight flight) async {
    try {
        await _dbService.createFlight({
            'id': flight.id,
            'airline': flight.airline,
            'flightNumber': flight.flightNumber,
            'source': flight.source,
            'destination': flight.destination,
            'departureTime': flight.departureTime,
            'arrivalTime': flight.arrivalTime,
            'price': flight.price,
            'seatsAvailable': flight.seatsAvailable,
        });

        _flights.add(flight);
        notifyListeners();
    } catch (e) {
        print('Error adding flight: $e');
        rethrow;
    }
}

```



```

    }
    }

Future<void> updateFlight(Flight updatedFlight) async {
    try {
        await _dbService.updateFlight({
            'id': updatedFlight.id,
            'airline': updatedFlight.airline,
            'flightNumber': updatedFlight.flightNumber,
            'source': updatedFlight.source,
            'destination': updatedFlight.destination,
            'departureTime': updatedFlight.departureTime,
            'arrivalTime': updatedFlight.arrivalTime,
            'price': updatedFlight.price,
            'seatsAvailable': updatedFlight.seatsAvailable,
        });

        final index =
            _flights.indexWhere((flight) => flight.id == updatedFlight.id);
        if (index >= 0) {
            _flights[index] = updatedFlight;
            notifyListeners();
        }
    } catch (e) {
        print('Error updating flight: $e');
        rethrow;
    }
}

Future<void> deleteFlight(String id) async {
    try {
        await _dbService.deleteFlight(id);
        _flights.removeWhere((flight) => flight.id == id);
        notifyListeners();
    } catch (e) {
        print('Error deleting flight: $e');
        rethrow;
    }
}

}

/// BOOKING PROVIDER
class BookingProvider with ChangeNotifier {
    List<Booking> _bookings = [];
    final _dbService = DatabaseService();

    BookingProvider() {

```

```

        _loadBookings();
    }

    List<Booking> get bookings => [..._bookings];

    Future<void> _loadBookings() async {
        try {
            final bookingData = await _dbService.getBookings();
            _bookings = bookingData
                .map((data) => Booking(
                    id: data['id'],
                    userId: data['userId'],
                    flightId: data['flightId'],
                    passengerName: data['passengerName'],
                    passengerAge: data['passengerAge'],
                    seatNumber: data['seatNumber'],
                    bookingTime: data['bookingTime'],
                    status: data['status'],
                ))
                .toList();
            notifyListeners();
        } catch (e) {
            print('Error loading bookings: $e');
        }
    }

    Future<List<Booking>> getUserBookings(String userId) async {
        try {
            final userBookingData = await _dbService.getUserBookings(userId);
            return userBookingData
                .map((data) => Booking(
                    id: data['id'],
                    userId: data['userId'],
                    flightId: data['flightId'],
                    passengerName: data['passengerName'],
                    passengerAge: data['passengerAge'],
                    seatNumber: data['seatNumber'],
                    bookingTime: data['bookingTime'],
                    status: data['status'],
                ))
                .toList();
        } catch (e) {
            print('Error getting user bookings: $e');
            return [];
        }
    }
}

```

```

Future<void> addBooking(Booking booking) async {
  try {
    await _dbService.createBooking({
      'id': booking.id,
      'userId': booking.userId,
      'flightId': booking.flightId,
      'passengerName': booking.passengerName,
      'passengerAge': booking.passengerAge,
      'seatNumber': booking.seatNumber,
      'bookingTime': booking.bookingTime,
      'status': booking.status,
    });

    _bookings.add(booking);
    notifyListeners();
  } catch (e) {
    print('Error adding booking: $e');
    rethrow;
  }
}

Future<void> cancelBooking(String bookingId) async {
  try {
    await _dbService.cancelBooking(bookingId);

    final index = _bookings.indexWhere((booking) => booking.id == bookingId);
    if (index >= 0) {
      final booking = _bookings[index];
      _bookings[index] = Booking(
        id: booking.id,
        userId: booking.userId,
        flightId: booking.flightId,
        passengerName: booking.passengerName,
        passengerAge: booking.passengerAge,
        seatNumber: booking.seatNumber,
        bookingTime: booking.bookingTime,
        status: 'cancelled',
      );
      notifyListeners();
    }
  } catch (e) {
    print('Error cancelling booking: $e');
    rethrow;
  }
}

Future<int> getActiveBookingsCount() async {

```

```

    try {
      return await _dbService.getActiveBookingsCount();
    } catch (e) {
      print('Error counting active bookings: $e');
      return 0;
    }
  }
}

/// MAIN APP WIDGET
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (_) => AuthProvider()),
        ChangeNotifierProvider(create: (_) => FlightProvider()),
        ChangeNotifierProvider(create: (_) => BookingProvider()),
      ],
      child: MaterialApp(
        debugShowCheckedModeBanner: false,
        title: 'SkyBoard Flight Booking',
        theme: AppTheme.lightTheme,
        darkTheme: AppTheme.darkTheme,
        themeMode: ThemeMode.system,
        home: const SplashScreen(),
        routes: {
          '/login': (ctx) => const LoginScreen(),
          '/register': (ctx) => const RegistrationScreen(),
          '/home': (ctx) => const HomeScreen(),
          '/flight-details': (ctx) => const FlightDetailsScreen(),
          '/my-bookings': (ctx) => const MyBookingsScreen(),
          '/admin-dashboard': (ctx) => const AdminDashboardScreen(),
          '/manage-flights': (ctx) => const ManageFlightsScreen(),
        },
      ),
    );
  }
}

/// WELCOME SCREEN
class WelcomeScreen extends StatelessWidget {
  const WelcomeScreen({super.key});

  @override

```

```

Widget build(BuildContext context) {
  return Scaffold(
    body: Container(
      decoration: BoxDecoration(
        gradient: LinearGradient(
          begin: Alignment.topCenter,
          end: Alignment.bottomCenter,
          colors: [
            AppTheme.primaryColor,
            AppTheme.primaryColor.withBlue(180),
          ],
        ),
      ),
    child: SafeArea(
      child: Padding(
        padding: const EdgeInsets.all(24.0),
        child: Column(
          children: [
            const Expanded(
              flex: 3,
              child: Center(
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.min,
                  children: [
                    Icon(
                      Icons.flight_takeoff,
                      size: 80,
                      color: Colors.white,
                    ),
                    SizedBox(height: 24),
                    Text(
                      'SkyBoard',
                      style: TextStyle(
                        fontSize: 32,
                        fontWeight: FontWeight.bold,
                        color: Colors.white,
                      ),
                    ),
                    SizedBox(height: 16),
                    Text(
                      'Your journey begins here',
                      style: TextStyle(
                        fontSize: 18,
                        color: Colors.white70,
                      ),
                    ),
                  ],
                ),
              ),
            ),
          ],
        ),
      ),
    ),
  );
}

```

```
),
),
Expanded(
  flex: 2,
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    crossAxisAlignment: CrossAxisAlignment.stretch,
    children: [
      ElevatedButton(
        style: ElevatedButton.styleFrom(
          backgroundColor: Colors.white,
          foregroundColor: AppTheme.primaryColor,
          padding: const EdgeInsets.symmetric(vertical: 16),
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(12),
          ),
        ),
        onPressed: () {
          Navigator.push(
            context,
            PageTransitions.fadeTransition(
              page: const RegistrationScreen(),
            ),
          );
        },
      ),
      child: const Text(
        'Create Account',
        style: TextStyle(
          fontSize: 16, fontWeight: FontWeight.bold),
      ),
    ],
  ),
  const SizedBox(height: 16),
  OutlinedButton(
    style: OutlinedButton.styleFrom(
      foregroundColor: Colors.white,
      side: const BorderSide(color: Colors.white),
      padding: const EdgeInsets.symmetric(vertical: 16),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(12),
      ),
    ),
    onPressed: () {
      Navigator.push(
        context,
        PageTransitions.fadeTransition(
          page: const LoginScreen(),
        ),
      );
    },
  ),
);
```

```

        },
        child: const Text(
          'Login',
          style: TextStyle(
            fontSize: 16, fontWeight: FontWeight.bold),
        ),
      ),
    ],
  ),
),
],
),
),
),
);
}
}

/// SPLASH SCREEN
class SplashScreen extends StatefulWidget {
  const SplashScreen({super.key});

  @override
  State<SplashScreen> createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> {
  @override
  void initState() {
    super.initState();
    Future.delayed(const Duration(seconds: 2), () {
      Navigator.pushReplacement(
        context,
        PageTransitions.fadeTransition(page: const WelcomeScreen()),
      );
    });
  }

  // Rest of the splash screen code stays the same
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Theme.of(context).primaryColor,
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,

```

```

children: [
  const Icon(
    Icons.flight_takeoff,
    size: 80,
    color: Colors.white,
  ),
  const SizedBox(height: 24),
  Text(
    'SkyBoard',
    style: Theme.of(context).textTheme.headlineLarge?.copyWith(
      color: Colors.white,
      fontWeight: FontWeight.bold,
    ),
  ),
  const SizedBox(height: 8),
  const Text(
    'Your Flight Booking App',
    style: TextStyle(color: Colors.white70),
  ),
],
),
);
}
}

```

/// LOGIN SCREEN

```

class LoginScreen extends StatefulWidget {
  const LoginScreen({super.key});

```

```

  @override
  State<LoginScreen> createState() => _LoginScreenState();
}

```

```

class _LoginScreenState extends State<LoginScreen> {
  final _formKey = GlobalKey<FormState>();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();
  String? _errorMessage;
  bool _isLoading = false;

```

```

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }
}

```



```

Future<void> _login() async {
  if (!_formKey.currentState!.validate()) return;

  setState(() {
    _isLoading = true;
    _errorMessage = null;
  });

  final authProvider = Provider.of<AuthProvider>(context, listen: false);

  try {
    await authProvider.login(_emailController.text, _passwordController.text);

    if (!mounted) return;

    if (authProvider.isAdmin) {
      Navigator.pushReplacementNamed(context, '/admin-dashboard');
    } else {
      Navigator.pushReplacementNamed(context, '/home');
    }
  } catch (e) {
    setState(() {
      _errorMessage = e.toString();
    });
  } finally {
    if (mounted) {
      setState(() {
        _isLoading = false;
      });
    }
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Padding(
      padding: const EdgeInsets.all(24.0),
      child: Center(
        child: SingleChildScrollView(
          child: Form(
            key: _formKey,
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              crossAxisAlignment: CrossAxisAlignment.stretch,
              children: [

```

```

const Icon(
  Icons.flight,
  size: 70,
  color: AppTheme.primaryColor,
),
const SizedBox(height: 16),
Text(
  'Welcome to SkyBoard',
  style: Theme.of(context).textTheme.headlineSmall?.copyWith(
    fontWeight: FontWeight.bold,
  ),
  textAlign: TextAlign.center,
),
const SizedBox(height: 8),
Text(
  'Sign in to continue',
  style: Theme.of(context).textTheme.bodyMedium?.copyWith(
    color: Colors.grey,
  ),
  textAlign: TextAlign.center,
),
const SizedBox(height: 32),
if (_errorMessage != null)
  Container(
    padding: const EdgeInsets.all(8),
    margin: const EdgeInsets.only(bottom: 16),
    decoration: BoxDecoration(
      color: Colors.red.shade100,
      borderRadius: BorderRadius.circular(8),
    ),
    child: Text(
      _errorMessage!,
      style: const TextStyle(color: Colors.red),
      textAlign: TextAlign.center,
    ),
  ),
TextFormField(
  controller: _emailController,
  decoration: const InputDecoration(
    labelText: 'Email',
    border: OutlineInputBorder(),
    prefixIcon: Icon(Icons.email),
  ),
  keyboardType: TextInputType.emailAddress,
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter your email';
    }
  },
)

```

```

    }
    return null;
  },
),
const SizedBox(height: 16),
TextFormField(
  controller: _passwordController,
  decoration: const InputDecoration(
    labelText: 'Password',
    border: OutlineInputBorder(),
    prefixIcon: Icon(Icons.lock),
  ),
  obscureText: true,
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter your password';
    }
    return null;
  },
),
const SizedBox(height: 8),
Align(
  alignment: Alignment.centerRight,
  child: TextButton(
    onPressed: () {
      // Could implement password reset in the future
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content:
            Text('Password reset feature coming soon')),
        );
    },
    child: const Text('Forgot Password?'),
  ),
),
const SizedBox(height: 16),
ElevatedButton(
  onPressed: _isLoading ? null : _login,
  style: ElevatedButton.styleFrom(
    padding: const EdgeInsets.symmetric(vertical: 16),
  ),
  child: _isLoading
    ? const SizedBox(
        height: 20,
        width: 20,
        child: CircularProgressIndicator(
          strokeWidth: 2,

```

```

        color: Colors.white,
      ),
    ),
    : const Text('Login'),
  ),
  const SizedBox(height: 24),
  Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      const Text("Don't have an account?"),
      TextButton(
        onPressed: () {
          Navigator.pushReplacement(
            context,
            PageTransitions.fadeTransition(
              page: const RegistrationScreen(),
            ),
          );
        },
        child: const Text('Register'),
      ),
    ],
  ),
  const SizedBox(height: 16),
  const Divider(),
  const SizedBox(height: 16),
  Text(
    'Admin access:',
    style: TextStyle(color: Colors.grey.shade600),
    textAlign: TextAlign.center,
  ),
  const SizedBox(height: 8),
  const Text(
    'admin@example.com / admin123',
    style: TextStyle(fontWeight: FontWeight.w500),
    textAlign: TextAlign.center,
  ),
),
],
),
),
),
),
),
);
}
}

```

```

/// HOME SCREEN (User)

```

```

class HomeScreen extends StatefulWidget {
  const HomeScreen({super.key});

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  String? _selectedSource;
  String? _selectedDestination;
  DateTime? _selectedDate = DateTime.now();
  List<Flight> _searchResults = [];
  bool _hasSearched = false;

  void _searchFlights() {
    if (_selectedSource == null ||
        _selectedDestination == null ||
        _selectedDate == null) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Please fill all fields')),
      );
      return;
    }

    final flightProvider = Provider.of<FlightProvider>(context, listen: false);
    flightProvider
      .searchFlights(
        source: _selectedSource!,
        destination: _selectedDestination!,
        date: _selectedDate!,
      )
      .then((results) {
        setState(() {
          _searchResults = results;
          _hasSearched = true;
        });
      }).catchError((error) {
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(content: Text('Error searching flights: $error')),
        );
      });

    setState(() {
      _hasSearched = true;
    });
  }
}

```

```

@override
Widget build(BuildContext context) {
  final authProvider = Provider.of<AuthProvider>(context);
  final flightProvider = Provider.of<FlightProvider>(context);
  final sources = flightProvider.allSources;
  final destinations = flightProvider.allDestinations;

  return Scaffold(
    appBar: AppBar(
      title: const Text('SkyBoard'),
      actions: [
        IconButton(
          icon: const Icon(Icons.airplane_ticket),
          onPressed: () {
            Navigator.pushNamed(context, '/my-bookings');
          },
        ),
        IconButton(
          icon: const Icon(Icons.logout),
          onPressed: () {
            authProvider.logout();
            Navigator.pushReplacementNamed(context, '/login');
          },
        ),
      ],
    ),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.stretch,
        children: [
          Card(
            elevation: 4,
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(12)),
            child: Padding(
              padding: const EdgeInsets.all(16.0),
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    'Search Flights',
                    style: Theme.of(context).textTheme.titleLarge?.copyWith(
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                ],
              ),
            ),
            const SizedBox(height: 16),

```

```

DropdownButtonFormField<String>(
  decoration: const InputDecoration(
    labelText: 'From',
    border: OutlineInputBorder(),
    prefixIcon: Icon(Icons.flight_takeoff),
  ),
  value: _selectedSource,
  items: sources
    .map((source) => DropdownMenuItem(
      value: source,
      child: Text(source),
    ))
    .toList(),
  onChanged: (value) {
    setState() {
      _selectedSource = value;
    };
  },
),
const SizedBox(height: 16),
DropdownButtonFormField<String>(
  decoration: const InputDecoration(
    labelText: 'To',
    border: OutlineInputBorder(),
    prefixIcon: Icon(Icons.flight_land),
  ),
  value: _selectedDestination,
  items: destinations
    .map((destination) => DropdownMenuItem(
      value: destination,
      child: Text(destination),
    ))
    .toList(),
  onChanged: (value) {
    setState() {
      _selectedDestination = value;
    };
  },
),
const SizedBox(height: 16),
InkWell(
  onTap: () async {
    final date = await showDatePicker(
      context: context,
      initialDate: _selectedDate ?? DateTime.now(),
      firstDate: DateTime.now(),
      lastDate:

```

```

        DateTime.now().add(const Duration(days: 365)),
      );
      if (date != null) {
        setState(() {
          _selectedDate = date;
        });
      }
    },
    child: InputDecorator(
      decoration: const InputDecoration(
        labelText: 'Date',
        border: OutlineInputBorder(),
        prefixIcon: Icon(Icons.calendar_today),
      ),
      child: Text(
        _selectedDate != null
          ? DateFormat('dd MMM yyyy').format(_selectedDate!)
          : 'Select a date',
      ),
    ),
  ),
  const SizedBox(height: 16),
  SizedBox(
    width: double.infinity,
    child: ElevatedButton.icon(
      icon: const Icon(Icons.search),
      label: const Text('Search Flights'),
      onPressed: _searchFlights,
    ),
  ),
],
),
),
const SizedBox(height: 16),
if (_hasSearched) ...[
  Text(
    'Search Results',
    style: Theme.of(context).textTheme.titleMedium?.copyWith(
      fontWeight: FontWeight.bold,
    ),
  ),
),
const SizedBox(height: 8),
Expanded(
  child: _searchResults.isEmpty
    ? const Center(
        child: Text('No flights found for selected criteria'),

```



```

)
: ListView.builder(
  itemCount: _searchResults.length,
  itemBuilder: (context, index) {
    final flight = _searchResults[index];
    return Card(
      margin: const EdgeInsets.only(bottom: 12),
      child: ListTile(
        contentPadding: const EdgeInsets.all(16),
        title: Row(
          mainAxisAlignment:
            MainAxisAlignment.spaceBetween,
          children: [
            Text(flight.airline),
            Text(
              '₹${flight.price.toStringAsFixed(0)}',
              style: const TextStyle(
                color: Colors.indigo,
                fontWeight: FontWeight.bold,
              ),
            ),
          ],
        ),
        subtitle: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            const SizedBox(height: 8),
            Row(
              mainAxisAlignment:
                MainAxisAlignment.spaceBetween,
              children: [
                Column(
                  crossAxisAlignment:
                    CrossAxisAlignment.start,
                  children: [
                    Text(
                      DateFormat('HH:mm')
                        .format(flight.departureTime),
                      style: const TextStyle(
                        fontWeight: FontWeight.bold,
                        fontSize: 16,
                      ),
                    ),
                    Text(flight.source),
                  ],
                ),
                Column(

```

```
      children: [
        const Icon(Icons.arrow_forward),
        Text('Flight ${flight.flightNumber}'),
      ],
    ),
    Column(
      crossAxisAlignment:
        CrossAxisAlignment.end,
      children: [
        Text(
          DateFormat('HH:mm')
            .format(flight.arrivalTime),
          style: const TextStyle(
            fontWeight: FontWeight.bold,
            fontSize: 16,
          ),
        ),
        Text(flight.destination),
      ],
    ),
  ],
),
const SizedBox(height: 12),
Row(
  mainAxisAlignment:
    MainAxisAlignment.spaceBetween,
  children: [
    Text(
      '${flight.seatsAvailable} seats left'),
    ElevatedButton(
      onPressed: () {
        Navigator.pushNamed(
          context,
          '/flight-details',
          arguments: flight.id,
        );
      },
      child: const Text('Book Now'),
    ),
  ],
),
);
```

```

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

/// FLIGHT DETAILS SCREEN
class FlightDetailsScreen extends StatefulWidget {
  const FlightDetailsScreen({super.key});

  @override
  State<FlightDetailsScreen> createState() => _FlightDetailsScreenState();
}

class _FlightDetailsScreenState extends State<FlightDetailsScreen> {
  final _formKey = GlobalKey<FormState>();
  final _nameController = TextEditingController();
  final _ageController = TextEditingController();
  String? _selectedSeat;

  @override
  void dispose() {
    _nameController.dispose();
    _ageController.dispose();
    super.dispose();
  }

  List<String> _generateSeats(int count) {
    const rows = ['A', 'B', 'C', 'D', 'E', 'F'];
    final seats = <String>[];

    for (int i = 1; i <= count ~/ 6 + 1; i++) {
      for (var row in rows) {
        seats.add('$i$row');
        if (seats.length >= count) break;
      }
      if (seats.length >= count) break;
    }

    return seats;
  }

  void _bookFlight(String flightId, Flight flight) {
    if (! _formKey.currentState!.validate() || _selectedSeat == null) {

```

```

ScaffoldMessenger.of(context).showSnackBar(
  const SnackBar(content: Text('Please fill all fields')),
);
return;
}

final authProvider = Provider.of<AuthProvider>(context, listen: false);
final bookingProvider =
  Provider.of<BookingProvider>(context, listen: false);

if (authProvider.currentUser == null) {
  ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(
      content: Text('You need to be logged in to book a flight')),
    );
  return;
}

// Create a new booking
final newBooking = Booking(
  id: 'b${DateTime.now().millisecondsSinceEpoch}',
  userId: authProvider.currentUser!.id,
  flightId: flightId,
  passengerName: _nameController.text,
  passengerAge: int.parse(_ageController.text),
  seatNumber: _selectedSeat!,
  bookingTime: DateTime.now(),
  status: 'booked',
);

bookingProvider.addBooking(newBooking);

// Update flight seats
final flightProvider = Provider.of<FlightProvider>(context, listen: false);
final updatedFlight = Flight(
  id: flight.id,
  airline: flight.airline,
  flightNumber: flight.flightNumber,
  source: flight.source,
  destination: flight.destination,
  departureTime: flight.departureTime,
  arrivalTime: flight.arrivalTime,
  price: flight.price,
  seatsAvailable: flight.seatsAvailable - 1,
);

flightProvider.updateFlight(updatedFlight);

```

```

// Show success message and navigate back
showDialog(
  context: context,
  builder: (ctx) => AlertDialog(
    title: const Text('Booking Confirmed'),
    content: Column(
      mainAxisAlignment: MainAxisAlignment.min,
      children: [
        const Icon(
          Icons.check_circle,
          color: Colors.green,
          size: 60,
        ),
        const SizedBox(height: 16),
        const Text('Your flight has been booked!'),
        const SizedBox(height: 8),
        Text('Booking ID: ${newBooking.id}'),
        Text('Passenger: ${newBooking.passengerName}'),
        Text('Seat: ${newBooking.seatNumber}'),
      ],
    ),
    actions: [
      TextButton(
        onPressed: () {
          Navigator.of(ctx).pop();
          Navigator.of(context).pushReplacementNamed('/my-bookings');
        },
        child: const Text('View My Bookings'),
      ),
      ElevatedButton(
        onPressed: () {
          Navigator.of(ctx).pop();
          Navigator.of(context).pushReplacementNamed('/home');
        },
        child: const Text('Back to Home'),
      ),
    ],
  ),
);
}

@override
Widget build(BuildContext context) {
  final flightId = ModalRoute.of(context)!.settings.arguments as String;
  final flightProvider = Provider.of<FlightProvider>(context);
  final flight = flightProvider.findById(flightId);

```



```

        style: TextStyle(color: Colors.grey.shade600),
      ),
    ],
  ),
  Column(
    children: [
      const Icon(Icons.flight, color: Colors.indigo),
      const SizedBox(height: 4),
      Container(
        padding: const EdgeInsets.symmetric(
          horizontal: 8,
          vertical: 4,
        ),
        decoration: BoxDecoration(
          color: Colors.indigo.shade100,
          borderRadius: BorderRadius.circular(12),
        ),
        child: Text(flight.flightNumber),
      ),
    ],
  ),
  Column(
    crossAxisAlignment: CrossAxisAlignment.end,
    children: [
      Text(
        DateFormat('HH:mm').format(flight.arrivalTime),
        style: const TextStyle(
          fontWeight: FontWeight.bold,
          fontSize: 20,
        ),
      ),
      Text(
        flight.destination,
        style: const TextStyle(fontSize: 16),
      ),
      Text(
        DateFormat('dd MMM yyyy')
          .format(flight.arrivalTime),
        style: TextStyle(color: Colors.grey.shade600),
      ),
    ],
  ),
  const Divider(height: 32),
  Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,

```

```

children: [
  Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      const Text('Airline'),
      const SizedBox(height: 4),
      Text(
        flight.airline,
        style: const TextStyle(
          fontWeight: FontWeight.bold,
          fontSize: 16,
        ),
      ),
    ],
  ),
  Column(
    crossAxisAlignment: CrossAxisAlignment.end,
    children: [
      const Text('Price'),
      const SizedBox(height: 4),
      Text(
        '₹${flight.price.toStringAsFixed(0)}',
        style: const TextStyle(
          fontWeight: FontWeight.bold,
          fontSize: 20,
          color: Colors.indigo,
        ),
      ),
    ],
  ),
],
),
const SizedBox(height: 8),
Text('Available Seats: ${flight.seatsAvailable}'),
],
),
),
const SizedBox(height: 24),
Text(
  'Passenger Information',
  style: Theme.of(context).textTheme.titleLarge?.copyWith(
    fontWeight: FontWeight.bold,
  ),
),
const SizedBox(height: 16),
Form(

```



```

key: _formKey,
child: Column(
  children: [
    TextFormField(
      controller: _nameController,
      decoration: const InputDecoration(
        labelText: 'Passenger Name',
        border: OutlineInputBorder(),
        prefixIcon: Icon(Icons.person),
      ),
      validator: (value) {
        if (value == null || value.isEmpty) {
          return 'Please enter passenger name';
        }
        return null;
      },
    ),
    const SizedBox(height: 16),
    TextFormField(
      controller: _ageController,
      decoration: const InputDecoration(
        labelText: 'Age',
        border: OutlineInputBorder(),
        prefixIcon: Icon(Icons.calendar_today),
      ),
      keyboardType: TextInputType.number,
      validator: (value) {
        if (value == null || value.isEmpty) {
          return 'Please enter age';
        }
        final age = int.tryParse(value);
        if (age == null || age <= 0 || age > 120) {
          return 'Please enter a valid age';
        }
        return null;
      },
    ),
    const SizedBox(height: 16),
    DropdownButtonFormField<String>(
      decoration: const InputDecoration(
        labelText: 'Select Seat',
        border: OutlineInputBorder(),
        prefixIcon: Icon(Icons.airline_seat_recline_normal),
      ),
      value: _selectedSeat,
      items: availableSeats
        .map((seat) => DropdownMenuItem(

```

```

        value: seat,
        child: Text('Seat $seat'),
      ))
      .toList(),
    onChanged: (value) {
      setState(() {
        _selectedSeat = value;
      });
    },
    validator: (value) {
      if (value == null || value.isEmpty) {
        return 'Please select a seat';
      }
      return null;
    },
  ),
  const SizedBox(height: 24),
  SizedBox(
    width: double.infinity,
    child: ElevatedButton.icon(
      icon: const Icon(Icons.confirmation_number),
      label: const Text('Book Flight'),
      style: ElevatedButton.styleFrom(
        padding: const EdgeInsets.symmetric(vertical: 16),
      ),
      onPressed: () => _bookFlight(flightId, flight),
    ),
  ),
],
),
),
],
),
),
);
}
}

```

/// MY BOOKINGS SCREEN

```

class MyBookingsScreen extends StatelessWidget {
  const MyBookingsScreen({super.key});

```

```

  @override

```

```

  Widget build(BuildContext context) {
    final authProvider = Provider.of<AuthProvider>(context);
    final bookingProvider = Provider.of<BookingProvider>(context);
    final flightProvider = Provider.of<FlightProvider>(context);

```

```

if (authProvider.currentUser == null) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('My Bookings'),
    ),
    body: const Center(
      child: Text('Please log in to view your bookings.'),
    ),
  );
}

final userBookings =
  bookingProvider.getUserBookings(authProvider.currentUser!.id);

return Scaffold(
  appBar: AppBar(
    title: const Text('My Bookings'),
  ),
  body: FutureBuilder<List<Booking>>(
    future: userBookings,
    builder: (context, snapshot) {
      if (snapshot.connectionState == ConnectionState.waiting) {
        return const Center(child: CircularProgressIndicator());
      } else if (snapshot.hasError) {
        return Center(child: Text('Error: ${snapshot.error}'));
      } else if (!snapshot.hasData || snapshot.data!.isEmpty) {
        return const Center(
          child: Text('You have no bookings.'),
        );
      } else {
        final bookings = snapshot.data!;
        return ListView.builder(
          itemCount: bookings.length,
          itemBuilder: (context, index) {
            final booking = bookings[index];
            final flight = flightProvider.findById(booking.flightId);

            return Card(
              margin: const EdgeInsets.all(8),
              child: ListTile(
                contentPadding: const EdgeInsets.all(16),
                title: Text(
                  '${flight.airline} - ${flight.flightNumber}',
                  style: const TextStyle(fontWeight: FontWeight.bold),
                ),
                subtitle: Column(

```

```

crossAxisAlignment: CrossAxisAlignment.start,
children: [
  const SizedBox(height: 8),
  Text('Passenger: ${booking.passengerName}'),
  Text('Seat: ${booking.seatNumber}'),
  Text('Status: ${booking.status}'),
  const SizedBox(height: 8),
  Text(
    'Departure: ${DateFormat('dd MMM yyyy,
HH:mm').format(flight.departureTime)}',
  ),
  Text(
    'Arrival: ${DateFormat('dd MMM yyyy,
HH:mm').format(flight.arrivalTime)}',
  ),
],
),
trailing: booking.status == 'booked'
? ElevatedButton(
  onPressed: () {
    bookingProvider.cancelBooking(booking.id);
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('Booking cancelled.')),
      );
  },
  child: const Text('Cancel'),
)
: null,
),
);
},
);
}
},
),
);
}
}

```

/// ADMIN DASHBOARD SCREEN

```

class AdminDashboardScreen extends StatelessWidget {
  const AdminDashboardScreen({super.key});

  @override
  Widget build(BuildContext context) {
    final authProvider = Provider.of<AuthProvider>(context);

```

```

final flightProvider = Provider.of<FlightProvider>(context);
final bookingProvider = Provider.of<BookingProvider>(context);

// Redirect if not admin
if (!authProvider.isAdmin) {
  WidgetsBinding.instance.addPostFrameCallback((_) {
    Navigator.pushReplacementNamed(context, '/login');
  });
}

return Scaffold(
  appBar: AppBar(
    title: const Text('Admin Dashboard'),
    actions: [
      IconButton(
        icon: const Icon(Icons.logout),
        onPressed: () {
          authProvider.logout();
          Navigator.pushReplacementNamed(context, '/login');
        },
      ),
    ],
  ),
  drawer: Drawer(
    child: ListView(
      padding: EdgeInsets.zero,
      children: [
        DrawerHeader(
          decoration: BoxDecoration(
            color: Theme.of(context).primaryColor,
          ),
          child: const Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              CircleAvatar(
                radius: 30,
                child: Icon(Icons.person, size: 30),
              ),
              SizedBox(height: 8),
              Text(
                'Admin Panel',
                style: TextStyle(
                  color: Colors.white,
                  fontSize: 18,
                ),
              ),
            ],
          ),
        ),
      ],
    ),
  ),
  Text(

```

```

'SkyBoard Management',
style: TextStyle(
  color: Colors.white70,
  fontSize: 14,
),
),
],
),
),
ListTile(
  leading: const Icon(Icons.dashboard),
  title: const Text('Dashboard'),
  selected: true,
  onTap: () {
    Navigator.pop(context);
  },
),
ListTile(
  leading: const Icon(Icons.flight),
  title: const Text('Manage Flights'),
  onTap: () {
    Navigator.pop(context);
    Navigator.pushNamed(context, '/manage-flights');
  },
),
const Divider(),
ListTile(
  leading: const Icon(Icons.logout),
  title: const Text('Logout'),
  onTap: () {
    authProvider.logout();
    Navigator.pushReplacementNamed(context, '/login');
  },
),
],
),
),
body: Padding(
  padding: const EdgeInsets.all(16.0),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        'Overview',
        style: Theme.of(context).textTheme.titleLarge?.copyWith(
          fontWeight: FontWeight.bold,
        ),
      ),

```

```

    ),
    const SizedBox(height: 16),
    // Stats Cards
    Row(
      children: [
        Expanded(
          child: _buildStatCard(
            context,
            Icons.flight,
            flightProvider.flights.length.toString(),
            'Total Flights',
            Colors.blue.shade100,
          ),
        ),
        const SizedBox(width: 16),
        Expanded(
          child: _buildStatCard(
            context,
            Icons.confirmation_number,
            bookingProvider.getActiveBookingsCount().toString(),
            'Active Bookings',
            Colors.green.shade100,
          ),
        ),
      ],
    ),
    const SizedBox(height: 24),
    Text(
      'Recent Bookings',
      style: Theme.of(context).textTheme.titleLarge?.copyWith(
        fontWeight: FontWeight.bold,
      ),
    ),
    const SizedBox(height: 16),
    Expanded(
      child: bookingProvider.bookings.isEmpty
        ? const Center(
            child: Text('No bookings found.'),
          )
        : ListView.builder(
            itemCount: bookingProvider.bookings.length,
            itemBuilder: (context, index) {
              final booking = bookingProvider.bookings[index];
              final flight =
                flightProvider.findById(booking.flightId);

              return Card(

```





```

String title,
Color color,
) {
return Card(
  elevation: 4,
  shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(12)),
  child: Padding(
    padding: const EdgeInsets.all(16.0),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        CircleAvatar(
          radius: 20,
          backgroundColor: color,
          child: Icon(icon, color: Colors.white),
        ),
        const SizedBox(height: 12),
        Text(
          value,
          style: Theme.of(context).textTheme.headlineMedium?.copyWith(
            fontWeight: FontWeight.bold,
          ),
        ),
        const SizedBox(height: 4),
        Text(title),
      ],
    ),
  ),
);
}
}

```

/// MANAGE FLIGHTS SCREEN

```

class ManageFlightsScreen extends StatelessWidget {
  const ManageFlightsScreen({super.key});

```

@override

```

Widget build(BuildContext context) {
  final authProvider = Provider.of<AuthProvider>(context);
  final flightProvider = Provider.of<FlightProvider>(context);

```

// Redirect if not admin

```

if (!authProvider.isAdmin) {
  WidgetsBinding.instance.addPostFrameCallback((_) {
    Navigator.pushReplacementNamed(context, '/login');
  });
}
}

```

```

return Scaffold(
  appBar: AppBar(
    title: const Text('Manage Flights'),
  ),
  body: Padding(
    padding: const EdgeInsets.all(16.0),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text(
          'All Flights',
          style: Theme.of(context).textTheme.titleLarge?.copyWith(
            fontWeight: FontWeight.bold,
          ),
        ),
        const SizedBox(height: 16),
        Expanded(
          child: ListView.builder(
            itemCount: flightProvider.flights.length,
            itemBuilder: (context, index) {
              final flight = flightProvider.flights[index];
              return Card(
                margin: const EdgeInsets.only(bottom: 12),
                child: ListTile(
                  contentPadding: const EdgeInsets.all(16),
                  title: Row(
                    mainAxisAlignment: MainAxisAlignment.spaceBetween,
                    children: [
                      Text(
                        '${flight.airline} - ${flight.flightNumber}',
                        style: const TextStyle(fontWeight: FontWeight.bold),
                      ),
                      Text(
                        '₹${flight.price.toStringAsFixed(0)}',
                        style: const TextStyle(
                          color: Colors.indigo,
                          fontWeight: FontWeight.bold,
                        ),
                      ),
                    ],
                  ),
                  subtitle: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      const SizedBox(height: 8),
                      Row(

```

	<pre> mainAxisAlignment: MainAxisAlignment.spaceBetween, children: [   Column(     crossAxisAlignment: CrossAxisAlignment.start,     children: [       Row(         children: [           const Icon(Icons.flight_takeoff,             size: 16),           const SizedBox(width: 4),           Text(flight.source),         ],       ),       Text(         DateFormat('HH:mm - dd MMM')           .format(flight.departureTime),         style:           TextStyle(color: Colors.grey.shade600),       ),     ],   ),   Column(     crossAxisAlignment: CrossAxisAlignment.end,     children: [       Row(         children: [           Text(flight.destination),           const SizedBox(width: 4),           const Icon(Icons.flight_land, size: 16),         ],       ),       Text(         DateFormat('HH:mm - dd MMM')           .format(flight.arrivalTime),         style:           TextStyle(color: Colors.grey.shade600),       ),     ],   ), ], ), const SizedBox(height: 8), Text('Available Seats: \${flight.seatsAvailable}'), const SizedBox(height: 8), Row(   mainAxisAlignment: MainAxisAlignment.end,   children: [ </pre>
--	--

	<div data-bbox="430 373 576 405">flight details</div> <pre> TextButton.icon(   icon: const Icon(Icons.edit),   label: const Text('Edit'),   onPressed: () {     // Handle edit flight - this would go to a form screen with   }, ), const SizedBox(width: 8), TextButton.icon(   icon:     const Icon(Icons.delete, color: Colors.red),   label: const Text('Delete',     style: TextStyle(color: Colors.red)),   onPressed: () {     // Confirm before deleting     showDialog(       context: context,       builder: (ctx) =&gt; AlertDialog(         title: const Text('Confirm Delete'),         content: Text(           '\${flight.flightNumber} from \${flight.source} to \${flight.destination}?',         ),         actions: [           TextButton(             onPressed: () =&gt;               Navigator.of(ctx).pop(),             child: const Text('Cancel'),           ),           TextButton(             onPressed: () {               flightProvider                 .deleteFlight(flight.id);               Navigator.of(ctx).pop();               ScaffoldMessenger.of(context)                 .showSnackBar(                   const SnackBar(                     content:                       Text('Flight deleted')),                 );             },             child: const Text(               'Delete',               style: TextStyle(color: Colors.red),             ),           ),         ],       ),     ); </pre>
--	---

```

        ],
      ),
    );
  },
),
],
),
],
),
),
);
},
),
),
],
),
),
floatingActionButton: FloatingActionButton(
  onPressed: () {
    // Navigate to add flight form
    // This would be a separate screen with a form for adding new flights
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('Add Flight functionality to be implemented')),
      );
    },
  child: const Icon(Icons.add),
),
);
}
}

class PageTransitions {
  // Fade transition
  static PageRouteBuilder fadeTransition({required Widget page}) {
    return PageRouteBuilder(
      pageBuilder: (context, animation, secondaryAnimation) => page,
      transitionsBuilder: (context, animation, secondaryAnimation, child) {
        const begin = 0.0;
        const end = 1.0;
        var curve = Curves.easeInOut;
        var curveTween = CurveTween(curve: curve);
        var tween = Tween(begin: begin, end: end).chain(curveTween);
        var opacityAnimation = animation.drive(tween);
        return FadeTransition(
          opacity: opacityAnimation,
          child: child,

```

```

    );
  },
  transitionDuration: const Duration(milliseconds: 300),
);
}

// Slide transition (from right)
static PageRouteBuilder slideTransition( {required Widget page} ) {
  return PageRouteBuilder(
    pageBuilder: (context, animation, secondaryAnimation) => page,
    transitionsBuilder: (context, animation, secondaryAnimation, child) {
      const begin = Offset(1.0, 0.0);
      const end = Offset.zero;
      var curve = Curves.easeInOut;
      var curveTween = CurveTween(curve: curve);
      var tween = Tween(begin: begin, end: end).chain(curveTween);
      var offsetAnimation = animation.drive(tween);
      return SlideTransition(
        position: offsetAnimation,
        child: child,
      );
    },
    transitionDuration: const Duration(milliseconds: 300),
  );
}

// Scale transition
static PageRouteBuilder scaleTransition( {required Widget page} ) {
  return PageRouteBuilder(
    pageBuilder: (context, animation, secondaryAnimation) => page,
    transitionsBuilder: (context, animation, secondaryAnimation, child) {
      const begin = 0.8;
      const end = 1.0;
      var curve = Curves.easeInOut;
      var curveTween = CurveTween(curve: curve);
      var tween = Tween(begin: begin, end: end).chain(curveTween);
      var scaleAnimation = animation.drive(tween);
      return ScaleTransition(
        scale: scaleAnimation,
        child: FadeTransition(
          opacity: animation,
          child: child,
        ),
      );
    },
    transitionDuration: const Duration(milliseconds: 300),
  );
}

```

```

    }
    }

class Responsive {
    static bool isMobile(BuildContext context) =>
        MediaQuery.of(context).size.width < 650;

    static bool isTablet(BuildContext context) =>
        MediaQuery.of(context).size.width >= 650 &&
        MediaQuery.of(context).size.width < 1100;

    static bool isDesktop(BuildContext context) =>
        MediaQuery.of(context).size.width >= 1100;

    static double horizontalPadding(BuildContext context) {
        if (isMobile(context)) return 16.0;
        if (isTablet(context)) return 24.0;
        return 32.0;
    }

    static Widget responsiveBuilder({
        required BuildContext context,
        required Widget mobile,
        Widget? tablet,
        Widget? desktop,
    }) {
        if (isDesktop(context) && desktop != null) {
            return desktop;
        }
        if (isTablet(context) && tablet != null) {
            return tablet;
        }
        return mobile;
    }
}

enum ErrorSeverity { info, warning, error }

class ErrorHandler {
    static void showError({
        required BuildContext context,
        required String message,
        ErrorSeverity severity = ErrorSeverity.error,
        String? actionLabel,
        VoidCallback? onAction,
        Duration duration = const Duration(seconds: 4),
    }) {

```

```

Color backgroundColor;
IconData icon;

switch (severity) {
  case ErrorSeverity.info:
    backgroundColor = Colors.blue;
    icon = Icons.info_outline;
    break;
  case ErrorSeverity.warning:
    backgroundColor = Colors.orange;
    icon = Icons.warning_amber_rounded;
    break;
  case ErrorSeverity.error:
    backgroundColor = Theme.of(context).colorScheme.error;
    icon = Icons.error_outline;
    break;
}

final snackBar = SnackBar(
  content: Row(
    children: [
      Icon(icon, color: Colors.white),
      const SizedBox(width: 8),
      Expanded(child: Text(message)),
    ],
  ),
  backgroundColor: backgroundColor,
  duration: duration,
  action: actionLabel != null && onAction != null
    ? SnackBarAction(
        label: actionLabel,
        textColor: Colors.white,
        onPressed: onAction,
      )
    : null,
  behavior: SnackBarBehavior.floating,
  margin: const EdgeInsets.all(8),
  shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(8)),
);

ScaffoldMessenger.of(context).showSnackBar(snackBar);
}

static Widget errorWidget({
  required String message,
  IconData icon = Icons.error_outline,
  String? buttonLabel,

```



```

VoidCallback? onPressed,
)) {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(
          icon,
          color: Colors.red,
          size: 60,
        ),
        const SizedBox(height: 16),
        Text(
          message,
          textAlign: TextAlign.center,
          style: const TextStyle(fontSize: 16),
        ),
        if (buttonLabel != null && onPressed != null) ...[
          const SizedBox(height: 24),
          ElevatedButton(
            onPressed: onPressed,
            child: Text(buttonLabel),
          ),
        ],
      ],
    ),
  );
}
}

```

```

class DateRangePicker extends StatefulWidget {
  final DateTime? startDate;
  final DateTime? endDate;
  final ValueChanged<DateTime?> onStartDateChanged;
  final ValueChanged<DateTime?> onEndDateChanged;
  final bool showEndDate;

```

```

  const DateRangePicker({
    Key? key,
    this.startDate,
    this.endDate,
    required this.onStartDateChanged,
    required this.onEndDateChanged,
    this.showEndDate = true,
  }) : super(key: key);

```

```

  @override

```

```

State<DateRangePicker> createState() => _DateRangePickerState();
}

class _DateRangePickerState extends State<DateRangePicker> {
  @override
  Widget build(BuildContext context) {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        InkWell(
          onTap: () async {
            final dateRange = await showDateRangePicker(
              context: context,
              firstDate: DateTime.now(),
              lastDate: DateTime.now().add(const Duration(days: 365)),
              initialDateRange:
                widget.startDate != null && widget.endDate != null
                  ? DateTimeRange(
                      start: widget.startDate!,
                      end: widget.endDate!,
                    )
                  : null,
              builder: (context, child) {
                return Theme(
                  data: Theme.of(context).copyWith(
                    colorScheme: Theme.of(context).colorScheme.copyWith(
                      primary: Theme.of(context).primaryColor,
                    ),
                ),
                  child: child!,
                );
              },
            );

            if (dateRange != null) {
              widget.onStartDateChanged(dateRange.start);
              widget
                .onEndDateChanged(widget.showEndDate ? dateRange.end : null);
            }
          },
          child: InputDecorator(
            decoration: const InputDecoration(
              labelText: 'Travel Dates',
              border: OutlineInputBorder(),
              prefixIcon: Icon(Icons.calendar_today),
            ),
            child: Row(

```

	<pre>mainAxisAlignment: MainAxisAlignment.spaceBetween, children: [   Text(     widget.startDate != null       ? DateFormat('dd MMM yyyy').format(widget.startDate!)       : 'Departure',   ),   if (widget.showEndDate) ...[     const Icon(Icons.arrow_forward, size: 16),     Text(       widget.endDate != null         ? DateFormat('dd MMM yyyy').format(widget.endDate!)         : 'Return (Optional)',     ),   ], ], ), ), ), ], ); }</pre>
<b>Screenshots</b>	

--	--

--	--

--	--

--	--

--	--



Database and tables :

Users table;

Flights table :

Booking table :

--	--

--	--

<b>Conclusion</b>	From this experiment, we have learned how to connect database to an application in real time - how it fetches data, how it stores data, etc.