МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа № 5**

*по дисциплине «Программирование мобильных устройств»*

Выполнил студен группы ПИбд-32

Преснякова В. В.

Проверил доцент кафедры

«Информационные системы»

Филиппов А.А.

Ульяновск, 2023

**Задание**

Взаимодействие с REST API. Необходимо:

1. Создать сервис для работы с REST API с помощью Retrofit. В качестве сервера можно использовать json-server или самостоятельную реализацию.

2. Создать репозитории для абстрагирования от реализации механизма доступа к данным. Новые репозитории должны реализовывать интерфейсы-репозитории из ЛР 4, а также использовать классы-репозитории для работы с БД.

3. Для работы со страничными данными (paging) необходимо реализовать RemoteMediator в соответствующем новом репозитории.

4. Адаптировать ViewModel для работы с новыми репозиториями. Бизнеслогика должна быть изменена минимально.

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

1. Сервер написан с использованием фреймоврка Java Spring. Создадим сервис с помощью Retrofit для их взаимодействия.
2. interface BackendService {  
     
    //HOTEL  
    @GET("hotel/get/{id}")  
    suspend fun getHotel(  
    @Path("id") id: Int,  
    ): HotelRemote  
     
    @GET("hotel/getAll")  
    suspend fun getHotels(  
    @Query("page") page: Int,  
    @Query("size") size: Int,  
    ): List<HotelRemote>  
     
    @POST("hotel/create")  
    suspend fun createHotel(  
    @Body hotel: HotelRemote,  
    ): HotelRemote  
     
    @PUT("hotel/update/{id}")  
    suspend fun updateHotel(  
    @Path("id") id: Int,  
    @Body hotel: HotelRemote  
    ): HotelRemote  
     
    @DELETE("hotel/delete/{id}")  
    suspend fun deleteHotel(  
    @Path("id") id: Int  
    )  
     
    //USER  
    @POST("user/signup")  
    suspend fun SignUp(  
    @Body user: UserRemote,  
    ): UserRemote  
     
    @POST("user/signin")  
    suspend fun SignIn(  
    @Body user: UserRemoteSignIn  
    ): UserRemote  
     
    //ORDER  
    @POST("order/createOrderHotel")  
    suspend fun createOrderHotel(  
    @Body orderHotel: OrderHotelRemote  
    )  
     
    @POST("order/create")  
    suspend fun createOrder(  
    @Body order: OrderRemote  
    ): Long  
     
    @GET("order/getUserOrders/{userId}")  
    suspend fun getUserOrders(  
    @Path("userId") userId: Int  
    ) : List<OrderRemote>  
     
    @GET("order/getHotelFromOrder/{orderId}")  
    suspend fun getHotelFromOrder(  
    @Path("orderId") orderId: Int  
    ) : List<HotelRemote>  
     
    @GET("order/deleteOrder/{orderId}")  
    suspend fun deleteOrder(  
    @Path("orderId") orderId: Int  
    )  
     
    companion object {  
    private const val BASE\_URL = "https://59k4pfj3-8080.euw.devtunnels.ms/api/"  
     
    @Volatile  
    private var INSTANCE: BackendService? = null  
     
    fun getInstance(): BackendService {  
    return INSTANCE ?: *synchronized*(this) **{** val logger = HttpLoggingInterceptor()  
    logger.level = HttpLoggingInterceptor.Level.*BASIC* val client = OkHttpClient.Builder()  
    .addInterceptor(logger)  
    .build()  
    return Retrofit.Builder()  
    .baseUrl(BASE\_URL)  
    .client(client)  
    .addConverterFactory(Json.*asConverterFactory*("application/json".*toMediaType*()))  
    .build()  
    .create(BackendService::class.*java*)  
    .*also* **{** INSTANCE = **it }  
    }** }  
    }  
   }
3. Реализация интерфейсов для доступа к данным с сервера

class RestOrderRepository(private val service: BackendService): OrderRepository {  
 override suspend fun createOrder(order: Order): Long {  
 return service.createOrder(order.*toOrderRemote*())  
 }  
  
 override suspend fun insertOrderHotel(orderHotel: OrderHotel) {  
 service.createOrderHotel(orderHotel.*toOrderHotelRemote*())  
 }  
  
 override suspend fun delete(orderId: Int) {  
 service.deleteOrder(orderId)  
 }  
  
 override suspend fun getHotelFromOrder(id: Int): Flow<List<Hotel>> {  
 val hotelsRemoteList = service.getHotelFromOrder(id)  
 val hotelsList = hotelsRemoteList.*map* **{ it**.*toHotel*() **}** return *flowOf*(hotelsList.*toList*())  
 }  
  
 override suspend fun getUserOrders(id: Int): Flow<List<Order>> {  
 val ordersRemoteList = service.getUserOrders(id)  
 val ordersList = ordersRemoteList.*map* **{ it**.*toOrder*() **}** return *flowOf*(ordersList.*toList*())  
 }  
}

class RestHotelRepository(  
 private val service: BackendService,  
 private val dbHotelRepository: HotelRepoImpl,  
 private val database: AppDatabase,  
 private val dbRemoteKeyRepository: RemoteKeysRepositoryImpl  
  
) : HotelRepository {  
  
 override fun getAllHotels(): Flow<PagingData<Hotel>> {  
 val pagingSourceFactory = **{** dbHotelRepository.getAllHotelsPagingSource() **}** @OptIn(ExperimentalPagingApi::class)  
 return Pager(  
 config = PagingConfig(  
 pageSize = AppContainer.LIMIT,  
 enablePlaceholders = false  
 ),  
 remoteMediator = HotelRemoteMediator(  
 service,  
 dbHotelRepository,  
 database,  
 dbRemoteKeyRepository,  
 ),  
 pagingSourceFactory = pagingSourceFactory  
 ).flow  
 }  
  
 override suspend fun getHotelById(id: Int): Hotel = service.getHotel(id).*toHotel*()  
 override suspend fun insertHotel(hotel: Hotel) {  
 service.createHotel(hotel.*toHotelRemote*())  
 }  
  
 override suspend fun updateHotel(hotel: Hotel) {  
 hotel.hotelId?.*let* **{** service.updateHotel(**it**, hotel.*toHotelRemote*()) **}** }  
  
 override suspend fun deleteHotel(hotel: Hotel) {  
 hotel.hotelId?.*let* **{** service.deleteHotel(**it**) **}** }  
}

class RestUserRepository(  
 private var service: BackendService  
): UserRepository {  
 override suspend fun createUser(user: User) {  
 service.SignUp(user.*toUserRemote*())  
 }  
  
 override suspend fun updateUser(user: User) {  
 *println*()  
 }  
  
 override suspend fun deleteUser(user: User) {  
 *println*()  
 }  
 override suspend fun authUser(user: UserRemoteSignIn): User {  
 return service.SignIn(user).*toUser*()  
 }  
}

Также создадим модели, для data transfer object

@Serializable  
data class OrderRemote(  
 val id: Int? = 0,  
 val dateFrom: String = "",  
 val dateTo: String = "",  
 val rooms: Int = 0,  
 val total: Double = 0.0,  
 val userId: Int = 0,  
 val hotelId: Int = 0  
 )  
  
fun OrderRemote.toOrder(): Order = Order(  
 id,  
 dateFrom,  
 dateTo,  
 rooms,  
 total,  
 userId,  
 hotelId,  
 )  
  
fun Order.toOrderRemote():OrderRemote = OrderRemote(  
 orderId,  
 dateFrom,  
 dateTo,  
 rooms,  
 total,  
 creatorUserId,  
 bookedHotelId,  
)

@Serializable  
data class HotelRemote (  
 val id: Int? = 0,  
 val name: String = "",  
 val price: Double = 0.0,  
 val img: Int = 0,  
 val stars: Int = 0,  
 val location: String = "",  
 val info: String = "",  
  
 )  
  
fun HotelRemote.toHotel(): Hotel = Hotel(  
 id,  
 name,  
 price,  
 img,  
 stars,  
 location,  
 info  
)  
  
fun Hotel.toHotelRemote():HotelRemote = HotelRemote(  
 hotelId,  
 name,  
 price,  
 img,  
 stars,  
 location,  
 info  
)

@Serializable  
data class UserRemote (  
 val id: Int? = 0,  
 val name: String = "",  
 val surname: String = "",  
 val email: String = "",  
 val password: String = "",  
 val role: String = "",  
 val photo: Int? = 0,  
)  
  
  
fun UserRemote.toUser(): User = User(  
 id,  
 name,  
 surname,  
 email,  
 password,  
 role,  
 photo  
)  
  
fun User.toUserRemote():UserRemote = UserRemote(  
 userId,  
 name,  
 surname,  
 email,  
 password,  
 role,  
 photo  
)

@Serializable  
data class UserRemoteSignIn(  
 val email: String = "",  
 val password: String = "",  
)

1. Реализуем RemoteMediator для работы со списком каталога отелей
2. @OptIn(ExperimentalPagingApi::class)  
   class HotelRemoteMediator(  
    private val service: BackendService,  
    private val hotelRepository: HotelRepoImpl,  
    private val database: AppDatabase,  
    private val dbRemoteKeyRepository: RemoteKeysRepositoryImpl  
   ) : RemoteMediator<Int, Hotel>() {  
    override suspend fun initialize(): InitializeAction {  
    return InitializeAction.*LAUNCH\_INITIAL\_REFRESH* }  
    override suspend fun load(  
    loadType: LoadType,  
    state: PagingState<Int, Hotel>  
    ): MediatorResult {  
    val page = when (loadType) {  
    LoadType.*REFRESH* -> {  
    val remoteKeys = getRemoteKeyClosestToCurrentPosition(state)  
    remoteKeys?.nextKey?.minus(1) ?: 1  
    }  
     
    LoadType.*PREPEND* -> {  
    val remoteKeys = getRemoteKeyForFirstItem(state)  
    remoteKeys?.prevKey  
    ?: return MediatorResult.Success(endOfPaginationReached = remoteKeys != null)  
    }  
     
    LoadType.*APPEND* -> {  
    val remoteKeys = getRemoteKeyForLastItem(state)  
    remoteKeys?.nextKey  
    ?: return MediatorResult.Success(endOfPaginationReached = remoteKeys != null)  
    }  
    }  
     
    try {  
    val hotels = service.getHotels(page, state.config.pageSize).*map* **{ it**.*toHotel*() **}** val endOfPaginationReached = hotels.isEmpty()  
    database.withTransaction **{** if (loadType == LoadType.*REFRESH*) {  
    dbRemoteKeyRepository.deleteRemoteKey(RemoteKeyType.*HOTEL*)  
    hotelRepository.clearHotels()  
    }  
    val prevKey = if (page == 1) null else page - 1  
    val nextKey = if (endOfPaginationReached) null else page + 1  
    val keys = hotels.*map* **{** RemoteKeys(  
    entityId = **it**.hotelId!!,  
    type = RemoteKeyType.*HOTEL*,  
    prevKey = prevKey,  
    nextKey = nextKey  
    )  
    **}** dbRemoteKeyRepository.createRemoteKeys(keys)  
    hotelRepository.insertHotels(hotels)  
    **}** return MediatorResult.Success(endOfPaginationReached = endOfPaginationReached)  
    } catch (exception: IOException) {  
    return MediatorResult.Error(exception)  
    } catch (exception: HttpException) {  
    return MediatorResult.Error(exception)  
    }  
    }  
     
    private suspend fun getRemoteKeyForLastItem(state: PagingState<Int, Hotel>): RemoteKeys? {  
    return state.pages.*lastOrNull* **{ it**.data.*isNotEmpty*() **}**?.data?.*lastOrNull*()  
    ?.*let* **{** hotel **->** hotel.hotelId?.*let* **{** dbRemoteKeyRepository.getAllRemoteKeys(**it**, RemoteKeyType.*HOTEL*) **}  
    }** }  
     
    private suspend fun getRemoteKeyForFirstItem(state: PagingState<Int, Hotel>): RemoteKeys? {  
    return state.pages.*firstOrNull* **{ it**.data.*isNotEmpty*() **}**?.data?.*firstOrNull*()  
    ?.*let* **{** hotel **->** hotel.hotelId?.*let* **{** dbRemoteKeyRepository.getAllRemoteKeys(**it**, RemoteKeyType.*HOTEL*) **}  
    }** }  
     
    private suspend fun getRemoteKeyClosestToCurrentPosition(  
    state: PagingState<Int, Hotel>  
    ): RemoteKeys? {  
    return state.anchorPosition?.*let* **{** position **->** state.closestItemToPosition(position)?.hotelId?.*let* **{** hotelUid **->** dbRemoteKeyRepository.getAllRemoteKeys(hotelUid, RemoteKeyType.*HOTEL*)  
    **}  
    }** }  
   }

Также поменяем в di реализацию вместо бд, на апи

class AppDataContainer(private val context: Context) : AppContainer {  
 override val hotelRepo: HotelRepository by *lazy* **{** RestHotelRepository(  
 BackendService.getInstance(),  
 hotelRepository,  
 AppDatabase.getInstance(context),  
 remoteKeyRepository  
 )  
 **}** override val userRepo: UserRepository by *lazy* **{** RestUserRepository(BackendService.getInstance())  
 **}** override val orderRepo: OrderRepository by *lazy* **{** RestOrderRepository(BackendService.getInstance())  
 **}** private val hotelRepository: HotelRepoImpl by *lazy* **{** HotelRepoImpl(AppDatabase.getInstance(context).hotelDao())  
 **}** private val remoteKeyRepository: RemoteKeysRepositoryImpl by *lazy* **{** RemoteKeysRepositoryImpl(AppDatabase.getInstance(context).remoteKeysDao())  
 **}**}

1. Адаптируем view модели под новую логику взаимодействия с данными

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 = "USER",  
 photo = R.drawable.*shailushai* )  
 userRepository.createUser(user)  
 **}** fun authUser() = *viewModelScope*.*launch* **{** val user = userRepository.authUser(UserRemoteSignIn(email.value, password.value))  
 GlobalUser.getInstance().setUser(user)  
 **}** fun isValidEmail(email: String): Boolean {  
 return android.util.Patterns.*EMAIL\_ADDRESS*.matcher(email).matches()  
 }  
}