МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное бюджетное образовательное учреждение высшего образования  
**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Факультет информационных систем и технологий  
Кафедра «Информационные системы»  
Дисциплина «Программирование мобильных устройств»

Лабораторная работа №5

Выполнил:  
студент гр. ПИбд-32  
Бондаренко М.С.  
Проверил:  
доцент кафедры  
Филиппов А.А.

Ульяновск, 2023 г

**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.

**Выполнение**

1. Создал сервис для сетевых взаимодействий:

interface ServerService {  
 //USER  
 @GET("user/get/{id}")  
 suspend fun getUser(  
 @Path("id") id: Int,  
 ): UserRemote  
  
 @POST("user/signup")  
 suspend fun SignUp(  
 @Body user: UserRemote,  
 ): UserRemote  
  
 @POST("user/signin")  
 suspend fun SignIn(  
 @Body user: UserRemoteSignIn  
 ): UserRemote  
  
 @POST("user/update")  
 suspend fun updateUser(  
 @Body user: UserRemote  
 ): UserRemote  
  
 @GET("user/getAll")  
 suspend fun getUsers(): List<UserRemote>  
  
 //STORY  
 @GET("story/get/{id}")  
 suspend fun getStory(  
 @Path("id") id: Int,  
 ): StoryRemote  
  
 @GET("story/getAll")  
 suspend fun getStories(  
 @Query("page") page: Int,  
 @Query("size") size: Int,  
 ): List<StoryRemote>  
  
 @POST("story/create")  
 suspend fun createStory(  
 @Body service: StoryRemote,  
 ): StoryRemote  
  
 @PUT("story/update/{id}")  
 suspend fun updateStory(  
 @Path("id") id: Int,  
 @Body service: StoryRemote  
 ): StoryRemote  
  
 @DELETE("story/delete/{id}")  
 suspend fun deleteStory(  
 @Path("id") id: Int  
 )  
  
 @GET("story/getByUser")  
 suspend fun getUserStories(  
 @Query("page") page: Int,  
 @Query("size") size: Int,  
 @Query("userId") userId: Int,  
 ): List<StoryRemote>  
  
 //MAIL  
 @GET("mail/get/{id}")  
 suspend fun getMail(  
 @Path("id") id: Int,  
 ): MailRemote  
  
 @GET("mail/getAll")  
 suspend fun getMails(  
 @Query("page") page: Int,  
 @Query("size") size: Int,  
 ): List<MailRemote>  
  
 @POST("mail/create")  
 suspend fun createMail(  
 @Body service: MailRemote,  
 ): MailRemote  
  
 @PUT("mail/update/{id}")  
 suspend fun updateMail(  
 @Path("id") id: Int,  
 @Body service: MailRemote  
 ): MailRemote  
  
 @DELETE("mail/delete/{id}")  
 suspend fun deleteMail(  
 @Path("id") id: Int  
 )  
  
 //INSTANCE  
 companion object {  
 private const val BASE\_URL = "https://7hz21fz1-8080.euw.devtunnels.ms/api/"  
  
 @Volatile  
 private var INSTANCE: ServerService? = null  
  
 fun getInstance(): ServerService {  
 return INSTANCE ?: *synchronized*(this) **{** val client = OkHttpClient.Builder()  
 .build()  
 return Retrofit.Builder()  
 .baseUrl(BASE\_URL)  
 .client(client)  
 .addConverterFactory(Json.*asConverterFactory*("application/json".*toMediaType*()))  
 .build()  
 .create(ServerService::class.*java*)  
 .*also* **{** INSTANCE = **it }  
 }** }  
 }  
}

2. Создал репозитории для абстрагирования от реализации:

class RestUserRepository(private var service: ServerService): UserRepository {  
 override fun getAllUsers(): Flow<List<User>> {  
 *TODO*("Not yet implemented")  
 }  
  
 override suspend fun getUser(id: Int): User = service.getUser(id).*toUser*()  
  
  
 override suspend fun getUserByLogin(user: UserRemoteSignIn): User {  
 return service.SignIn(user).*toUser*()  
 }  
  
 override suspend fun insertUser(user: User) {  
 service.SignUp(user.*toUserRemote*())  
 }  
  
 override suspend fun updateUser(user: User) {  
 service.updateUser(user.*toUserRemote*())  
 }  
  
 override suspend fun deleteUser(user: User) {  
 *TODO*("Not yet implemented")  
 }  
}

class RestStoryRepository(private var service: ServerService,  
 private val dbStoryRepository: OfflineStoryRepository,  
 private val dbUserRepository: OfflineUserRepository,  
 private val database: MobileAppDataBase,  
 private val dbRemoteKeyRepository: RemoteKeysRepositoryImpl  
): StoryRepository {  
  
 override fun getAllStories(): Flow<PagingData<Story>> {  
 val pagingSourceFactory = **{** dbStoryRepository.getAllStoriesPagingSource()  
 **}** @OptIn(ExperimentalPagingApi::class)  
 return Pager(  
 config = PagingConfig(  
 pageSize = MobileAppContainer.LIMIT,  
 enablePlaceholders = false  
 ),  
 remoteMediator = ServiceRemoteMediator(  
 service,  
 dbStoryRepository,  
 dbUserRepository,  
 database,  
 dbRemoteKeyRepository,  
 ),  
 pagingSourceFactory = pagingSourceFactory  
 ).flow  
 /\*return Pager(  
 config = PagingConfig(  
 pageSize = MobileAppContainer.LIMIT,  
 enablePlaceholders = false  
 ),  
 pagingSourceFactory = { StoryPagingSource(service) }  
 ).flow\*/  
 }  
  
 override fun getStoriesByUserId(userId: Int): Flow<PagingData<Story>> {  
 return Pager(  
 config = PagingConfig(  
 pageSize = MobileAppContainer.LIMIT,  
 enablePlaceholders = false  
 ),  
 pagingSourceFactory = **{** StoryPagingSource(service, userId) **}** ).flow  
 }  
  
 override suspend fun getStoryById(id: Int): Story? = service.getStory(id).*toStory*()  
  
 override suspend fun insertStory(story: Story) {  
 service.createStory(story.*toStoryRemote*())  
 }  
  
 override suspend fun updateStory(story: Story) {  
 story.id?.*let* **{** service.updateStory(**it**, story.*toStoryRemote*())  
 **}** }  
  
 override suspend fun deleteStory(story: Story) {  
 try {  
 story.id?.*let* **{** this.service.deleteStory(**it**) **}** dbStoryRepository.deleteStory(story)  
 }catch (ex: Exception){}  
 }  
}  
  
class StoryPagingSource(private val service: ServerService,  
 private val userId: Int? = null) : PagingSource<Int, Story>() {  
 override suspend fun load(params: LoadParams<Int>): LoadResult<Int, Story> {  
 try {  
 val nextPageNumber = params.key ?: 1  
 val pageSize = params.loadSize  
  
 val response = if (userId != null) {  
 service.getUserStories(nextPageNumber, pageSize, userId)  
 } else {  
 service.getStories(nextPageNumber, pageSize)  
 }  
 val stories = response.*map* **{ it**.*toStory*() **}** return LoadResult.Page(  
 data = stories,  
 prevKey = if (nextPageNumber == 1) null else nextPageNumber - 1,  
 nextKey = if (stories.isEmpty()) null else nextPageNumber + 1  
 )  
 } catch (exception: Exception) {  
 return LoadResult.Error(exception)  
 }  
 }  
  
 override fun getRefreshKey(state: PagingState<Int, Story>): Int? {  
 *TODO*("Not yet implemented")  
 }  
}

class RestMailRepository(private var service: ServerService): MailRepository {  
 override fun getAllMails(): Flow<PagingData<Mail>> {  
 return Pager(  
 config = PagingConfig(  
 pageSize = MobileAppContainer.LIMIT,  
 enablePlaceholders = false  
 ),  
 pagingSourceFactory = **{** MailPagingSource(service) **}** ).flow  
 }  
  
 override suspend fun getMail(id: Int): Mail? = service.getMail(id).*toMail*()  
  
 override suspend fun insertMail(mail: Mail) {  
 service.createMail(mail.*toMailRemote*())  
 }  
  
 override suspend fun updateMail(mail: Mail) {  
 *TODO*("Not yet implemented")  
 }  
  
 override suspend fun deleteMail(mail: Mail) {  
 *TODO*("Not yet implemented")  
 }  
}  
  
class MailPagingSource(private val service: ServerService) : PagingSource<Int, Mail>() {  
 override suspend fun load(params: LoadParams<Int>): LoadResult<Int, Mail> {  
 try {  
 val nextPageNumber = params.key ?: 1  
 val pageSize = params.loadSize  
  
 val response = service.getMails(nextPageNumber, pageSize)  
 val mails = response.*map* **{ it**.*toMail*() **}** // Преобразование MailRemote в Mail  
  
 return LoadResult.Page(  
 data = mails,  
 prevKey = if (nextPageNumber == 1) null else nextPageNumber - 1,  
 nextKey = if (mails.isEmpty()) null else nextPageNumber + 1  
 )  
 } catch (exception: Exception) {  
 return LoadResult.Error(exception)  
 }  
 }  
  
 override fun getRefreshKey(state: PagingState<Int, Mail>): Int? {  
 *TODO*("Not yet implemented")  
 }  
}

3. Реализовал RemoteMediator:

@OptIn(ExperimentalPagingApi::class)  
class ServiceRemoteMediator(private val service: ServerService,  
 private val storyRepository: OfflineStoryRepository,  
 private val userRepository: OfflineUserRepository,  
 private val database: MobileAppDataBase,  
 private val dbRemoteKeyRepository: RemoteKeysRepositoryImpl  
) : RemoteMediator<Int, Story>() {  
 override suspend fun initialize(): InitializeAction {  
 return InitializeAction.*LAUNCH\_INITIAL\_REFRESH* }  
 override suspend fun load(  
 loadType: LoadType,  
 state: PagingState<Int, Story>  
 ): 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 users = service.getUsers().*map* **{ it**.*toUser*() **}** val stories = service.getStories(page, state.config.pageSize).*map* **{ it**.*toStory*() **}** val endOfPaginationReached = stories.isEmpty()  
 database.withTransaction **{** if (loadType == LoadType.*REFRESH*) {  
 dbRemoteKeyRepository.deleteRemoteKey(RemoteKeyType.*STORY*)  
 storyRepository.clearStories()  
 userRepository.clearUsers()  
 }  
 val prevKey = if (page == 1) null else page - 1  
 val nextKey = if (endOfPaginationReached) null else page + 1  
 val keys = stories.*map* **{** RemoteKeys(  
 entityId = **it**.id!!,  
 type = RemoteKeyType.*STORY*,  
 prevKey = prevKey,  
 nextKey = nextKey  
 )  
 **}** dbRemoteKeyRepository.createRemoteKeys(keys)  
 userRepository.insertUsers(users)  
 storyRepository.insertStories(stories)  
 **}** 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, Story>): RemoteKeys? {  
 return state.pages.*lastOrNull* **{ it**.data.*isNotEmpty*() **}**?.data?.*lastOrNull*()  
 ?.*let* **{** story **->** story.id?.*let* **{** dbRemoteKeyRepository.getAllRemoteKeys(**it**, RemoteKeyType.*STORY*) **}  
 }** }  
  
 private suspend fun getRemoteKeyForFirstItem(state: PagingState<Int, Story>): RemoteKeys? {  
 return state.pages.*firstOrNull* **{ it**.data.*isNotEmpty*() **}**?.data?.*firstOrNull*()  
 ?.*let* **{** story **->** story.id?.*let* **{** dbRemoteKeyRepository.getAllRemoteKeys(**it**, RemoteKeyType.*STORY*) **}  
 }** }  
  
 private suspend fun getRemoteKeyClosestToCurrentPosition(  
 state: PagingState<Int, Story>  
 ): RemoteKeys? {  
 return state.anchorPosition?.*let* **{** position **->** state.closestItemToPosition(position)?.id?.*let* **{** storyUid **->** dbRemoteKeyRepository.getAllRemoteKeys(storyUid, RemoteKeyType.*STORY*)  
 **}  
 }** }  
}

А также добавлены сериализуемые классы и методы конвертации из Bitmap в String и в обратную сторону:

@Serializable  
data class UserRemote(  
 val id: Int? = 0,  
 val login: String,  
 val password: String = "",  
 val email: String = "",  
 val photo: String? = null  
)  
  
fun UserRemote.toUser(): User = User(  
 id,  
 login,  
 password,  
 email,  
 photo?.*let* **{** RemoteConverters.toBitmap(**it**) **}**,  
)  
  
fun User.toUserRemote():UserRemote = UserRemote(  
 id,  
 login,  
 password,  
 email,  
 photo?.*let* **{** RemoteConverters.fromBitmap(**it**) **}**,  
)

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

@Serializable  
data class StoryRemote(  
 val id: Int? = null,  
 val title: String,  
 val description: String,  
 val cover: String,  
 val postdate: Long? = Date().*time*,  
 val userId: Int  
)  
  
fun StoryRemote.toStory(): Story = Story(  
 id,  
 title,  
 description,  
 RemoteConverters.toBitmap(cover),  
 postdate,  
 userId  
)  
  
fun Story.toStoryRemote(): StoryRemote = StoryRemote(  
 id,  
 title,  
 description,  
 RemoteConverters.fromBitmap(cover),  
 postdate,  
 userId  
)

@Serializable  
data class MailRemote(  
 val id: Int? = null,  
 val message: String,  
 val postdate: Long? = Date().*time*,  
 val userId: Int  
)  
  
fun MailRemote.toMail(): Mail = Mail(  
 id,  
 message,  
 postdate,  
 userId  
)  
  
fun Mail.toMailRemote():MailRemote = MailRemote(  
 id,  
 message,  
 postdate,  
 userId  
)

class RemoteConverters {  
 companion object {  
 private const val CHARSET\_UTF8 = "UTF-8"  
  
 fun fromBitmap(bitmap: Bitmap): String {  
 val outputStream = ByteArrayOutputStream()  
 bitmap.compress(Bitmap.CompressFormat.*PNG*, 100, outputStream)  
 val byteArray = outputStream.toByteArray()  
 return Base64.encodeToString(byteArray, Base64.*NO\_WRAP*)  
 }  
  
 fun toBitmap(base64String: String): Bitmap {  
 val byteArray = Base64.decode(base64String, Base64.*NO\_WRAP*)  
 return BitmapFactory.decodeByteArray(byteArray, 0, byteArray.size)  
 }  
 }  
}

4. Демонстрация работы приложения. Внешний вид программы не изменился с прошлой лабораторной.