

MVVM

Материалы



github.com/adamxrvn/hse-lyceum-android-course

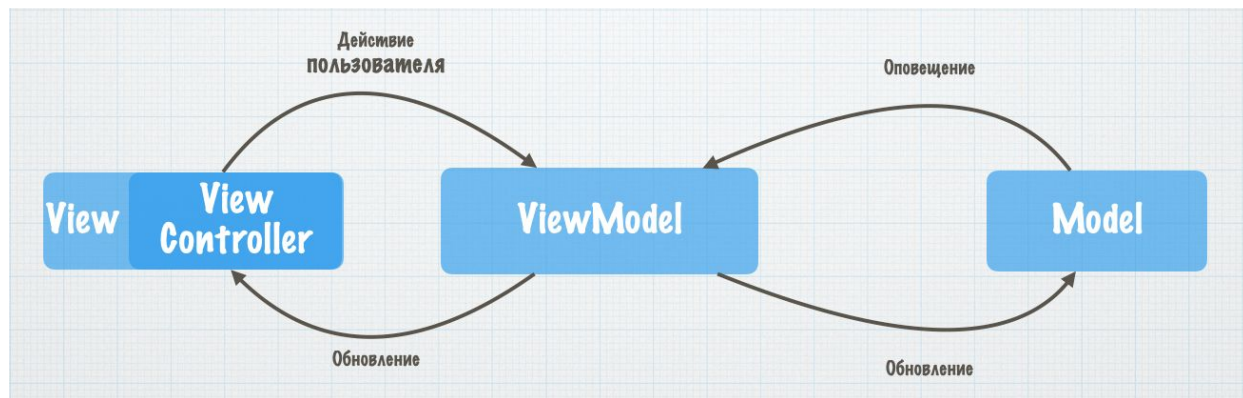
Что такое MVVM?

MVVM — это паттерн разработки, позволяющий разделить приложение на три функциональные части:

Model — основная логика программы (работа с данными, вычисления, запросы и так далее).

View — вид или представление (пользовательский интерфейс).

ViewModel — модель представления, которая служит прослойкой между View и Model.



Задача на сегодня

4:42

Login/Signup

Username

Password

Login

Signup

4:43

Login/Signup

User already exists


Username
1


Password
•


Login


Signup


4:43


 Classic Italian
290 p.


 Turkey Ranch & Swiss
400 p.


 Veggie Delight
52 p.


 Tuna Melt
600 p.


 Chicken Pesto
550 p.


 Roast Beef
490 p.

 Tower Sandwich
320 p.

 Water Sandwich
777 p.

 Sandwich Deluxe
300 p.

 Club Sandwich
100 p.

 Pastrami

Model

```
data class Sandwich(val id: Int,  
                    val name: String,  
                    val price: Int,  
                    val icon: String)
```

Импорты

// Для REST API

```
implementation("com.squareup.retrofit2:retrofit:2.9.0")
```

```
implementation("com.squareup.retrofit2:converter-gson:2.9.0")
```

// Для загрузки картинок с сервера

```
implementation("io.coil-kt:coil-compose:2.0.0-rc01")
```

// ViewModel и LiveData для MVVM

```
implementation ("androidx.lifecycle:lifecycle-viewmodel-compose:2.7.0")
```

```
implementation ("androidx.lifecycle:lifecycle-livedata:2.7.0")
```

```
implementation ("androidx.navigation:navigation-compose:2.7.7")
```

```
implementation("com.squareup.okhttp3:okhttp:4.11.0")
```

```
implementation("com.squareup.okhttp3:logging-interceptor:4.11.0")
```

ApiService

```
interface ApiService {  
    // Authorization  
  
    @POST("/signup")  
    suspend fun signup(@Body user: User): Message  
  
    @FormUrlEncoded  
    @POST("/token")  
    suspend fun login(  
        @Field("username") username: String,  
        @Field("password") password: String  
    ): Token  
  
    // Sandwiches  
  
    @GET("/sandwiches")  
    suspend fun getSandwiches(@Header("Authorization") token: String): List<Sandwich>  
  
    @POST("/sandwiches")  
    suspend fun addSandwich(  
        @Body sandwich: Sandwich,  
        @Header("Authorization") token: String  
    ): Sandwich  
  
    @PUT("/sandwiches/{id}")  
    suspend fun updateSandwich(  
        @Path("id") id: Int,  
        @Header("Authorization") token: String  
    ): Sandwich  
  
    @DELETE("/sandwiches/{id}")  
    suspend fun deleteSandwich(@Path("id") id: Int, @Header("Authorization") token: String)  
}
```

RetrofitInstance

```
object RetrofitInstance {  
  
    // Приватная константа, хранящая базовый URL-адрес сервера  
    private const val BASE_URL = "https://hse-sandwich-a47b9a61e68b.herokuapp.com/"  
  
    // Ленивое создание экземпляра ApiService  
    // Этот блок кода выполнится только при первом обращении к переменной api  
    val api: ApiService by lazy {  
  
        val interceptor = HttpLoggingInterceptor().apply { this: HttpLoggingInterceptor  
            this.level = HttpLoggingInterceptor.Level.BODY  
        }  
        val client = OkHttpClient.Builder().apply { this: OkHttpClient.Builder  
            this.addInterceptor(interceptor)  
            // time out setting  
            .connectTimeout( timeout: 3, TimeUnit.SECONDS)  
            .readTimeout( timeout: 20, TimeUnit.SECONDS)  
            .writeTimeout( timeout: 25, TimeUnit.SECONDS)  
        }.build()  
        // Создание объекта Retrofit  
        val retrofit = Retrofit.Builder() Retrofit.Builder  
            .baseUrl(BASE_URL) // Установка базового адреса сервера  
            .addConverterFactory(GsonConverterFactory.create()) // Добавление конвертера из JSON в объекты Kotlin  
            .client(client) Retrofit.Builder  
            .build() // Построение объекта Retrofit  
        // Создание реализации ApiService из объекта Retrofit  
        retrofit.create(ApiService::class.java) ^lazy  
    }  
}
```


AuthViewModel

```
class AuthViewModel : ViewModel() {

    private val apiService = RetrofitInstance.api

    val message: MutableState<Message> = mutableStateOf(Message(""))
    val error: MutableState<String> = mutableStateOf( value: "")

    val token: MutableState<Token> = mutableStateOf(Token( access_token: "", token_type: ""))

    fun signup(username: String, password: String) {
        viewModelScope.launch { this: CoroutineScope
            try {
                val response = apiService.signup(User(username, password))
                message.value = response
            } catch (e: Exception) {
                error.value = e.message.toString()
            }
        }
    }

    fun login(username: String, password: String) {
        viewModelScope.launch { this: CoroutineScope
            try {
                val response = apiService.login(username, password)
                token.value = response
            } catch (e: Exception) {
                error.value = e.message.toString()
            }
        }
    }
}
```

SandwichViewModel

```
class SandwichViewModel : ViewModel() {  
    // Получение экземпляра ApiService через синглтон RetrofitInstance для выполнения сетевых запросов  
    private val apiService = RetrofitInstance.api  
  
    // Состояние списка сэндвичей, обёрнутое в MutableState для наблюдения изменений в Jetpack Compose  
    val sandwiches: MutableState<List<Sandwich>> = mutableStateOf(emptyList())  
  
    // Функция для получения списка сэндвичей с сервера  
    fun getSandwiches(token: String) {  
        // Запуск корутины в области видимости ViewModel  
        viewModelScope.launch { this: CoroutineScope  
            try {  
                // Попытка получить список сэндвичей через сетевой запрос  
                val response = apiService.getSandwiches( token: "Bearer " + token)  
  
                // Проверка, что полученный ответ не пустой  
                if (response.isNotEmpty()) {  
                    // Обновление состояния списка сэндвичей, если данные получены  
                    sandwiches.value = response  
                }  
            } catch (e: Exception) {  
                // Обработка ошибок сетевого запроса или преобразования данных  
                // Здесь можно добавить логирование или показать пользовательское сообщение об ошибке  
            }  
        }  
    }  
}
```

View - Login - 1

```
@Composable
fun LoginScreen(viewModel: AuthViewModel, navController: NavController) {
    var email by remember { mutableStateOf( value: "" ) }
    var password by remember { mutableStateOf( value: "" ) }

    val token = viewModel.token.value

    LaunchedEffect(token) { this: CoroutineScope
        if (token.access_token.isNotEmpty()) {
            // If token is not empty, navigate to the main activity screen
            // and remove login page from backstack
            navController.navigate( route: "mainActivity" ) { this: NavOptionsBuilder
                popUpTo( route: "login" ) { inclusive = true }
            }
        }
    }

    Column(modifier = Modifier.padding(16.dp), verticalArrangement = Arrangement.Center) { this: ColumnScope
        Text(text = "Login/Signup", style = MaterialTheme.typography.titleLarge)
        Text(text = viewModel.message.value.message, style = MaterialTheme.typography.bodyMedium)

        Spacer(modifier = Modifier.height(16.dp))
        OutlinedTextField(
            modifier = Modifier
                .fillMaxWidth()
                .padding(10.dp, 0.dp),
            value = email,
            onValueChange = { email = it },
            label = { Text( text: "Username" ) },
            keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Text)
        )
        Spacer(modifier = Modifier.height(8.dp))
        OutlinedTextField(
```

View - Login - 2

```
OutlinedTextField(  
    modifier = Modifier  
        .fillMaxWidth()  
        .padding(10.dp, 0.dp),  
    value = password,  
    onValueChange = { password = it },  
    label = {  
        Text(text = "Password")  
    },  
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),  
    visualTransformation = PasswordVisualTransformation()  
)  
Spacer(modifier = Modifier.height(16.dp))  
Button(modifier = Modifier  
    .fillMaxWidth()  
    .padding(10.dp, 0.dp),  
    onClick = {  
        viewModel.login(email, password)  
    }) { this: RowScope  
    Text( text: "Login")  
}  
Spacer(modifier = Modifier.height(8.dp))  
Button(  
    onClick = {  
        viewModel.signup(email, password)  
    },  
    modifier = Modifier  
        .fillMaxWidth()  
        .padding(10.dp, 0.dp)  
) { this: RowScope  
    Text( text: "Signup")  
}
```

View - Sandwiches - 1

```
@Composable
fun SandwichScreen(viewModel: SandwichViewModel, token: String, navController: NavController) {
    // Подписка на изменения списка сэндвичей в ViewModel
    val sandwiches by viewModel.sandwiches

    // Создание вертикального списка
    LazyColumn { this: LazyListScope
        // Проходит по списку сэндвичей и создает для каждого элемента UI
        items(sandwiches) { this: LazyItemScope sandwich ->
            // Строка для размещения изображения и текста сэндвича
            Row(
                modifier = Modifier.fillMaxWidth(), // Занимает максимальную ширину
                verticalAlignment = Alignment.CenterVertically // Вертикальное выравнивание элементов
            ) { this: RowScope
                // Вывод изображения сэндвича
                Image(
                    painter = rememberAsyncImagePainter(sandwich.icon), // Загрузка изображения
                    contentDescription = null, // Описание контента (для доступности)
                    modifier = Modifier
                        .size(82.dp) // Размер изображения
                        .padding(8.dp) // Отступы вокруг изображения
                        .clip(RoundedCornerShape(8.dp)) // Скругленные углы изображения
                )

                // Колонка для названия и цены сэндвича
                Column { this: ColumnScope
                    // Название сэндвича
                    Text(
                        text = sandwich.name, // Текстовое содержимое
                        fontSize = 26.sp, // Размер шрифта
                        modifier = Modifier.padding(8.dp, 0.dp) // Отступы
                    )

                    // Цена сэндвича
                    Text(
                        text = "${sandwich.price} p.", // Текстовое содержимое
                        fontSize = 18.sp, // Размер шрифта
                    )
                }
            }
        }
    }
}
```

View - Sandwiches - 2

```
// Цена сэндвича
Text(
  text = "${sandwich.price} p.", // Текстовое содержимое
  fontSize = 18.sp, // Размер шрифта
  modifier = Modifier.padding(8.dp, 0.dp) // Отступы
)
}
}
// Разделитель между элементами списка
Divider()
}
}
// Эффект для однократного выполнения действий при первом рендеринге компонента
DisposableEffect(Unit) { this: DisposableEffectScope
  viewModel.getSandwiches(token) // Запрос списка сэндвичей у ViewModel
  onDispose {} ^DisposableEffect // Операции при удалении компонента из дерева композиции
}
}
```

MainActivity

```
class MainActivity : ComponentActivity() {

    private val viewModel: AuthViewModel by viewModels()
    private val sandwichViewModel: SandwichViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            HSELyceum2Theme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {

                    val navController = rememberNavController()
                    NavHost(navController = navController, startDestination = "login") { this: NavGraphBuilder
                        composable( route: "login") { this: AnimatedContentScope it: NavBackStackEntry
                            LoginScreen(viewModel = viewModel, navController = navController)
                        }
                        composable( route: "mainActivity") { this: AnimatedContentScope it: NavBackStackEntry
                            SandwichScreen(
                                sandwichViewModel,
                                navController = navController,
                                token = viewModel.token.value.access_token
                            )
                        }
                    }
                }
            }
        }
    }
}
```