Jetpack Compose ile Android Uygulama Geliştirme Kursu

Jetpack Uygulama Mimarisi

Kasım ADALAN
Elektronik ve Haberleşme Mühendisi
Android - IOS Developer and Trainer

Eğitim İçeriği

- MVVM
- ViewModel
- LiveData
- Repository

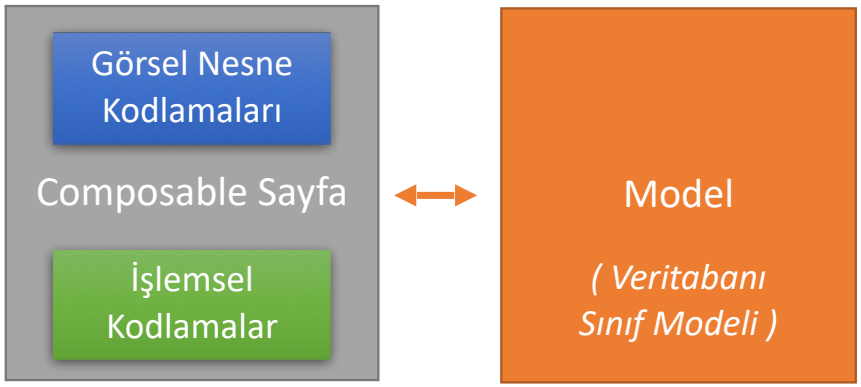
MVVM

MVVM (Model – View – ViewModel) Mimarisi

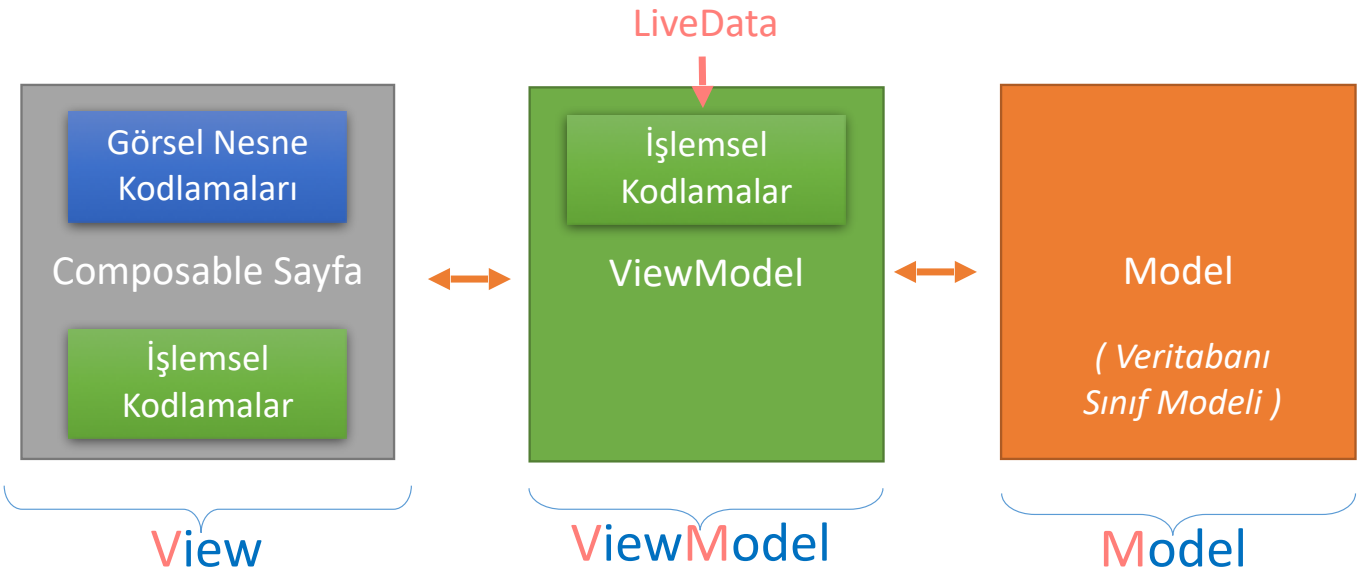
- Bu mimari sayesinde daha profesyonel kodlamalar yapabiliriz.
- Özellikle büyük projelerde oluşan kodlama karmaşasını azaltır.
- Mimarinin amacı proje içerisindeki işlemsel kodları ve tasarımsal kodları ayırıp dah modüller ve düzenli hale getirmektir.
- Bu mimari kullanımını öğrenmek için mimarinin temel yapılarını öğrenmemiz gereklidir.
- Başlıca yapılar ; ViewMode,LiveData ve DataBinding'dir.
- ViewModel yapısı kullanımı aslında MVVM mimarisinin temel alt yapısını oluşturur.
- Temel alt yapıya DataBinding yapısı eklenirse daha profesyonel ve düzenli bir MVVM mimarisi olmuş olur.
- DataBinding,Activity içinde görsel nesne tanımlamalarını azaltır.
- ViewModel,Activity içinde işlemsel kodlamaları azaltır.
- Her ikiside aynı modeli kullanırlar ve kompact bir yapı olmuş olur.
- LiveData,ViewModel içerisinde kullanılır ve ViewModel kullanımının yeteğini artırır.

MVVM (Model – View – ViewModel) Mimarisi Süreci

Klasik Yöntem

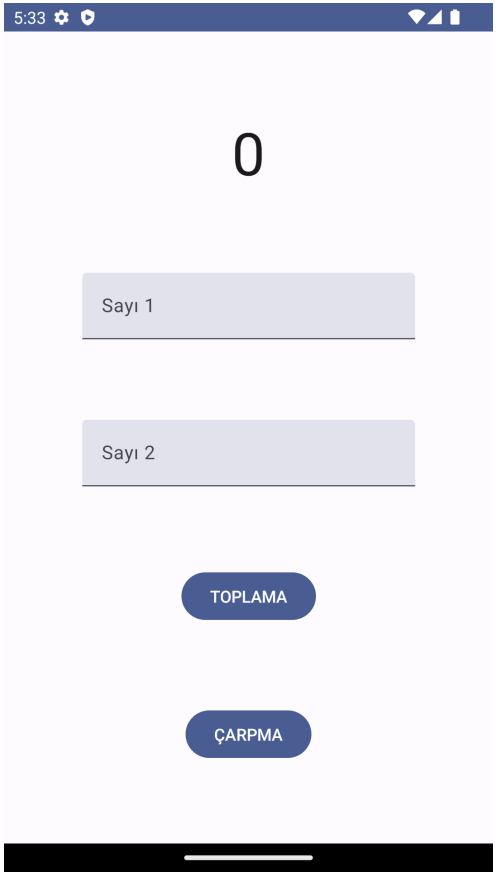


MVVM



Klasik Yöntem

```
@SuppressLint("UnusedMaterial3ScaffoldPaddingParameter")
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun Sayfa() {
    val tfSayi1 = remember { mutableStateOf( value: "") }
    val tfSayi2 = remember { mutableStateOf( value: "") }
    val sonuc   = remember { mutableStateOf( value: "0") }
    Column(modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.SpaceEvenly,
        horizontalAlignment = Alignment.CenterHorizontally) {

        Text(text = sonuc.value,fontSize = 50.sp)

        TextField(
            value = tfSayi1.value,
            onValueChange = {tfSayi1.value = it},
            label = { Text(text = "Sayı 1")})

        TextField(
            value = tfSayi2.value,
            onValueChange = {tfSayi2.value = it},
            label = { Text(text = "Sayı 2")})

        Button(onClick = {
            val alinanSayi1 = tfSayi1.value
            val alinanSayi2 = tfSayi2.value

            val sayi1 = alinanSayi1.toInt()
            val sayi2 = alinanSayi2.toInt()

            val toplam = sayi1 + sayi2

            sonuc.value = toplam.toString()
        }) { Text(text = "TOPLAMA") }

        Button(onClick = {
            val alinanSayi1 = tfSayi1.value
            val alinanSayi2 = tfSayi2.value

            val sayi1 = alinanSayi1.toInt()
            val sayi2 = alinanSayi2.toInt()

            val carpma = sayi1 * sayi2

            sonuc.value = carpma.toString()
        }) { Text(text = "ÇARPMA") }
    }
}
```



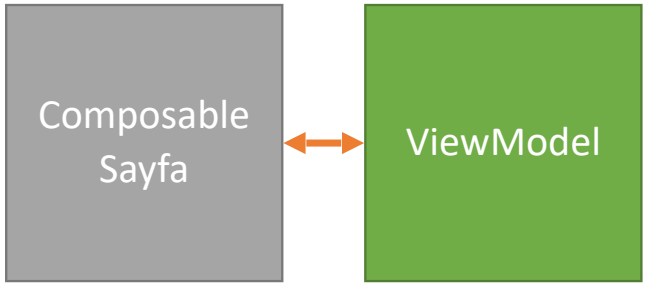
ViewModel

ViewModel

- Viewmodel'in asıl amacı arayüzü besleyecek verileri organize etmektir.
- Ayrıca sayfa rotasyonlarında (Dikey ve Yatay konuma geçişte) veriyi korumaktadır.
- Arayüzü (Activity) verilerden ayırarak daha kontrollü yapı oluşturabiliriz.
- ViewModel içinde sayfa üzerinde yapılacak işlemleri bulundurabiliriz.
- Activity'de verileri arayüze aktarma işlemi yaparız.
- View Modelde veritabanından veri alma , arayüzde matematiksel işlem vb. gibi şeyler yapılır.
- Verilerdeki değişimleri gözlemlemek için *LiveData* yapısını kullanabiliriz.
- DataBinding ile karışmaması lazım , Data binding tasarım alanına verileri kolay şekilde aktarmamızı sağlar.
- View Model ise Activity içinde yapılan arayüz ile ilgili veri işlemlerini kontrol eder.
- Genel kullanım her sayfaya özgü ViewModel oluşturmaktır.

Not : ViewModel aslında MVVM mimarisinin alt yapısıdır.Tek eksik DataBinding yapısıdır onuda ViewModele entegre edebilirsek MVVM mimarisi oluşmuş olur.

Ama çoğu kaynak DataBinding yapısı olmadan da MVVM olabileceğini söylemektedir.



Neler Yapamaz

- Toast,SnackBar,Alert gösteremez.
- Sayfa geçişi için intent kullanılamaz.
- Görsel nesne ile ilgili işlemler olmaz.

ViewModel Tanımlası İçin Kurulum

```
dependencies {
    implementation 'androidx.core:core-ktx:1.6.0'
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation "androidx.compose.ui:ui:$compose_version"
    implementation "androidx.compose.material:material:$compose_version"
    implementation "androidx.compose.ui:ui-tooling-preview:$compose_version"
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.3.1'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
    androidTestImplementation "androidx.compose.ui:ui-test-junit4:$compose_version"
    debugImplementation "androidx.compose.ui:ui-tooling:$compose_version"
    //View Model
    implementation "androidx.navigation:navigation-compose:2.4.0-alpha02"
}
```

```
implementation "androidx.navigation:navigation-compose:2.4.0-alpha02"
```

ViewModel Sınıfı Oluşturma

- Arayüzde kullanacağımız veriyi bu sınıfta tanımlarız ve yönetiriz.

Yönetilecek ve arayüzde kullanılacak veri

Yönettiğimiz değere matematiksel işlemi aktardık.

Yönettiğimiz değere matematiksel işlemi aktardık.

ViewModel özelliği aktarılır.

```
class SayfaViewModel:ViewModel() {
    var sonuc = "0"

    fun toplamaYap(alinanSayi1:String,alinanSayi2:String){
        val sayi1 = alinanSayi1.toInt()
        val sayi2 = alinanSayi2.toInt()
        val toplam = sayi1 + sayi2
        sonuc = toplam.toString()
    }

    fun carpmaYap(alinanSayi1:String,alinanSayi2:String){
        val sayi1 = alinanSayi1.toInt()
        val sayi2 = alinanSayi2.toInt()
        val carpma = sayi1 * sayi2
        sonuc = carpma.toString()
    }
}
```

Buradaki metodların amacı arayüzü değiştirecek veya besleyecek verilere değer üretmektir.
Bundan dolayı return demeden yönettiğimiz veriye elde ettiğimiz değer aktarılır.
Aktarılma işlemi olduğunda yönettiğimiz verinin içeriği değişmiş olur.

ViewModel Kurulum

```
@Composable
fun Sayfa() {
    val tfSayi1 = remember { mutableStateOf( value: "") }
    val tfSayi2 = remember { mutableStateOf( value: "") }
    val viewModel: SayfaViewModel = viewModel()
    val sonuc  = remember { mutableStateOf( value: "0") }

    Column(modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.SpaceEvenly,
        horizontalAlignment = Alignment.CenterHorizontally) {
        Text(text = sonuc.value,fontSize = 50.sp)
        TextField(
            value = tfSayi1.value,
            onValueChange = {tfSayi1.value = it},
            label = { Text(text = "Sayı 1")})
```

ViewModel Kullanımı

View model içinde yönettiğimiz veriye hesaplanan değeri aktarmak için View model içindeki metod çalıştırılır.

Bu metod yönettiğimiz değerin içeriğini değiştirir.

```
@Composable
fun Sayfa() {
    val tfSayi1 = remember { mutableStateOf( value: "") }
    val tfSayi2 = remember { mutableStateOf( value: "") }
    val viewModel: SayfaViewModel = viewModel()
    val sonuc  = remember { mutableStateOf( value: "0") }

    Column(modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.SpaceEvenly,
        horizontalAlignment = Alignment.CenterHorizontally) {
        Text(text = sonuc.value,fontSize = 50.sp)
        TextField(
            value = tfSayi1.value,
            onValueChange = {tfSayi1.value = it},
            label = { Text(text = "Sayı 1")})
        TextField(
            value = tfSayi2.value,
            onValueChange = {tfSayi2.value = it},
            label = { Text(text = "Sayı 2")})
        Button(onClick = {
            viewModel.toplamaYap(tfSayi1.value,tfSayi2.value)
            sonuc.value = viewModel.sonuc
        }) { Text(text = "TOPLAMA") }
        Button(onClick = {
            viewModel.carpmaYap(tfSayi1.value,tfSayi2.value)
            sonuc.value = viewModel.sonuc
        }) { Text(text = "ÇARPMA") }
    }
}
```

Yukardaki metod sayesinde yönettiğimiz değerin içeriğinin değişmiş en son halini alırız ve arayüzde gösteririz.

View Model İçinde Context Kullanımı

ViewModel içinde veritabanı gibi çalışmalarda context gerekli olabilir.

1

```
class MainActivityViewModelFactory(private val application: Application)
    : ViewModelProvider.NewInstanceFactory() {

    override fun <T : ViewModel?> create(modelClass: Class<T>): T {
        return MainActivityViewModel(application) as T
    }

}
```

2

```
class MainActivityViewModel(application: Application) : AndroidViewModel(application) {

    private val kdaor = KisilerDaoRepository(application)
    var kisilerListesi = MutableLiveData<List<Kisiler>>()
```

3

```
class MainActivity : AppCompatActivity(),SearchView.OnQueryTextListener {
    private lateinit var adapter:KisilerAdapter
    private val viewModel: MainActivityViewModel by viewModels() {MainActivityViewModelFactory(application)}
```

LiveData

LiveData

- View Model ile kullanılır.
- LiveData View Model‘in kullanımı kolaylaştırır.
- Temelde yaptığı işlem ViewModel içinde yönetilen verinin tetiklenmesini sağlamak ve değişimi dinlemektir.
- Bu tetikleme ve değişimi dinleme işlemi kodlama yapımızı sadeleştirir.

- Örnek : Textview içeriği çok fazla yerde değiştiriliyorsa , normalde her değişim kodlamasında textview kodlamasını yazmamız gerekir.

- LiveData sayesinde dinleme yapısı kurularak tek bir textview tanımlaması yapılarak , değişen değeri dinleriz ve değeri textview içeriğine aktarırız.

- Bu işlem gereksiz textview kodlamalarından bizi kurtarır.

Kütüphane Kurulumu

```
dependencies {
    implementation 'androidx.core:core-ktx:1.6.0'
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation "androidx.compose.ui:ui:$compose_version"
    implementation "androidx.compose.material:material:$compose_version"
    implementation "androidx.compose.ui:ui-tooling-preview:$compose_version"
    implementation 'androidx.lifecycle:lifecycle-runtime-ktx:2.3.1'
    implementation 'androidx.activity:activity-compose:1.3.1'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
    androidTestImplementation "androidx.compose.ui:ui-test-junit4:$compose_version"
    debugImplementation "androidx.compose.ui:ui-tooling:$compose_version"
    //View Model
    implementation "androidx.navigation:navigation-compose:2.4.0-alpha02"
    //Live data
    implementation "androidx.compose.runtime:runtime-livedata:1.0.0-beta08"
}
```

```
implementation "androidx.compose.runtime:runtime-livedata:1.0.0-beta08"
```

LiveData – ViewModel içinde Kullanımı

- ViewModel yapısını güçlendirmek için kullanılır.

Yönetilecek veri live data türüne dönüştürülür.

Verimize varsayılan değer atanır. Atama işlemi constructor ile yapılır. Bu değer atandığı anda dinleme işlemini tetikler.

```
class SayfaViewModel:ViewModel() {
    var sonuc = MutableLiveData<String>()

    init {
        sonuc = MutableLiveData<String>( value: "0")
    }

    fun toplamaYap(alinanSayi1:String,alinanSayi2:String){
        val sayi1 = alinanSayi1.toInt()
        val sayi2 = alinanSayi2.toInt()
        val toplam = sayi1 + sayi2
        sonuc.value = toplam.toString()
    }

    fun carpmaYap(alinanSayi1:String,alinanSayi2:String){
        val sayi1 = alinanSayi1.toInt()
        val sayi2 = alinanSayi2.toInt()
        val carpma = sayi1 * sayi2
        sonuc.value = carpma.toString()
    }
}
```

value metodu ile hesaplama sonucunda oluşan değeri verimize atadık ve dinleme işlemini tetikledik.

LiveData – Sayfa İçinde Kullanımı

ViewModel içindeki LiveData verimizi sayfa içinde kullanmak için state özelliği olan değişkene dönüştürüyoruz ve varsayılan değer veriyoruz.

```
@Composable
fun Sayfa() {
    val tfSayi1 = remember { mutableStateOf( value: "") }
    val tfSayi2 = remember { mutableStateOf( value: "") }
    val viewModel: SayfaViewModel = viewModel()
    val sonuc  = viewModel.sonuc.observeAsState( initial: "0")

    Column(modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.SpaceEvenly,
        horizontalAlignment = Alignment.CenterHorizontally) {
        Text(text = sonuc.value,fontSize = 50.sp)
        TextField(
            value = tfSayi1.value,
            onValueChange = {tfSayi1.value = it},
            label = { Text(text = "Sayı 1")})
        TextField(
            value = tfSayi2.value,
            onValueChange = {tfSayi2.value = it},
            label = { Text(text = "Sayı 2")})
        Button(onClick = {
            viewModel.toplamaYap(tfSayi1.value,tfSayi2.value)
        }) { Text(text = "TOPLAMA") }
        Button(onClick = {
            viewModel.carpmaYap(tfSayi1.value,tfSayi2.value)
        }) { Text(text = "ÇARPMA") }
    }
}
```

value metodu ile veride değişim olduğu anda en son değer buraya gelir ve arayüzde kullanılabilir.

Bu metod ile ViewModel içindeki verimizin içeriğini bu metodlar ile değiştirebiliriz ve ViewModel içinde value metodu olduğu için dinleme işlemini tetikler.

Repository

Repository Sınıfı

- Ortak kullanım için oluşturduğumuz metodların yer aldığı sınıftır.
- Veritabanı işlemlerinde bazı metodları bir çok sayfa kullanabilir.
- Ortak erişebilecek bir sınıf oluşturup kodlama tekrarlarını azaltırız.
- Aslında dao (Database access object) sınıfıdır.
- Alt yapısı View Model örnek alınarak oluşturulur.

Repository Oluşturma

Yönetilecek veri ve varsayılan değer

Yönetilen veriye erişmek için metod.

Yönetilen verinin içeriğini değiştirme

```
class MatematikRepository {
    var matematikselSonuc = MutableLiveData<String>()

    init {
        matematikselSonuc = MutableLiveData<String>( value: "0")
    }

    fun matematikselSonucuGetir():MutableLiveData<String>{
        return matematikselSonuc
    }

    fun topla(alinanSayi1:String,alinanSayi2:String){
        val sayi1 = alinanSayi1.toInt()
        val sayi2 = alinanSayi2.toInt()
        val toplam = sayi1 + sayi2
        matematikselSonuc.value = toplam.toString()
    }

    fun carp(alinanSayi1:String,alinanSayi2:String){
        val sayi1 = alinanSayi1.toInt()
        val sayi2 = alinanSayi2.toInt()
        val carpma = sayi1 * sayi2
        matematikselSonuc.value = carpma.toString()
    }
}
```

ViewModel İçinde Kullanımı

```
class SayfaViewModel:ViewModel() {
    var sonuc = MutableLiveData<String>()
    var mrepo = MatematikRepository()

    init {
        sonuc = mrepo.matematikselSonucuGetir()
    }

    fun toplamaYap(alinanSayi1:String,alinanSayi2:String){
        mrepo.topla(alinanSayi1,alinanSayi2)
    }

    fun carpmaYap(alinanSayi1:String,alinanSayi2:String){
        mrepo.carp(alinanSayi1,alinanSayi2)
    }
}
```

Repository sınıfına erişmek için nesne

ViewModel çalıştığı anda repo dan veriyi alır ve arayüzde gösterir.

Repository sınıfındaki metodu çalıştırır ve repo içindeki değer değişir , değişen değeri matematikselSonucuGetir metodu view modele aktarır.

Sayfa İçinde Kullanımı

ViewModel içindeki LiveData verimizi sayfa içinde kullanmak için state özelliği olan değişkene dönüştürüyoruz ve varsayılan değer veriyoruz.

```
@Composable
fun Sayfa() {
    val tfSayi1 = remember { mutableStateOf( value: "") }
    val tfSayi2 = remember { mutableStateOf( value: "") }
    val viewModel: SayfaViewModel = viewModel()
    val sonuc  = viewModel.sonuc.observeAsState( initial: "0")

    Column(modifier = Modifier.fillMaxSize(),
        verticalArrangement = Arrangement.SpaceEvenly,
        horizontalAlignment = Alignment.CenterHorizontally) {
        Text(text = sonuc.value,fontSize = 50.sp)
        TextField(
            value = tfSayi1.value,
            onValueChange = {tfSayi1.value = it},
            label = { Text(text = "Sayı 1")})
        TextField(
            value = tfSayi2.value,
            onValueChange = {tfSayi2.value = it},
            label = { Text(text = "Sayı 2")})
        Button(onClick = {
            viewModel.toplamaYap(tfSayi1.value,tfSayi2.value)
        }) { Text(text = "TOPLAMA") }
        Button(onClick = {
            viewModel.carpmaYap(tfSayi1.value,tfSayi2.value)
        }) { Text(text = "ÇARPMA") }
    }
}
```

value metodu ile veride değişim olduğu anda en son değer buraya gelir ve arayüzde kullanılabilir.

Bu metod ile ViewModel içindeki verimizin içeriğini bu metodlar ile değiştirebiliriz ve ViewModel içinde value metodu olduğu için dinleme işlemini tetikler.

Teşekkürler...

kasım-adalan

kasimadalan@gmail.com

kasimadalan