Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА ПО ANDROID**

**Дисциплина: Программирование для мобильных платформ**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В.Чухиль

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. И. Шиян

# Постановка задачи

Разработать мобильное приложение «Notes» в IDE Android Studio.

**Используемые технологии**

**Среда разработки: Android Studio.**

Android Studio – интегрированная среда разработки (IDE) для работы с платформой Android, анонсированная 16 мая 2013 года на конференции Google I/O. В последней версии Android Studio поддерживается Android 4.1 и выше.

**Язык программирования: Kotlin.**

Kotlin – статически типизированный, объектно-ориентированный язык программирования, работающий поверх Java Virtual Machine и разрабатываемый компанией JetBrains. Также компилируется в JavaScript и в исполняемый код ряда платформ через инфраструктуру LLVM.

Авторы ставили целью создать язык более лаконичный и типобезопасный, чем Java, и более простой, чем Scala. Следствием упрощения по сравнению со Scala стали также более быстрая компиляция и лучшая поддержка языка в IDE. Язык полностью совместим с Java, что позволяет Java-разработчикам постепенно перейти к его использованию; в частности, язык также встраивается Android, что позволяет для существующего Android-приложения внедрять новые функции на Kotlin без переписывания приложения целиком.

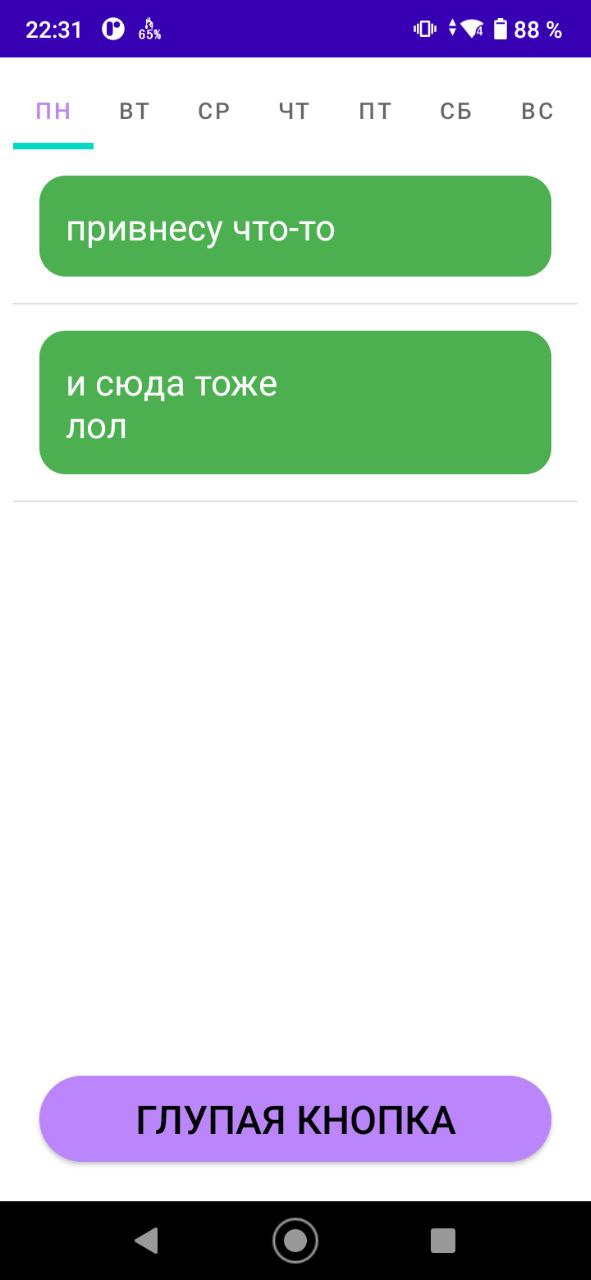
**Используемые библиотеки: Room, RxJava3, Fragment.**

Room — это библиотека, которая является частью архитектурных компонентов Android. Она облегчает работу с объектами SQLiteDatabase в приложении, уменьшая объём стандартного кода и проверяя SQL-запросы во время компиляции.

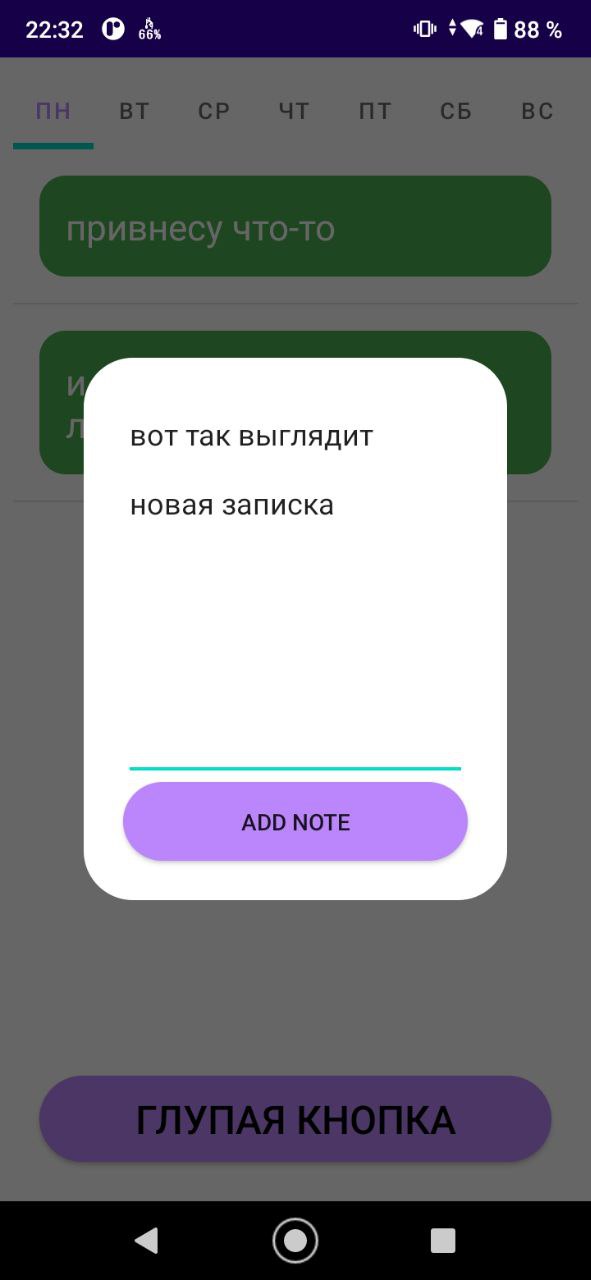
RxJava — это фреймворк от ReactiveX (RX) для реактивного программирования на Java — подхода, при котором в ответ на изменения одних сущностей автоматически меняются другие. Он используется в мобильной и веб-разработке и помогает реализовать асинхронность — поочередное выполнение действий.

Fragment представляет кусочек визуального интерфейса приложения, который может использоваться повторно и многократно. У fragment может быть собственный файл layout, а также есть свой собственный жизненный цикл. Fragment существует в контексте activity и имеет свой жизненный цикл, вне activity обособлено он существовать не может. Каждая activity может иметь несколько fragment.

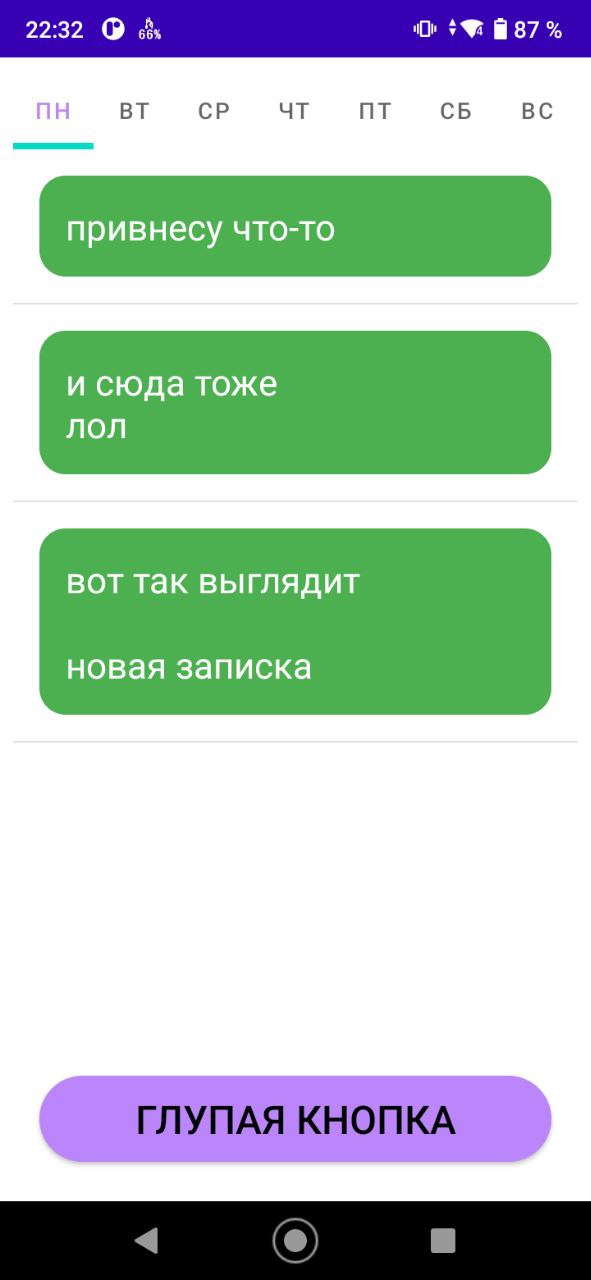
**Ход работы**

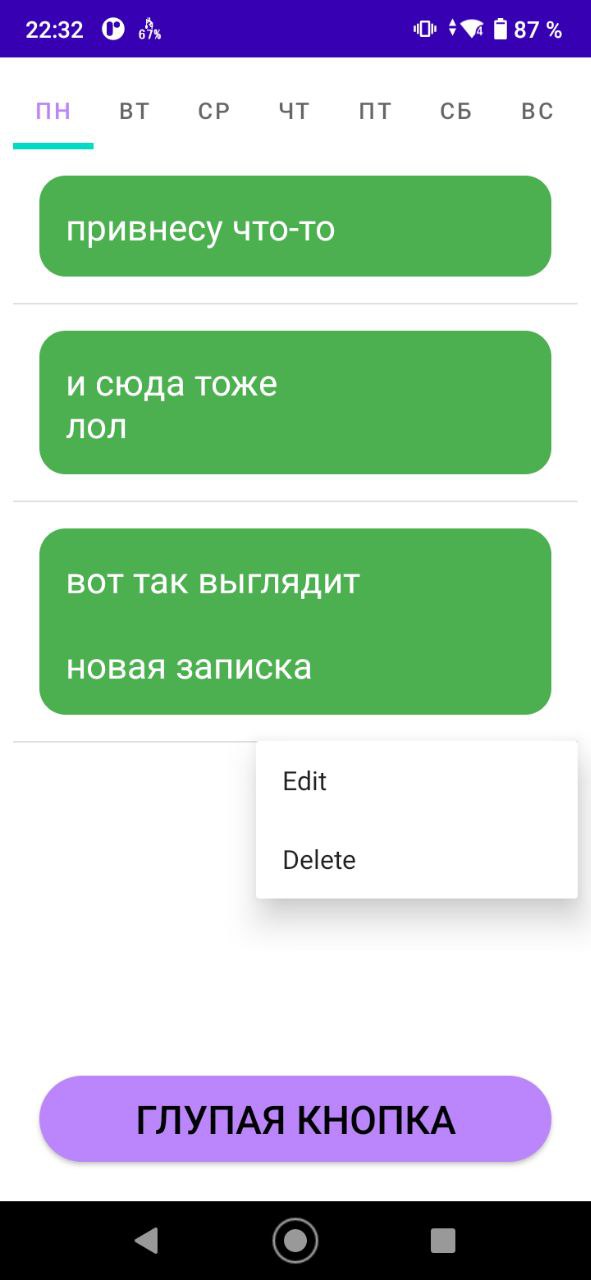
  
Рисунок 1 – Основной экран

На рисунке можно сверху увидеть вкладки «ПН», «ВТ» и т.д., обозначающие дни недели. Набор вкладок представляет собой элемент TabLayout, который управляется элементом ViewPager2. Каждая вкладка представляет собой отдельный Fragment со списком элементов RecyclerView. Список отображается из базы данных (отображает новые данные при обновлении базы данных). Элемент списка – карточка с текстом заметки. По нажатию на любой элемент откроется контекстное меню PopupMenu с двумя кнопками – Edit и Delete (рисунок 4). Также снизу можно увидеть кнопку, по нажатию на которую откроется окно DialogFragment, где можно написать новую заметку и добавить её в список.

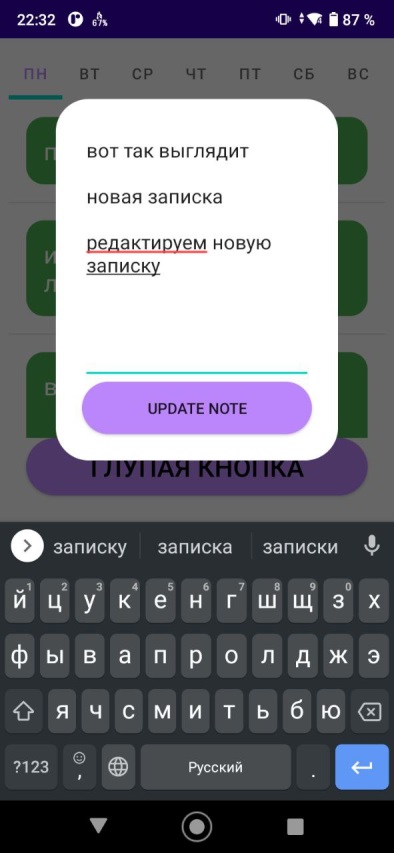
  
Рисунок 2 – Новая заметка

Новая заметка – окно DialogFragment, где можно установить любой Layout. В моём случае это EditText для записи текста и Button для добавления заметки. После нажатия на кнопку заметка добавится в базу данных и окно закроется.

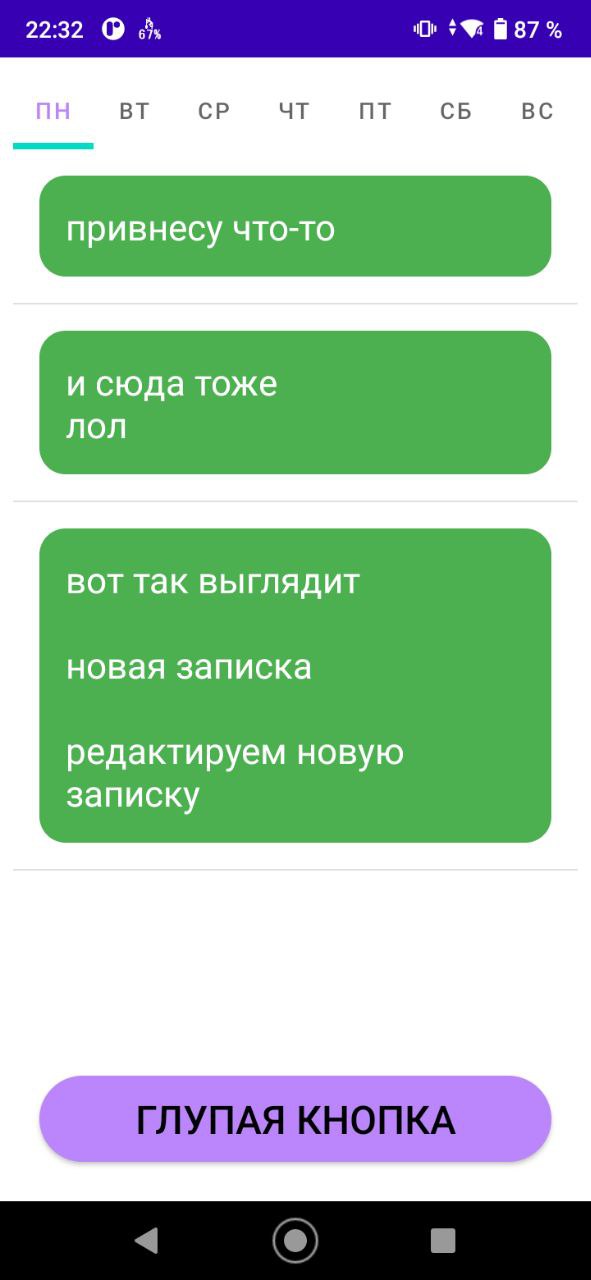
  
Рисунок 3 – Заметка добавлена в список

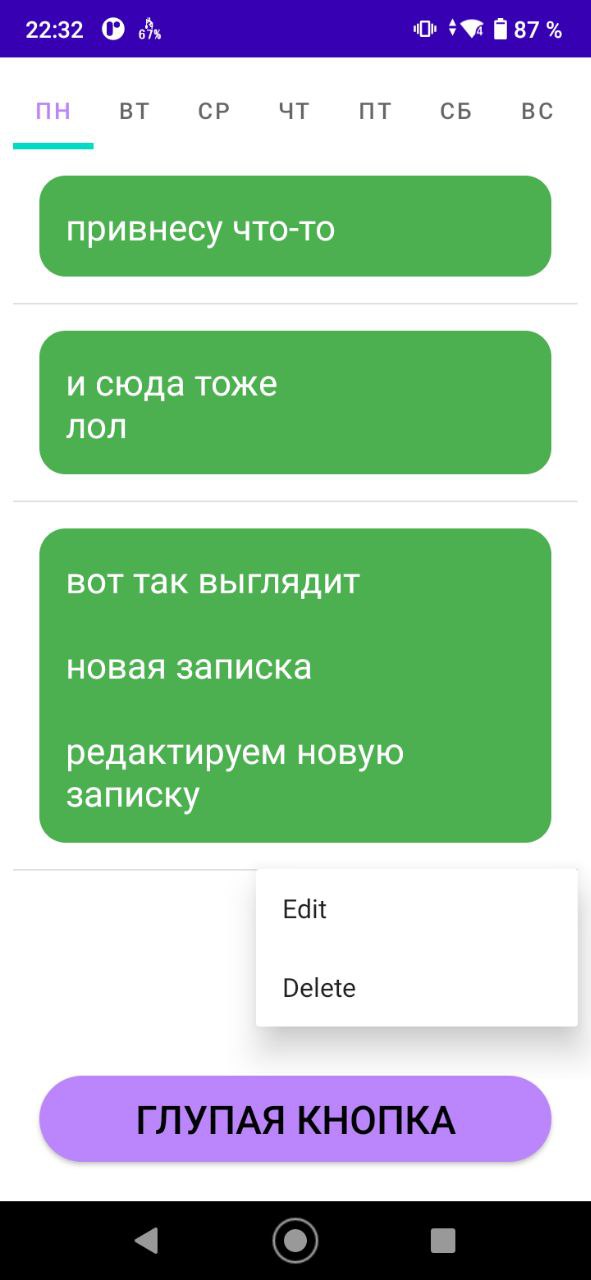
  
Рисунок 4 – Контекстное меню элемента RecyclerView

Контекстное меню – 2 кнопки. При нажатии на Edit откроется окно DialogFragment для редактирования заметки (рисунок 5). При нажатии на Delete заметка будет удалена (рисунок 7-8). После нажатия на любую из кнопок меню закроется.

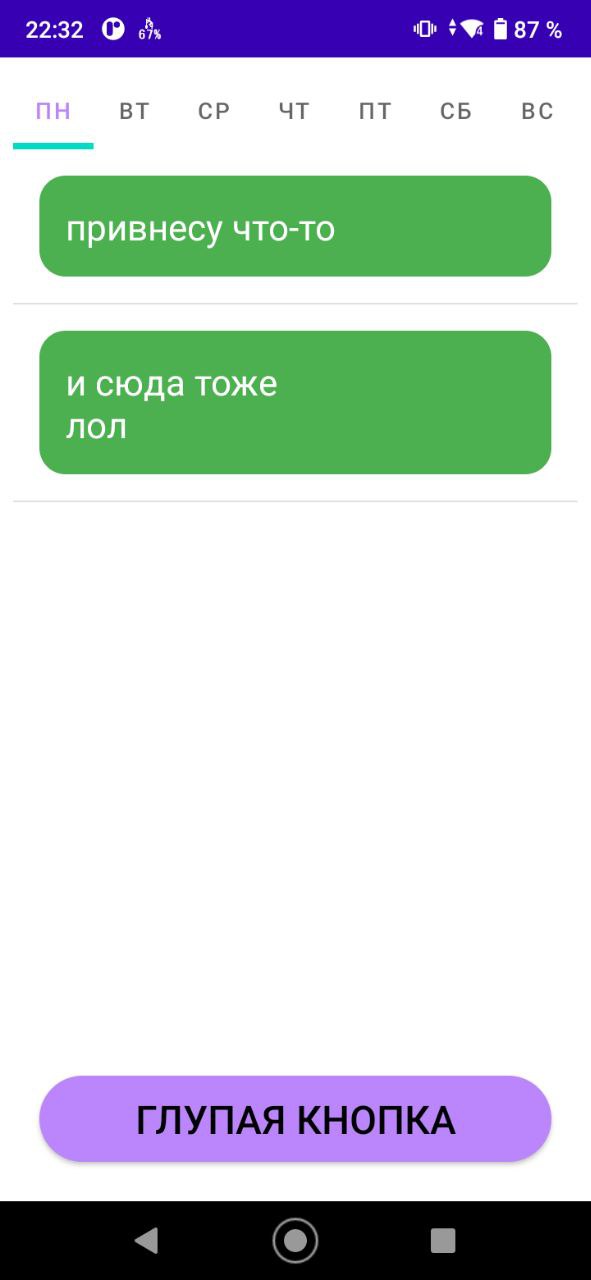
  
Рисунок 5 – Редактирование заметки

Похоже на окно создания заметки. Отличие в названии кнопки, ну и в её функциональности, конечно. После нажатия на кнопку заметка обновится в базе данных и окно закроется.

  
Рисунок 6 – Заметка обновлена

  
Рисунок 7 – Снова контекстное меню (для наглядности)

Нажмём кнопку Delete и (рисунок 8)…

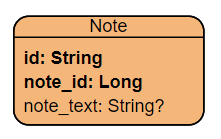
  
Рисунок 8 – Заметка удалена

Заметка удалена из базы данных и список обновился.

  
Рисунок 9 – Пример отображения другой вкладки

Кстати, здесь можно увидеть, как выглядит нижняя кнопка в нажатом состоянии. Кнопки в окнах DialogFragment в нажатом состоянии выглядят также, потому что их background ссылается на один xml-ресурс из папки ресурсов drawable.

**Структура базы данных**

  
Рисунок 10 – Сущность

Сущность имеет составной ключ (id, note\_id). id – день недели (например «ПН»). note\_id – уникальный числовой идентификатор заметки для каждого дня недели. note\_text – текст, хранящийся в заметке, который также может отсутствовать. Работа с базой данных в приложении осуществляется с помощью библиотеки Room.

**Листинг**

Класс App

package com.darf.android.uni1  
  
import android.app.Application  
import androidx.room.Room  
  
class App : Application() {  
 companion object {  
 private lateinit var instance: App  
 fun getInstance() = instance  
 }  
  
 override fun onCreate() {  
 super.onCreate()  
 instance = this  
 }   
}

Класс AppDatabase

package com.darf.android.uni1  
  
import android.content.Context  
import androidx.room.Database  
import androidx.room.Room  
import androidx.room.RoomDatabase  
  
@Database(entities = [Note::class], version = 3, exportSchema = false)  
abstract class AppDatabase : RoomDatabase() {  
 abstract fun noteDao(): NoteDao  
  
 companion object {  
 @Volatile  
 private var INSTANCE: AppDatabase? = null  
 private const val DB\_NAME = "database"  
  
 fun getDatabase(context: Context): AppDatabase {  
 val tempInstance = INSTANCE  
 if (tempInstance != null)  
 return tempInstance  
 return INSTANCE  
 ?: *synchronized*(this) **{**  
 val instance = Room.databaseBuilder(  
 context.*applicationContext*,  
 AppDatabase::class.*java*,  
 DB\_NAME  
 )   
 .fallbackToDestructiveMigration()  
 .build()  
 INSTANCE = instance  
 return instance  
 **}** }  
 }  
}

Класс MainActivity

package com.darf.android.uni1  
  
import android.os.Bundle  
import androidx.fragment.app.FragmentActivity  
import androidx.viewpager2.widget.ViewPager2  
import com.google.android.material.tabs.TabLayout  
import com.google.android.material.tabs.TabLayoutMediator  
  
class MainActivity : FragmentActivity() {  
 private lateinit var tabListAdapter: TabListAdapter  
 private lateinit var viewPager: ViewPager2  
 private lateinit var tabLayout: TabLayout  
 private val tabNames: Array<String> by *lazy* **{** *resources*.getStringArray(R.array.*weekdays*) **}** override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
 *title* = "Notes"  
  
 tabListAdapter = TabListAdapter(this)  
 viewPager = findViewById(R.id.*pager*)  
 viewPager.*adapter* = tabListAdapter  
  
 tabLayout = findViewById(R.id.*tab\_layout*)  
 TabLayoutMediator(tabLayout, viewPager) **{** tab, position **->** tab.*text* = tabNames[position]  
 **}**.attach()  
 }  
}

Класс NewNoteFragment

package com.darf.android.uni1  
  
import android.graphics.Color  
import android.graphics.drawable.ColorDrawable  
import android.os.Build  
import android.os.Bundle  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.widget.Button  
import android.widget.EditText  
import androidx.annotation.RequiresApi  
import androidx.fragment.app.DialogFragment  
  
class NewNoteFragment(private val fragment: TabFragment) : DialogFragment() {  
 @RequiresApi(Build.VERSION\_CODES.*O*)  
 override fun onCreateView(  
 inflater: LayoutInflater,  
 container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 val v = inflater.inflate(R.layout.*fragment\_new\_note*, container) // null  
 v.findViewById<Button>(R.id.*new\_note\_button*).setOnClickListener **{** val newText: String =  
 v.findViewById<EditText>(R.id.*new\_note\_text*).*text*.toString()  
 fragment.addItemClick(newText)  
 dismiss()  
 **}** return v  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 *dialog*!!.*window*!!.setBackgroundDrawable(ColorDrawable(Color.*TRANSPARENT*));  
 }  
}

Класс Note

package com.darf.android.uni1  
  
import androidx.room.Entity  
import java.io.Serializable  
import java.time.LocalDate  
  
@Entity(primaryKeys = ["id", "note\_id"])  
data class Note(  
 val id: String,  
 val note\_id: Long,  
 val note\_text: String?  
) : Serializable

Интерфейс NoteDao

package com.darf.android.uni1  
  
import androidx.room.Dao  
import androidx.room.Delete  
import androidx.room.Insert  
import androidx.room.Query  
import androidx.room.Update  
import io.reactivex.rxjava3.core.Completable  
import io.reactivex.rxjava3.core.Maybe  
  
@Dao  
interface NoteDao {  
 @Query("select \* from note")  
 fun getAll(): Maybe<List<Note>>  
  
 @Query("select \* from note where id = :id")  
 fun getById(id: String): Maybe<List<Note>>  
  
 @Insert  
 fun insert(note: Note): Completable  
  
 @Update  
 fun update(note: Note): Completable  
  
 @Delete  
 fun delete(note: Note): Completable  
}

Класс NoteListAdapter

package com.darf.android.uni1  
  
import android.view.\*  
import android.widget.PopupMenu  
import android.widget.TextView  
import androidx.recyclerview.widget.RecyclerView  
  
class NoteListAdapter(  
 private val onUpdateClicked: (note: Note) -> Unit,  
 private val onDeleteClicked: (note: Note) -> Unit  
) :  
 RecyclerView.Adapter<NoteListAdapter.ViewHolder>() {  
  
 private var noteList = mutableListOf<Note>()  
  
 inner class ViewHolder(  
 itemView: View,  
 private val onUpdateClicked: (note: Note) -> Unit,  
 private val onDeleteClicked: (note: Note) -> Unit  
 ) :  
 RecyclerView.ViewHolder(itemView) {  
 private var note: Note? = null  
 private val itemTextView: TextView = itemView.findViewById(R.id.item\_text)  
  
 fun bind(data: Note) {  
 note = data  
 itemTextView.text = data.note\_text  
 itemView.setOnClickListener **{** val popup = PopupMenu(itemView.context, itemView)  
 popup.setOnMenuItemClickListener(onEditMenu)  
 popup.inflate(R.menu.menu\_note\_item)  
 popup.gravity = Gravity.END  
 popup.show()  
 **}** }  
  
 private val onEditMenu: PopupMenu.OnMenuItemClickListener =  
 PopupMenu.OnMenuItemClickListener **{** item **->** when (item.itemId) {  
 R.id.edit\_item -> onUpdateClicked(note!!)  
 R.id.delete\_item -> onDeleteClicked(note!!)  
 }  
 return@OnMenuItemClickListener true  
 **}** }  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {  
 val itemView = LayoutInflater.from(parent.context)  
 .inflate(R.layout.note\_list\_item, parent, false)  
 return ViewHolder(  
 itemView,  
 onUpdateClicked,  
 onDeleteClicked  
 )  
 }  
  
 override fun onBindViewHolder(holder: ViewHolder, position: Int) {  
 holder.bind(noteList[position])  
 }  
  
 override fun getItemCount(): Int = noteList.size  
  
 fun addItem(note: Note) {  
 noteList.add(note)  
 notifyItemInserted(itemCount - 1)  
 }  
  
 fun addItems(list: List<Note>) {  
 noteList.clear()  
 noteList.addAll(list)  
 notifyDataSetChanged()  
 }  
  
 fun getLastItem(): Note? {  
 return if (itemCount == 0) null else noteList[itemCount - 1]  
 }  
  
 fun removeAt(position: Int): Note {  
 val note = noteList[position]  
 noteList.removeAt(position)  
 notifyItemRemoved(position)  
 return note  
 }  
}

Класс NoteRepository

package com.darf.android.uni1  
  
import io.reactivex.rxjava3.core.Completable  
import io.reactivex.rxjava3.core.Maybe  
  
class NoteRepository(private val noteDao: NoteDao) {  
 fun getNotes(): Maybe<List<Note>> {  
 return noteDao.getAll()  
 }  
  
 fun getNotesById(id: String): Maybe<List<Note>> {  
 return noteDao.getById(id)  
 }  
  
 fun insertNote(note: Note): Completable {  
 return noteDao.insert(note)  
 }  
  
 fun updateNote(note: Note): Completable {  
 return noteDao.update(note)  
 }  
  
 fun deleteNote(note: Note): Completable {  
 return noteDao.delete(note)  
 }  
}

Класс NoteViewModel

package com.darf.android.uni1  
  
import android.app.Application  
import androidx.lifecycle.AndroidViewModel  
import androidx.lifecycle.LiveData  
import androidx.lifecycle.MutableLiveData  
import io.reactivex.rxjava3.android.schedulers.AndroidSchedulers  
import io.reactivex.rxjava3.disposables.CompositeDisposable  
import io.reactivex.rxjava3.schedulers.Schedulers  
  
class NoteViewModel(application: Application) : AndroidViewModel(application) {  
 private val compositeDisposable = CompositeDisposable()  
  
 private val \_notes = MutableLiveData<List<Note>>()  
 var notes: LiveData<List<Note>> = \_notes  
  
 private var noteRepository: NoteRepository =  
 NoteRepository(AppDatabase.getDatabase(application).noteDao())  
  
 fun getNotesById(id: String) {  
 noteRepository.getNotesById(id)  
 .subscribeOn(Schedulers.io())  
 .observeOn(AndroidSchedulers.mainThread())  
 .subscribe **{** if (!it.isNullOrEmpty()) {  
 \_notes.postValue(it)  
 } else {  
 \_notes.postValue(listOf())  
 }  
 **}**.let **{** compositeDisposable.add(it)  
 **}** }  
  
 override fun onCleared() {  
 compositeDisposable.dispose()  
 compositeDisposable.clear()  
 super.onCleared()  
 }  
  
 fun updateNote(note: Note) {  
 noteRepository.updateNote(note)  
 .subscribeOn(Schedulers.io())  
 .observeOn(AndroidSchedulers.mainThread())  
 .subscribe(**{** getNotesById(note.id) **}**, **{}**)  
 .let **{** compositeDisposable.add(it) **}** }  
  
 fun saveNote(note: Note) {  
 noteRepository.insertNote(note)  
 .subscribeOn(Schedulers.io())  
 .observeOn(AndroidSchedulers.mainThread())  
 .subscribe(**{** getNotesById(note.id) **}**, **{}**)  
 .let **{** compositeDisposable.add(it) **}** }  
  
 fun deleteNote(note: Note) {  
 noteRepository.deleteNote(note)  
 .subscribeOn(Schedulers.io())  
 .observeOn(AndroidSchedulers.mainThread())  
 .subscribe(**{** getNotesById(note.id) **}**, **{}**)  
 .let **{** compositeDisposable.add(it) **}** }  
}

Класс TabFragment

package com.darf.android.uni1  
  
import android.os.Build  
import android.os.Bundle  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.widget.Button  
import androidx.annotation.RequiresApi  
import androidx.fragment.app.Fragment  
import androidx.lifecycle.ViewModelProvider  
import androidx.recyclerview.widget.DividerItemDecoration  
import androidx.recyclerview.widget.LinearLayoutManager  
import androidx.recyclerview.widget.RecyclerView  
import java.util.\*  
  
class TabFragment(private val position: Int) : Fragment() {  
 private lateinit var noteListView: RecyclerView  
 private lateinit var viewModel: NoteViewModel  
 private lateinit var noteListAdapter: NoteListAdapter  
 private val weekday by *lazy* **{** *resources*.getStringArray(R.array.*weekdays*)[position] **}** override fun onCreateView(  
 inflater: LayoutInflater,  
 container: ViewGroup?,  
 savedInstanceState: Bundle?,  
 ): View? {  
 val view = inflater.inflate(R.layout.*fragment\_note\_list*, container, false)  
 noteListView = view.findViewById(R.id.*note\_list\_view*)  
 noteListView.*apply* **{** setHasFixedSize(true)  
 *layoutManager* = LinearLayoutManager(view.*context*)  
 addItemDecoration(  
 DividerItemDecoration(  
 view.*context*,  
 DividerItemDecoration.*VERTICAL* )  
 )  
 noteListAdapter = NoteListAdapter(  
 this@TabFragment::onUpdateClicked,  
 this@TabFragment::onDeleteClicked  
 )  
 *adapter* = noteListAdapter  
 scrollToPosition(*adapter*!!.*itemCount* - 1)  
 **}** viewModel = ViewModelProvider  
 .AndroidViewModelFactory  
 .getInstance(App.getInstance())  
 .create(NoteViewModel::class.*java*)  
 viewModel.getNotesById(weekday)  
 viewModel.notes.observe(*viewLifecycleOwner*) **{** if (**it**.*isNotEmpty*()) {  
 noteListAdapter.addItems(**it**)  
 }  
 **}** return view  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 val addItemButton: Button = view.findViewById(R.id.*note\_list\_add\_item\_button*)  
 addItemButton.setOnClickListener **{** val fragment = NewNoteFragment(this)  
 fragment.show(requireActivity().*supportFragmentManager*, null)  
 **}** }  
  
 @RequiresApi(Build.VERSION\_CODES.*O*)  
 fun addItemClick(text: String) {  
 val temp = (noteListView.*adapter* as NoteListAdapter).getLastItem()  
 val newNote = Note(  
 weekday,  
 temp?.note\_id?.plus(1) ?: 1,  
 text  
 )  
 viewModel.saveNote(newNote)   
 }  
  
 fun updateItemClick(note: Note, text: String) {  
 val updateNote = Note(  
 note.id,  
 note.note\_id,  
 text  
 )  
 viewModel.updateNote(updateNote)  
 }  
  
 private fun onUpdateClicked(note: Note) {  
 val fragment = UpdateNoteFragment(this, note)  
 fragment.show(requireActivity().*supportFragmentManager*, null)  
 }  
  
 private fun onDeleteClicked(note: Note) {  
 viewModel.deleteNote(note)  
 }  
}

Класс TabListAdapter

package com.darf.android.uni1  
  
import androidx.fragment.app.Fragment  
import androidx.fragment.app.FragmentActivity  
import androidx.viewpager2.adapter.FragmentStateAdapter  
  
class TabListAdapter(private val fragment: FragmentActivity) :  
 FragmentStateAdapter(fragment) {  
 override fun getItemCount(): Int = fragment.*resources*.getStringArray(R.array.*weekdays*).size  
  
 override fun createFragment(position: Int): Fragment {  
 return TabFragment(position)  
 }  
}

Класс UpdateNoteFragment

package com.darf.android.uni1  
  
import android.graphics.Color  
import android.graphics.drawable.ColorDrawable  
import android.os.Build  
import android.os.Bundle  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.widget.Button  
import android.widget.EditText  
import androidx.annotation.RequiresApi  
import androidx.fragment.app.DialogFragment  
  
class UpdateNoteFragment(  
 private val fragment: TabFragment,  
 private val note: Note  
) :  
 DialogFragment() {  
 @RequiresApi(Build.VERSION\_CODES.*O*)  
 override fun onCreateView(  
 inflater: LayoutInflater,  
 container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 val v = inflater.inflate(R.layout.*fragment\_update\_note*, container) // null  
 val updateNoteText = v.findViewById<EditText>(R.id.*update\_note\_text*)  
 updateNoteText.setText(note.note\_text)  
 v.findViewById<Button>(R.id.*update\_note\_button*).setOnClickListener **{** val newText = updateNoteText.*text*.toString()  
 fragment.updateItemClick(note, newText)  
 dismiss()  
 **}** return v  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 *dialog*!!.*window*!!.setBackgroundDrawable(ColorDrawable(Color.*TRANSPARENT*));  
 }  
}