MVVM (model - View - View Model)

• Ui geliştirmede kullanılır özellikle

Model → uygulamanın veri katmanıdır. Genellikle veri tabanından gelen verileri temsil eder. (tüm veri işlemlerini gerçekleştirir ve ViewModel ile etkileşimdedir.

View → ui, veri gösterir ve kullanıcı girdilerini alır. iş mantığı içermez kullanıcı girdilerini viewModele iletir ve ardaki verile bağlı olarak ui'yi günceller

ViewModel → Ara köprü, kullanıcı etkileşimiyle veriyi günceller. ve View'e gerekli değişiklikleri bildirir. Bu katmanı komutlar ve veri bağlama (data binding) işlevlerini sağlar. (durumu state'i yönetir) (verileri ve iş mantığını içerir) (stateManagement aracılığıyla view'e veri sağlar

Flutter'da → arayüz widgetlar oluşturur, State Management ile ViewModel işlevlerini yerine getirir.

model → genellikle Json API'lerden veri çekmek veya yerel veri tabanıyla (SQLite) etkileşimde bulunmak için kullanılır. Modelde veri doğrulama ve iş mantığı bulunur.

→ genellikle "Entity" sınıfları (user, product) veya servisler (ApiService gibi) ile temsil edilir. viewmodel'den bağımsızdır.

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

  userModel({required this.name, required this.email});
  factory UserModel.fromJson(Map<String, dynamic> Json) {
    return UserModel(
        name: Json['name'],
        email: Json['email'],
    );
  }
}
```

→ constructer

→ factory constructer Jsondan userModel oluşturmak için

View → statelessWidget veya statefulWidget sınıfları aracılığıyla yazılır. arayüz viewmodeldeki verilere göre şekillenir. Bunun için StreamBuilder, FutureBuilder veya state Management yöntemlerinden biri kullanılır.

```
class UserView extends statelessWidget {
  final UserModel user;
  userView({required this.user});
  @override
  Widget build(BuildContext context) {
```

```
return Column(
  children: [
  Text('Name: ${user.name}'),
  Text('Email: ${user.email}'),
  ],
);
}
```

~~V [illegible] (mode - Model~~

ViewModel → State Management çözümüyle uygulanır. Popüler çözümler arasında Providor, Riverpod, Bloc, Getx, gibi araçlar bulunur.

provider kullanılarak → ViewModel

```
class UserViewModel with ChangeNotifier {
  UserModel? _user;
  UserModel get user => _user;

  Future<void> fetchUser() async {
    //örneğin, bir API'den veri çekiyoruz
    final response = await http.get(Uri.parse('https://api.example.com/user'));
    if (response.statusCode = 200) {
      _user = UserModel.fromJson(JsonDecode(response.body));
      notifyListeners(); //View'ı günceller
    }
  }
}
```

ChangeNotifier → state değişikliklerini dinleyicilere bildirir

_user; → dışardan direkt erişilemez (private)

_user; → değişkene dışardan erişim sağlar (public geter)

(ok) durum kontrolü

→ Json verisini decode edip UserModele dönüştürme

JsonDecode → Json stringini Map'e çevirir

fromJson → Map'i UserModel nesnesine dönüştürür

notifyListeners → Dinleyicilere (UI) değişikliği bildirir. Bu çağrı, bu ViewModeli dinleyen widgetlar rebuild eder

çağrısı consumeri tetikleyerek UI'i günceller

State Management ile MVVM Uygulaması:

Provider ile MVVM →

```
class UserScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider(
      create: (context) => UserViewModel(),
      child: Scaffold(
        appbar: AppBar(title: Text("User Info")),
        body: Consumer<userViewModel>(
          builder: (context, viewModel, child) {
            if (viewModel.user == null) {
              return Center(child: CircularProgressIndicator());
            }
            return UserView(user: viewModel.user!);
          },
        ),
        floatingActionButton: FloatingActionButton(
          onPressed: () => context.read<UserViewModel>().fetchUser(),
          child: Icon(Icons.refresh),
        ),
      ),
    );
  }
}
```

StatelessWidget → stateless (durum yönetimi ViewModel'de)

ChangeNotifierProvider ile ViewModel'i widget ağacına sağlama

UserViewModel() → user ViewModel örneği oluşturma

Scaffold → temel sayfa yapısını sağlar

Consumer → Consumer, viewModeldeki değişiklikleri dinler

Consumer widget'i gerektiğinde rebuild olur

CircularProgressIndicator → kullanıcı verisi yüklenmemişse yükleme göstergesi göster

UserView → veri hazırsa UserView widget'ını göster

FloatingActionButton → yenileme butonu

ile widgetlar arasında veri/state paylaşımı yapılır

fetchUser() → tıklandığında ViewModel'deki fetchUser metodunu çağırır

Icons.refresh → yenileme ikonu