Flutter + Riverpod + Supabase API Architecture (with .env)

🧭 Goal

To integrate **Supabase API** in a **clean Riverpod architecture**,
storing sensitive configurations like keys and URLs in a `.env` file
for secure and maintainable project structure.

🧱 Folder Structure

```
lib/
├── core/
│   ├── config/
│   │   └── env_config.dart
│   └── constants/
│       └── api_endpoints.dart
│
├── data/
│   ├── models/
│   │   └── user_model.dart
│   ├── services/
│   │   └── supabase_service.dart
│   └── repositories/
│       └── user_repository.dart
│
├── presentation/
│   ├── controllers/
│   │   └── user_controller.dart
│   ├── providers/
│   │   └── user_providers.dart
│   └── screens/
│       └── user_screen.dart
│
├── .env
└── main.dart
```

🗂️ 1. `.env` File

File: `.env` (store in root folder)

```
SUPABASE_URL=https://xyzcompany.supabase.co
SUPABASE_ANON_KEY=eyJhbGciOiJI...
```

✅ Do **NOT** commit this file to GitHub.
Add this line to your `.gitignore`:

```
.env
```

⚙️ 2. Environment Config Loader

File: `core/config/env_config.dart`

```
import 'package:flutter_dotenv/flutter_dotenv.dart';

class EnvConfig {
  static String get supabaseUrl => dotenv.env['SUPABASE_URL'] ?? '';
  static String get supabaseAnonKey => dotenv.env['SUPABASE_ANON_KEY'] ?? '';
}
```

This loads your environment variables safely.
You can access them anywhere with:

```
EnvConfig.supabaseUrl
EnvConfig.supabaseAnonKey
```

🔗 3. Supabase Service

File: `data/services/supabase_service.dart`

```
import 'package:supabase_flutter/supabase_flutter.dart';
import '../../core/config/env_config.dart';

class SupabaseService {
  final SupabaseClient client = SupabaseClient(
    EnvConfig.supabaseUrl,
    EnvConfig.supabaseAnonKey,
  );

  Future<List<Map<String, dynamic>>> fetchUsers() async {
    final response = await client.from('users').select();

    if (response.isEmpty) {
      throw Exception('No data found');
    }

    return List<Map<String, dynamic>>.from(response);
  }
}
```

🧠 4. Repository Layer

File: `data/repositories/user_repository.dart`

```
import '../models/user_model.dart';
import '../services/supabase_service.dart';

class UserRepository {
  final SupabaseService supabase;

  UserRepository(this.supabase);

  Future<List<UserModel>> fetchUsers() async {
    final data = await supabase.fetchUsers();
    return data.map((e) => UserModel.fromJson(e)).toList();
  }
}
```

🧩 5. User Model

File: `data/models/user_model.dart`

```
class UserModel {
  final String id;
  final String name;
  final String email;

  UserModel({required this.id, required this.name, required this.email});

  factory UserModel.fromJson(Map<String, dynamic> json) {
    return UserModel(
      id: json['id'].toString(),
      name: json['name'] ?? '',
      email: json['email'] ?? '',
    );
  }
}
```

🔧 6. Controller (Notifier)

File: `presentation/controllers/user_controller.dart`

```
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../../data/models/user_model.dart';
import '../../data/repositories/user_repository.dart';

class UserController extends AutoDisposeAsyncNotifier<List<UserModel>> {
  late final UserRepository _repo;

  @override
  Future<List<UserModel>> build() async {
    _repo = ref.read(userRepoProvider);
    return fetchUsers();
  }

  Future<List<UserModel>> fetchUsers() async {
    try {
      final users = await _repo.fetchUsers();
      state = AsyncValue.data(users);
      return users;
    } catch (e, st) {
      state = AsyncValue.error(e, st);
      rethrow;
    }
  }
}
```

🧩 7. Providers

File: `presentation/providers/user_providers.dart`

```
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../../data/services/supabase_service.dart';
import '../../data/repositories/user_repository.dart';
```

```
import '../controllers/user_controller.dart';

// Supabase Service
final supabaseServiceProvider = Provider<SupabaseService>(
  (ref) => SupabaseService(),
);

// Repository
final userRepoProvider = Provider<UserRepository>(
  (ref) => UserRepository(ref.read(supabaseServiceProvider)),
);

// Controller
final userControllerProvider =
    AutoDisposeAsyncNotifierProvider<UserController, List<UserModel>>(
  UserController.new,
);
```

🖥️ 8. UI Screen

File: `presentation/screens/user_screen.dart`

```
import 'package:flutter/material.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';
import '../providers/user_providers.dart';

class UserScreen extends ConsumerWidget {
  const UserScreen({super.key});

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final usersState = ref.watch(userControllerProvider);

    return Scaffold(
      appBar: AppBar(title: const Text('Supabase Users')),
      body: usersState.when(
        data: (users) => ListView.builder(
          itemCount: users.length,
          itemBuilder: (_, i) => ListTile(
            title: Text(users[i].name),
            subtitle: Text(users[i].email),
          ),
        ),
        loading: () => const Center(child: CircularProgressIndicator()),
        error: (err, _) => Center(child: Text('Error: $err')),
      ),
    );
  }
}
```

🚀 9. Main Entry

File: `main.dart`

```
import 'package:flutter/material.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart';
```

```
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'presentation/screens/user_screen.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await dotenv.load(fileName: ".env");

  runApp(const ProviderScope(child: MyApp()));
}

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      debugShowCheckedModeBanner: false,
      home: UserScreen(),
    );
  }
}
```

📊 Data Flow Summary

```
UI (ConsumerWidget)
   ↓
UserController (Notifier)
   ↓
UserRepository
   ↓
SupabaseService
   ↓
Supabase API (via URL + Anon Key)
```

🗒️ Rules Summary

Layer	File	Role
`.env`	`.env`	Stores Supabase URL + anon key
`EnvConfig`	core/config/env_config.dart	Reads environment safely
`SupabaseService`	data/services/supabase_service.dart	Handles API calls
`Repository`	data/repositories/user_repository.dart	Converts to models
`Controller`	presentation/controllers/user_controller.dart	State management
`Providers`	presentation/providers/user_providers.dart	Dependency graph
`UI`	presentation/screens/user_screen.dart	Displays data

✅ **Benefits of using `.env`:**

- Keeps your Supabase keys out of source code
- Allows switching between **dev/staging/prod** easily
- Clean, scalable, and secure for production use