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

Емельянов А. С.

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
2. interface BasketRepository {  
    suspend fun insertBasketSneaker(basketSneaker: BasketSneakers)  
    fun getBasketWithSneakers(id: Int): Flow<BasketWithSneakers>  
    fun getAllBasket(): Flow<List<Basket>>  
    suspend fun delete(basket: Basket)  
    suspend fun createBasket(basket: Basket):Long  
    suspend fun removeSneakerFromBasket(basketId: Int, sneakerId: Int)  
    suspend fun updateSneakerQuantity(basketId: Int, sneakerId: Int, quantity: Int)  
    suspend fun incrementSneakerQuantity(basketId: Int, sneakerId: Int)  
    suspend fun decrementSneakerQuantity(basketId: Int, sneakerId: Int)  
    suspend fun getQuantity(basketId: Int, sneakerId: Int): Int?  
    suspend fun getSneaker(basketId: Int, sneakerId: Int): BasketSneakers?  
    suspend fun getTotalPriceForUser(userId: Int): Double?  
   }

interface OrderRepository {  
 suspend fun createOrder(order: Order): Long  
 suspend fun insertOrderSneaker(orderSneaker: OrderSneaker)  
 suspend fun delete(order: Order)  
 fun getOrderWithSneakers(id: Int): Flow<OrderWithSneakers>  
 fun getAllOrder(): Flow<List<Order>>  
 fun getUserOrders(id: Int) : Flow<UserWithOrder>  
}

interface SneakerRepository {  
 suspend fun insertSneaker(sneaker: Sneaker)  
 suspend fun updateSneaker(sneaker: Sneaker)  
 suspend fun deleteSneaker(sneaker: Sneaker)  
 suspend fun getSneakerById(id: Int): Sneaker  
 fun getAllSneakersPaged(): PagingSource<Int, Sneaker>  
 fun call(): Flow<PagingData<Sneaker>>  
}

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 BasketRepoImpl(private val basketDao: BasketDao): BasketRepository {  
 override suspend fun createBasket(basket: Basket): Long = basketDao.createBasket(basket)  
 override suspend fun removeSneakerFromBasket(basketId: Int, sneakerId: Int) = basketDao.removeSneakerFromBasket(basketId, sneakerId)  
 override suspend fun updateSneakerQuantity(basketId: Int, sneakerId: Int, quantity: Int) = basketDao.updateSneakerQuantity(basketId, sneakerId, quantity)  
 override suspend fun incrementSneakerQuantity(basketId: Int, sneakerId: Int) = basketDao.incrementSneakerQuantity(basketId, sneakerId)  
 override suspend fun decrementSneakerQuantity(basketId: Int, sneakerId: Int) = basketDao.decrementSneakerQuantity(basketId, sneakerId)  
 override suspend fun insertBasketSneaker(basketSneaker: BasketSneakers) = basketDao.insertBasketSneaker(basketSneaker)  
 override fun getBasketWithSneakers(id: Int): Flow<BasketWithSneakers> = basketDao.getBasketWithSneakers(id)  
 override fun getAllBasket(): Flow<List<Basket>> = basketDao.getAllBasket()  
 override suspend fun delete(basket: Basket) = basketDao.delete(basket)  
 override suspend fun getQuantity(basketId: Int, sneakerId: Int): Int? = basketDao.getQuantity(basketId, sneakerId)  
 override suspend fun getSneaker(basketId: Int, sneakerId: Int): BasketSneakers? = basketDao.getSneaker(basketId, sneakerId)  
 override suspend fun getTotalPriceForUser(userId: Int): Double? = basketDao.getTotalPriceForUser(userId)  
}

class OrderRepoImpl(private val orderDao: OrderDao) : OrderRepository {  
  
 override suspend fun createOrder(order: Order): Long = orderDao.createOrder(order)  
  
 override suspend fun insertOrderSneaker(orderSneaker: OrderSneaker) = orderDao.insertOrderSneaker(orderSneaker)  
  
 override suspend fun delete(order: Order) = orderDao.delete(order)  
  
 override fun getOrderWithSneakers(id: Int): Flow<OrderWithSneakers> = orderDao.getOrderWithSneakers(id)  
  
 override fun getAllOrder(): Flow<List<Order>> = orderDao.getAllOrder()  
  
 override fun getUserOrders(id: Int): Flow<UserWithOrder> = orderDao.getUserOrders(id)  
}

class SneakerRepoImpl(private val sneakerDao: SneakerDao) : SneakerRepository {  
  
 override suspend fun insertSneaker(sneaker: Sneaker) = sneakerDao.insert(sneaker)  
  
 override suspend fun updateSneaker(sneaker: Sneaker) = sneakerDao.update(sneaker)  
  
 override suspend fun deleteSneaker(sneaker: Sneaker) = sneakerDao.delete(sneaker)  
  
 override suspend fun getSneakerById(id: Int): Sneaker = sneakerDao.getSneakerById(id)  
 override fun getAllSneakersPaged(): PagingSource<Int, Sneaker> = sneakerDao.getAllSneakersPaged()  
 override fun call(): Flow<PagingData<Sneaker>> {  
 return Pager(  
 PagingConfig(pageSize = 5)  
 ) **{** sneakerDao.getAllSneakersPaged()  
 **}**.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 sneakerRepo: SneakerRepository  
    val userRepo: UserRepository  
    val orderRepo: OrderRepository  
    val basketRepo: BasketRepository  
   }

class AppDataContainer(private val context: Context) : AppContainer {  
 override val sneakerRepo: SneakerRepository by *lazy* **{** SneakerRepoImpl(AppDatabase.getInstance(context).sneakerDao())  
 **}** override val userRepo: UserRepository by *lazy* **{** UserRepoImpl(AppDatabase.getInstance(context).userDao())  
 **}** override val orderRepo: OrderRepository by *lazy* **{** OrderRepoImpl(AppDatabase.getInstance(context).orderDao())  
 **}** override val basketRepo: BasketRepository by *lazy* **{** BasketRepoImpl(AppDatabase.getInstance(context).basketDao())  
 **}**}

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* **{** SneakerViewModel(*app*().container.sneakerRepo)  
    **}** *initializer* **{** UserViewModel(*app*().container.userRepo)  
    **}** *initializer* **{** OrderViewModel(*app*().container.orderRepo, *app*().container.basketRepo)  
    **}** *initializer* **{** BasketViewModel(*app*().container.basketRepo)  
    **}  
    }**}  
     
   fun CreationExtras.app(): App =  
    (this[ViewModelProvider.AndroidViewModelFactory.APPLICATION\_KEY] as App)

class BasketViewModel(private val basketRepository: BasketRepository): ViewModel() {  
  
 private val \_quantityStateMap = *mutableMapOf*<Int, MutableStateFlow<Int>>()  
  
 fun getQuantityState(basketId: Int, sneakerId: Int): StateFlow<Int> {  
 val quantityStateFlow = \_quantityStateMap.*getOrPut*(sneakerId) **{** *MutableStateFlow*(0)  
 **}** *viewModelScope*.*launch* **{** val quantityFromDb = basketRepository.getQuantity(basketId, sneakerId)  
 quantityFromDb?.*let* **{** quantityStateFlow.value = **it }  
 }** return quantityStateFlow  
 }  
  
 suspend fun isSneakerInBasket(basketId: Int, sneakerId: Int): Boolean {  
 return basketRepository.getSneaker(basketId, sneakerId) != null  
 }  
 fun addToBasket(basketSneakers: BasketSneakers) = *viewModelScope*.*launch* **{** val isSneakerInBasket = isSneakerInBasket(basketSneakers.basketId, basketSneakers.sneakerId)  
  
 if (isSneakerInBasket) {  
 incrementQuantity(basketSneakers.basketId, basketSneakers.sneakerId)  
 } else {  
 basketRepository.insertBasketSneaker(basketSneakers)  
 }  
 **}** fun getBasketSneakers(id: Int): Flow<BasketWithSneakers> {  
 return basketRepository.getBasketWithSneakers(id)  
 }  
  
 fun deleteSneakerFromBasket(basketId: Int, sneakerId: Int) = *viewModelScope*.*launch* **{** basketRepository.removeSneakerFromBasket(basketId, sneakerId)  
 **}** fun incrementQuantity(basketId: Int, sneakerId: Int) {  
 val currentQuantity = \_quantityStateMap[sneakerId]?.value ?: 1  
 \_quantityStateMap[sneakerId]?.value = currentQuantity + 1  
  
 *viewModelScope*.*launch* **{** basketRepository.incrementSneakerQuantity(basketId, sneakerId)  
 **}** }  
  
 fun decrementQuantity(basketId: Int, sneakerId: Int) {  
 val currentQuantity = \_quantityStateMap[sneakerId]?.value ?: 1  
 if (currentQuantity > 1) {  
 \_quantityStateMap[sneakerId]?.value = currentQuantity - 1  
  
 *viewModelScope*.*launch* **{** basketRepository.decrementSneakerQuantity(basketId, sneakerId)  
 **}** }  
 }  
}

class OrderViewModel(private val orderRepository: OrderRepository, private val basketRepository: BasketRepository) : ViewModel() {  
  
 var city = *mutableStateOf*("")  
 val street = *mutableStateOf*("")  
 val house = *mutableStateOf*("")  
 private val \_selectedItems = MutableLiveData<List<Sneaker>>()  
 private val \_subTotal = *mutableStateOf*(0.0)  
 val subTotal: State<Double> get() = \_subTotal  
 val selectedItems: LiveData<List<Sneaker>> get() = \_selectedItems  
  
 fun updateSelectedItems(items: List<Sneaker>) {  
 \_selectedItems.*value* = items  
 }  
  
 fun deleteOrder(order: Order) = *viewModelScope*.*launch* **{** orderRepository.delete(order)  
 **}** fun getOrderList(id: Int) : Flow<UserWithOrder> {  
 return orderRepository.getUserOrders(id)  
 }  
  
 fun getOrderWithSneakers(id: Int) : Flow<OrderWithSneakers> {  
 return orderRepository.getOrderWithSneakers(id)  
 }  
  
 fun createOrder() = *viewModelScope*.*launch* **{** val userId = GlobalUser.getInstance().getUser()?.userId!!  
 val order = Order(  
 date = Date().*time*,  
 city = city.value,  
 street = street.value,  
 house = house.value,  
 subtotal = getSubTotal(userId),  
 taxes = "%.2f".*format*(getSubTotal(userId) \* 0.05).*toDouble*(),  
 total = "%.2f".*format*(getSubTotal(userId) \* 0.05 + getSubTotal(userId)).*toDouble*(),  
 creatorUserId = GlobalUser.getInstance().getUser()?.userId!!  
 )  
  
 val orderId = orderRepository.createOrder(order)  
  
 for (sneaker in selectedItems.*value*.*orEmpty*()) {  
 val userId = GlobalUser.getInstance().getUser()?.userId!!  
 val orderSneaker = basketRepository.getQuantity(userId, sneaker.sneakerId!!)  
 ?.*let* **{** OrderSneaker( orderId.toInt(), sneaker.sneakerId!!, **it**) **}** if (orderSneaker != null) {  
 orderRepository.insertOrderSneaker(orderSneaker)  
 }  
 }  
 city.value = ""  
 street.value = ""  
 house.value = ""  
 **}** fun updateSubTotal(userId: Int) {  
 *viewModelScope*.*launch* **{** \_subTotal.value = getSubTotal(userId)  
 **}** }  
  
 suspend fun getSubTotal(userId: Int): Double {  
 return basketRepository.getTotalPriceForUser(userId) ?: 0.0  
 }  
}

class SneakerViewModel(private val sneakerRepository: SneakerRepository): ViewModel() {  
 var brand = *mutableStateOf*("")  
 val model = *mutableStateOf*("")  
 val description = *mutableStateOf*("")  
 val price = *mutableStateOf*("")  
 val photo = *mutableIntStateOf*(R.drawable.*img*)  
 val sneakerList = sneakerRepository.call().*cachedIn*(*viewModelScope*)  
 var sneaker: Sneaker? = null  
  
 fun insertSneaker() = *viewModelScope*.*launch* **{** val sneaker = Sneaker(  
 brand = brand.value,  
 model = model.value,  
 description = description.value,  
 price = price.value.*toDouble*(),  
 photo = photo.value  
 )  
 sneakerRepository.insertSneaker(sneaker)  
 **}** fun deleteSneaker(sneaker : Sneaker) = *viewModelScope*.*launch* **{** sneakerRepository.deleteSneaker(sneaker)  
 **}** fun getSneakerById(id: Int) = *viewModelScope*.*launch* **{** sneakerRepository.getSneakerById(id)  
 **}** fun UpdateSneaker(sneaker: Sneaker) = *viewModelScope*.*launch* **{** sneakerRepository.updateSneaker(sneaker)  
 **}**}

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
2. @Composable  
   fun OrderScreen(navHostController: NavHostController, basketViewModel: BasketViewModel = viewModel(factory = AppViewModelProvider.Factory), orderViewModel: OrderViewModel = viewModel(factory = AppViewModelProvider.Factory)) {  
    Column(  
    modifier = Modifier  
    .*fillMaxSize*()  
    .*background*(Color.White)  
    .*padding*(bottom = 60.*dp*)  
    .*verticalScroll*(rememberScrollState())  
    )**{** DeliveryAddress(orderViewModel)  
    val userId = GlobalUser.getInstance().getUser()?.userId  
    if (userId != null) {  
    val list by basketViewModel.getBasketSneakers(userId!!).collectAsState(initial = null)  
    val SneakerList: List<Sneaker>? = list?.sneakers  
    if (SneakerList != null) {  
    orderViewModel.updateSelectedItems(SneakerList)  
    ShoppingList(SneakerList)  
    SubTotal(orderViewModel)  
    }  
    }  
    Button(  
    colors = ButtonDefaults.buttonColors(  
    backgroundColor = colorResource(id = R.color.*figma\_blue*),  
    contentColor = Color.White  
    ),  
    onClick = **{** if(GlobalUser.getInstance().getUser() != null){  
    orderViewModel.createOrder()  
    navHostController.navigate("home")  
    }else{  
    navHostController.navigate("login")  
    }  
    **}**,  
    modifier = Modifier  
    .*fillMaxWidth*()  
    .*padding*(16.*dp*, 0.*dp*, 16.*dp*, 16.*dp*)  
    ) **{** Text("Confirm order")  
    **}  
    }**}
3. Для пагинации списка вызываются соответствующие функции

@Composable  
fun RecyclerView(navHostController: NavHostController, sneakerViewModel: SneakerViewModel = viewModel(factory = AppViewModelProvider.Factory)) {  
 Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*padding*(bottom = 60.*dp*)  
 ) **{** val sneakerLazyPagingItems = sneakerViewModel.sneakerList.collectAsLazyPagingItems()  
  
 LazyVerticalGrid(  
 columns = GridCells.Fixed(2)  
 ) **{** items(  
 count = sneakerLazyPagingItems.itemCount,  
 key = sneakerLazyPagingItems.*itemKey* **{** sneaker **->** sneaker.sneakerId!! **}** ) **{** index: Int **->** val sneaker: Sneaker? = sneakerLazyPagingItems[index]  
 if (sneaker != null) {  
 CardSneaker(sneaker, navHostController)  
 }  
 **}  
 }  
 }**}