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 №4

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

Ульяновск, 2023 г

**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.

**Выполнение**

1. Были созданы интерфейсы-репозитории и их реализации для пользователя, истории и почты:

interface UserRepository {  
 fun getAllUsers(): Flow<List<User>>  
  
 suspend fun getUser(id: Int): User?  
  
 suspend fun getUserByLogin(user: UserRemoteSignIn): User?  
  
 suspend fun insertUser(user: User)  
  
 suspend fun updateUser(user: User)  
  
 suspend fun deleteUser(user: User)  
}

interface StoryRepository {  
 fun getAllStories(): Flow<PagingData<Story>>  
  
 fun getStoriesByUserId(userId: Int): Flow<PagingData<Story>>  
  
 suspend fun getStoryById(id: Int): Story?  
  
 suspend fun insertStory(story: Story)  
  
 suspend fun updateStory(story: Story)  
  
 suspend fun deleteStory(story: Story)  
}

interface MailRepository {  
 fun getAllMails(): Flow<PagingData<Mail>>  
  
 suspend fun getMail(id: Int): Mail?  
  
 suspend fun insertMail(mail: Mail)  
  
 suspend fun updateMail(mail: Mail)  
  
 suspend fun deleteMail(mail: Mail)  
}

class OfflineUserRepository(private val userDao: UserDao): UserRepository {  
 override fun getAllUsers(): Flow<List<User>> = userDao.getAll()  
  
 override suspend fun getUser(id: Int): User? = userDao.getById(id)  
  
 override suspend fun getUserByLogin(user: UserRemoteSignIn): User? = userDao.getByLogin(user.login)  
  
 override suspend fun insertUser(user: User) = userDao.insert(user)  
  
 override suspend fun updateUser(user: User) = userDao.update(user)  
  
 override suspend fun deleteUser(user: User) = userDao.delete(user)  
  
 suspend fun clearUsers() = userDao.deleteAll()  
 suspend fun insertUsers(users: List<User>) =  
 userDao.insert(\*users.*toTypedArray*())  
}

class OfflineStoryRepository(private val storyDao: StoryDao): StoryRepository {  
 override fun getAllStories(): Flow<PagingData<Story>> {  
 return Pager(  
 config = PagingConfig(  
 pageSize = 5,  
 prefetchDistance = 1,  
 enablePlaceholders = true,  
 initialLoadSize = 10,  
 maxSize = 15  
 ),  
 pagingSourceFactory = **{** storyDao.getAll()  
 **}** ).flow  
 }  
  
 override fun getStoriesByUserId(userId: Int): Flow<PagingData<Story>> {  
 *TODO*("Not yet implemented")  
 }  
  
 override suspend fun getStoryById(id: Int): Story? = storyDao.getById(id)  
  
 override suspend fun insertStory(story: Story) = storyDao.insert(story)  
  
 override suspend fun updateStory(story: Story) = storyDao.update(story)  
  
 override suspend fun deleteStory(story: Story) = storyDao.delete(story)  
  
 suspend fun clearStories() = storyDao.deleteAll()  
 suspend fun insertStories(stories: List<Story>) =  
 storyDao.insert(\*stories.*toTypedArray*())  
  
 fun getAllStoriesPagingSource(): PagingSource<Int, Story> = storyDao.getAll()  
}

class OfflineMailRepository(private val mailDao: MailDao): MailRepository {  
 override fun getAllMails(): Flow<PagingData<Mail>> {  
 return Pager(  
 config = PagingConfig(  
 pageSize = 8,  
 prefetchDistance = 2,  
 enablePlaceholders = true,  
 initialLoadSize = 12,  
 maxSize = 24  
 ),  
 pagingSourceFactory = **{** mailDao.getAll()  
 **}** ).flow  
 }  
  
 override suspend fun getMail(id: Int): Mail? = mailDao.getById(id)  
  
 override suspend fun insertMail(mail: Mail) = mailDao.insert(mail)  
  
 override suspend fun updateMail(mail: Mail) = mailDao.update(mail)  
  
 override suspend fun deleteMail(mail: Mail) = mailDao.delete(mail)  
}

2. Были разработаны классы ViewModel для реализации бизнес-логики, в них была перенесена и доработана логика из представлений, слой UI теперь обращается к этим классам:

class UserViewModel(private val userRepository: UserRepository): ViewModel() {  
 //val getAllUsers = userRepository.getAllUsers()  
  
 suspend fun getUser(id: Int): User? = userRepository.getUser(id)  
  
 fun updateUser(user: User) = *viewModelScope*.*launch* **{** if (user.login.*isEmpty*()){  
 return@launch  
 }  
  
 if (user.email.*isEmpty*() || !isValidEmail(user.email)){  
 return@launch  
 }  
  
 if (user.password.*isEmpty*()){  
 return@launch  
 }  
 userRepository.updateUser(user)  
 GlobalUser.getInstance().setUser(user)  
 **}** fun deleteUser(user: User) = *viewModelScope*.*launch* **{** userRepository.deleteUser(user)  
 **}** fun regUser(user: User) = *viewModelScope*.*launch* **{** if(user.password.*isEmpty*()){  
 return@launch  
 }  
  
 if(user.email.*isEmpty*() || !isValidEmail(user.email)){  
 return@launch  
 }  
 userRepository.insertUser(user)  
 GlobalUser.getInstance().setUser(userRepository.getUserByLogin(  
 UserRemoteSignIn(user.login, user.password)))  
 **}** fun authUser(user: User) = *viewModelScope*.*launch* **{** val globalUser = userRepository.getUserByLogin(UserRemoteSignIn(user.login, user.password))  
 globalUser?.*let* **{** if (user.password.*isNotEmpty*() && user.password == globalUser.password){  
 GlobalUser.getInstance().setUser(globalUser)  
 }  
 **}  
 }** private fun isValidEmail(email: String): Boolean {  
 return android.util.Patterns.*EMAIL\_ADDRESS*.matcher(email).matches()  
 }  
}

class StoryViewModel(private val storyRepository: StoryRepository): ViewModel() {  
  
 val getAllStories: Flow<PagingData<Story>> = storyRepository.getAllStories().*cachedIn*(*viewModelScope*)  
  
 suspend fun getStoryById(id: Int): Story? = storyRepository.getStoryById(id)  
  
 val getStoriesByUserId: Flow<PagingData<Story>> = GlobalUser.getInstance().getUser()?.id?.*let* **{** storyRepository.getStoriesByUserId(**it**).*cachedIn*(*viewModelScope*)  
 **}** ?: *flowOf*(PagingData.empty())  
  
 fun insertStory(story: Story) = *viewModelScope*.*launch* **{** storyRepository.insertStory(story)  
 **}** fun updateStory(story: Story) = *viewModelScope*.*launch* **{** storyRepository.updateStory(story)  
 **}** fun deleteStory(story: Story) = *viewModelScope*.*launch* **{** storyRepository.deleteStory(story)  
 **}**}

class MailViewModel(private val mailRepository: MailRepository): ViewModel() {  
 val getAllMails: Flow<PagingData<Mail>> = mailRepository.getAllMails().*cachedIn*(*viewModelScope*)  
  
 suspend fun getMail(id: Int): Mail? = mailRepository.getMail(id)  
  
 fun insertMail(mail: Mail) = *viewModelScope*.*launch* **{** mailRepository.insertMail(mail)  
 **}** fun updateMail(mail: Mail) = *viewModelScope*.*launch* **{** mailRepository.updateMail(mail)  
 **}** fun deleteMail(mail: Mail) = *viewModelScope*.*launch* **{** mailRepository.deleteMail(mail)  
 **}**}

3. Был внедрен механизм пагинации для списков товаров, это было сделано для того, чтобы данные подгружались частями незаметно для пользователя, не тормозя приложение:

В интерфейсе-репозитории

@Query("select \* from stories order by id desc")  
fun getAll(): PagingSource<Int, Story>

@Query("select \* from mails order by id desc")  
fun getAll(): PagingSource<Int, Mail>

В реализации интерфейса

override fun getAllStories(): Flow<PagingData<Story>> {  
 return Pager(  
 config = PagingConfig(  
 pageSize = 5,  
 prefetchDistance = 1,  
 enablePlaceholders = true,  
 initialLoadSize = 10,  
 maxSize = 15  
 ),  
 pagingSourceFactory = **{** storyDao.getAll()  
 **}** ).flow  
}

В ViewModel-классе

val getAllStories: Flow<PagingData<Story>> = storyRepository.getAllStories().*cachedIn*(*viewModelScope*)

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