МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа № 4**

*по дисциплине «Программирование мобильных устройств»*

Выполнил студен группы ПИбд-32

Преснякова В. В.

Проверил доцент кафедры

«Информационные системы»

Филиппов А.А.

Ульяновск, 2023

**Задание**

Архитектура приложения (MVVM). Необходимо:

1. Создать интерфейсы-репозитории для абстрагирования от реализации механизма хранения данных и их реализацию для работы с DAO, которые были разработаны в ЛР №3.

2. Создать ViewModel для реализации бизнес-логики.

3. Вынести всю бизнес-логику из представлений (UI) во ViewModel-классы.

4. Организовать взаимодействие между UI и слоем бизнес-логики.

5. Все списки элементов в UI должны поддерживать пагинацию.

6. Отчет и изменения проекта загрузить в репозиторий по адресу <http://student.git.athene.tech>

**Решение**

1. Создадим интерфейсы

interface OrderRepository {  
 suspend fun createOrder(order: Order): Long  
 suspend fun insertOrderHotel(orderHotel: OrderHotel)  
 suspend fun delete(order: Order)  
 fun getOrderWithHotels(id: Int): Flow<OrderWithHotels>  
 fun getAllOrder(): Flow<List<Order>>  
 fun getUserOrders(id: Int) : Flow<UserWithOrder>  
}

interface HotelRepository {  
 suspend fun insertHotel(hotel: Hotel)  
 suspend fun updateHotel(hotel: Hotel)  
 suspend fun deleteHotel(hotel: Hotel)  
 suspend fun getHotelById(id: Int): Hotel  
 fun getAllHotelsPaged(): PagingSource<Int, Hotel>  
 fun call(): Flow<PagingData<Hotel>>  
}

interface UserRepository {  
 suspend fun createUser(user: User)  
 suspend fun updateUser(user: User)  
 suspend fun deleteUser(user: User)  
 suspend fun getUserById(id: Int): User  
 suspend fun getUserByEmail(email: String): User  
}

Сделаем реализацию для них

class OrderRepoImpl(private val orderDao: OrderDao) : OrderRepository {  
  
 override suspend fun createOrder(order: Order): Long = orderDao.createOrder(order)  
  
 override suspend fun insertOrderHotel(orderHotel: OrderHotel) = orderDao.insertOrderHotel(orderHotel)  
  
 override suspend fun delete(order: Order) = orderDao.delete(order)  
  
 override fun getOrderWithHotels(id: Int): Flow<OrderWithHotels> = orderDao.getOrderWithHotels(id)  
  
 override fun getAllOrder(): Flow<List<Order>> = orderDao.getAllOrder()  
  
 override fun getUserOrders(id: Int): Flow<UserWithOrder> = orderDao.getUserOrders(id)  
}

class HotelRepoImpl(private val hotelDao: HotelDao) : HotelRepository {  
  
 override suspend fun insertHotel(hotel: Hotel) = hotelDao.insert(hotel)  
  
 override suspend fun updateHotel(hotel: Hotel) = hotelDao.update(hotel)  
  
 override suspend fun deleteHotel(hotel: Hotel) = hotelDao.delete(hotel)  
  
 override suspend fun getHotelById(id: Int): Hotel = hotelDao.getHotelById(id)  
 override fun getAllHotelsPaged(): PagingSource<Int, Hotel> = hotelDao.getAllHotelsPaged()  
 override fun call(): Flow<PagingData<Hotel>> {  
 return Pager(  
 PagingConfig(pageSize = 5)  
 ) **{** hotelDao.getAllHotelsPaged()  
 **}**.flow  
 }  
}

class UserRepoImpl(private val userDao: UserDao) : UserRepository {  
  
 override suspend fun createUser(user: User) = userDao.createUser(user)  
  
 override suspend fun updateUser(user: User) = userDao.updateUser(user)  
  
 override suspend fun deleteUser(user: User) = userDao.deleteUser(user)  
  
 override suspend fun getUserById(id: Int): User = userDao.getUserById(id)  
  
 override suspend fun getUserByEmail(email: String): User = userDao.getUserByEmail(email)  
}

1. Сделаем реализацию di контейнера для уменьшения связности
2. interface AppContainer {  
    val hotelRepo: HotelRepository  
    val userRepo: UserRepository  
    val orderRepo: OrderRepository  
   }

class AppDataContainer(private val context: Context) : AppContainer {  
 override val hotelRepo: HotelRepository by *lazy* **{** HotelRepoImpl(AppDatabase.getInstance(context).hotelDao())  
 **}** override val userRepo: UserRepository by *lazy* **{** UserRepoImpl(AppDatabase.getInstance(context).userDao())  
 **}** override val orderRepo: OrderRepository by *lazy* **{** OrderRepoImpl(AppDatabase.getInstance(context).orderDao())  
 **}**}

class App : Application() {  
 lateinit var container: AppContainer  
  
 override fun onCreate() {  
 super.onCreate()  
 container = AppDataContainer(this)  
 }  
}

1. Сделаем view модели
2. object AppViewModelProvider {  
    val Factory = *viewModelFactory* **{** *initializer* **{** HotelViewModel(*app*().container.hotelRepo)  
    **}** *initializer* **{** UserViewModel(*app*().container.userRepo)  
    **}** *initializer* **{** OrderViewModel(*app*().container.orderRepo, *app*().container.basketRepo)  
    **}  
    }**}  
     
   fun CreationExtras.app(): App =  
    (this[ViewModelProvider.AndroidViewModelFactory.APPLICATION\_KEY] as App)

class OrderViewModel(private val orderRepository: OrderRepository) : ViewModel() {  
  
 var selectedItem: Hotel? = null  
val rooms = *mutableStateOf*("")  
var dateFrom = *mutableStateOf*("")  
var dateTo = *mutableStateOf*("")  
  
fun deleteOrder(orderId: Int) = *viewModelScope*.*launch* **{** orderRepository.delete(orderId)  
**}**suspend fun getOrderList(id: Int) : Flow<List<Order>> {  
 return orderRepository.getUserOrders(id)  
}  
  
suspend fun getHotelFromOrder(id: Int) : Hotel {  
 return orderRepository.getHotelFromOrder(id)  
}  
  
fun createOrder() = *viewModelScope*.*launch* **{** Log.d("MyLog", GlobalUser.getInstance().getUser()?.userId.*toString*())  
  
 val order = Order(  
 dateFrom = dateFrom.value,  
 dateTo = dateTo.value,  
 rooms = rooms.value.*toInt*(),  
 total = getSubTotal(),  
 bookedHotelId = selectedItem?.hotelId!!,  
 creatorUserId = GlobalUser.getInstance().getUser()?.userId!!,  
 //hotel = selectedItem!!  
 )  
  
 val orderId = orderRepository.createOrder(order)  
  
  
 rooms.value = ""  
 selectedItem = null  
**}**fun getSubTotal(): Double {  
 return selectedItem!!.price \* rooms.value.*toInt*()  
}

class HotelViewModel(private val hotelRepository: HotelRepository): ViewModel() {  
 var name = *mutableStateOf*("")  
val price = *mutableStateOf*("")  
val location = *mutableStateOf*("")  
val stars = *mutableStateOf*("")  
val info = *mutableStateOf*("")  
val img = *mutableStateOf*(R.drawable.*img*)  
val HotelList = hotelRepository.getAllHotels()  
var hotel: Hotel? = null  
  
fun insertHotel() = *viewModelScope*.*launch* **{** val hotel = Hotel(  
 name = name.value,  
 location = location.value,  
 price = price.value.*toDouble*(),  
 img = img.value,  
 stars = stars.value.*toInt*(),  
 info = info.value  
 )  
 hotelRepository.insertHotel(hotel)  
**}**fun deleteHotel(hotel : Hotel) = *viewModelScope*.*launch* **{** hotelRepository.deleteHotel(hotel)  
**}**fun getHotelById(id: Int) = *viewModelScope*.*launch* **{** hotelRepository.getHotelById(id)  
**}**fun UpdateHotel(hotel: Hotel) = *viewModelScope*.*launch* **{** hotelRepository.updateHotel(hotel)  
**}**

class UserViewModel(private val userRepository: UserRepository): ViewModel() {  
  
 var name = *mutableStateOf*("")  
 val surname = *mutableStateOf*("")  
 val email = *mutableStateOf*("")  
 val password = *mutableStateOf*("")  
 fun createUser() = *viewModelScope*.*launch* **{** val user = User(  
 name = name.value,  
 surname = surname.value,  
 email = email.value,  
 password = password.value,  
 role = RoleEnum.*User* )  
 userRepository.createUser(user)  
 **}** fun authUser() = *viewModelScope*.*launch* **{** val user = userRepository.getUserByEmail(email.value)  
 if (password.value != "" && user.password == password.value) {  
 val globalUser = GlobalUser.getInstance()  
 globalUser.setUser(user)  
 }  
 **}** fun isValidEmail(email: String): Boolean {  
 return android.util.Patterns.*EMAIL\_ADDRESS*.matcher(email).matches()  
 }  
}

1. Теперь для связи с UI нам нужно просто передать в compose функцию нашу view модель, и можно вызывать методы которые реализованы там, вот так

@Composable  
fun BookingScreen(hotel: Hotel, navHostController: NavHostController, orderViewModel: OrderViewModel = viewModel(factory = AppViewModelProvider.Factory)) {  
 Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*background*(Color.White)  
 .*padding*(16.*dp*)  
 ,  
 verticalArrangement = Arrangement.Center,  
 horizontalAlignment = Alignment.CenterHorizontally  
 ) **{** Text(  
 text = "Booking",  
 fontSize = 24.*sp*,  
 fontWeight = FontWeight.Bold,  
 modifier = Modifier  
 .*padding*(16.*dp*)  
 )  
  
 TextField(  
 value = orderViewModel.rooms.value,  
 onValueChange = **{** orderViewModel.rooms.value = **it}**,  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*height*(50.*dp*)  
 .*padding*(16.*dp*, 0.*dp*)  
 .*border*(1.*dp*, Color.Gray, *RoundedCornerShape*(4.*dp*)),  
 singleLine = true,  
 keyboardOptions = KeyboardOptions(  
 keyboardType = KeyboardType.Text,  
 imeAction = ImeAction.Next  
 ),  
 keyboardActions = KeyboardActions(  
 onNext = **{  
  
 }** ),  
 placeholder = **{** Text(  
 text = "Rooms",  
 style = TextStyle(fontSize = 12.*sp*)  
 )  
 **}** )  
  
 Spacer(modifier = Modifier.*height*(16.*dp*))  
  
// Fetching the Local Context  
 val mContext = *LocalContext*.current  
  
 // Declaring integer values  
 // for year, month and day  
 val mYearFrom: Int  
 val mMonthFrom: Int  
 val mDayFrom: Int  
  
 // Initializing a Calendar  
 val mCalendarFrom = Calendar.getInstance()  
  
 // Fetching current year, month and day  
 mYearFrom = mCalendarFrom.get(Calendar.*YEAR*)  
 mMonthFrom = mCalendarFrom.get(Calendar.*MONTH*)  
 mDayFrom = mCalendarFrom.get(Calendar.*DAY\_OF\_MONTH*)  
  
 mCalendarFrom.*time* = Date()  
  
 // Declaring a string value to  
 // store date in string format  
 val mDateFrom = remember **{** *mutableStateOf*("") **}** // Declaring DatePickerDialog and setting  
 // initial values as current values (present year, month and day)  
 val mDatePickerDialogFrom = DatePickerDialog(  
 mContext,  
 **{** \_: DatePicker, mYear: Int, mMonth: Int, mDayOfMonth: Int **->** mDateFrom.value = "$mDayOfMonth/${mMonth+1}/$mYear"  
 **}**, mYearFrom, mMonthFrom, mDayFrom  
 )  
  
  
 // Creating a button that on  
 // click displays/shows the DatePickerDialog  
 Button(  
 colors = ButtonDefaults.buttonColors(  
 backgroundColor = (colorResource(id = R.color.*figma\_blue*)),  
 contentColor = Color.White  
 ),  
 onClick = **{** mDatePickerDialogFrom.show()  
  
 **}**,  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(16.*dp*, 16.*dp*, 16.*dp*, 0.*dp*)  
 .*height*(50.*dp*)  
 ) **{** Text("Open Date Picker")  
 **}** // Displaying the mDate value in the Text  
 Text(text = "Selected Date From: ${mDateFrom.value}", fontSize = 15.*sp*)  
  
  
 Spacer(modifier = Modifier.*height*(16.*dp*))  
  
  
// Fetching the Local Context  
  
 // Declaring integer values  
 // for year, month and day  
 val mYear: Int  
 val mMonth: Int  
 val mDay: Int  
  
 // Initializing a Calendar  
 val mCalendar = Calendar.getInstance()  
  
 // Fetching current year, month and day  
 mYear = mCalendar.get(Calendar.*YEAR*)  
 mMonth = mCalendar.get(Calendar.*MONTH*)  
 mDay = mCalendar.get(Calendar.*DAY\_OF\_MONTH*)  
  
 mCalendar.*time* = Date()  
  
 // Declaring a string value to  
 // store date in string format  
 val mDate = remember **{** *mutableStateOf*("") **}** // Declaring DatePickerDialog and setting  
 // initial values as current values (present year, month and day)  
 val mDatePickerDialog = DatePickerDialog(  
 mContext,  
 **{** \_: DatePicker, mYear: Int, mMonth: Int, mDayOfMonth: Int **->** mDate.value = "$mDayOfMonth/${mMonth+1}/$mYear"  
 **}**, mYear, mMonth, mDay  
 )  
  
  
 // Creating a button that on  
 // click displays/shows the DatePickerDialog  
  
 Button(  
 colors = ButtonDefaults.buttonColors(  
 backgroundColor = (colorResource(id = R.color.*figma\_blue*)),  
 contentColor = Color.White  
 ),  
 onClick = **{** mDatePickerDialog.show()  
  
 **}**,  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(16.*dp*, 16.*dp*, 16.*dp*, 0.*dp*)  
 .*height*(50.*dp*)  
 ) **{** Text("Open Date Picker")  
 **}** // Displaying the mDate value in the Text  
 Text(text = "Selected Date To: ${mDate.value}", fontSize = 15.*sp*,)  
  
  
 Button(  
 colors = ButtonDefaults.buttonColors(  
 backgroundColor = (colorResource(id = R.color.*figma\_blue*)),  
 contentColor = Color.White  
 ),  
 onClick = **{** if(GlobalUser.getInstance().getUser() != null){  
 orderViewModel.selectedItem = hotel  
 orderViewModel.dateFrom = mDateFrom  
 orderViewModel.dateTo = mDate  
 orderViewModel.createOrder()  
 navHostController.navigate("home")  
 }else{  
 navHostController.navigate("login")  
 }  
 **}**,  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(16.*dp*, 16.*dp*, 16.*dp*, 0.*dp*)  
 .*height*(50.*dp*)  
 ) **{** Text("Book")  
 **}  
 }** }

1. Для пагинации списка вызываются соответствующие функции

@Composable  
fun HotelView(navHostController: NavHostController, hotelViewModel: HotelViewModel = viewModel(factory = AppViewModelProvider.Factory)) {  
 Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*padding*(bottom = 60.*dp*)  
 ) **{** val hotelLazyPagingItems = hotelViewModel.hotelList.collectAsLazyPagingItems()  
  
 LazyVerticalGrid(  
 columns = GridCells.Fixed(2)  
 ) **{** items(  
 count = hotelLazyPagingItems.itemCount,  
 key = hotelLazyPagingItems.*itemKey* **{** hotel **->** hotel.hotelId!! **}** ) **{** index: Int **->** val hotel: Hotel? = hotelLazyPagingItems[index]  
 if (hotel != null) {  
 CardHotel(hotel, navHostController)  
 }  
 **}  
 }  
 }**}