**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное автономное образовательное учреждение

высшего образования «Уральский федеральный университет

имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ

Центр ускоренного обучения

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 28**

по дисциплине «Разработка мобильных приложений»

**Тема: Unit 4 Настройки «Удаление данных из базы данных номеров»**

Студент группы РИЗ-300028у: А.С. Севостьянов

Преподаватель: С.И.Тимошенко,

доц., к.т.н.

**Екатеринбург 2022**

содержание

[1 Постановка задачи 3](#_Toc118041527)

[2 Результаты проделанной работы 4](#_Toc118041528)

[3 Анализ полученных результатов 5](#_Toc118041529)

## 1 Постановка задачи

Обновить приложение RoomWordsSample, чтобы сохранить данные при закрытии приложения. Разрешить пользователям удалять все слова, выбрав пункт меню Параметров. Разрешить пользователям удалять определенное слово, проводя пальцем по элементу в списке. При необходимости, в задаче кодирования, расширьте приложение, чтобы позволить пользователю обновлять существующие слова.

.

## Результаты проделанной работы

Программный код проекта, файл MainActivity.

package com.android.example.roomwordssample;  
  
import android.arch.lifecycle.Observer;  
import android.arch.lifecycle.ViewModelProviders;  
import android.content.Intent;  
import android.os.Bundle;  
import android.support.annotation.Nullable;  
import android.support.design.widget.FloatingActionButton;  
import android.support.v7.app.AppCompatActivity;  
import android.support.v7.widget.LinearLayoutManager;  
import android.support.v7.widget.RecyclerView;  
import android.support.v7.widget.Toolbar;  
import android.support.v7.widget.helper.ItemTouchHelper;  
import android.view.Menu;  
import android.view.MenuItem;  
import android.view.View;  
import android.widget.Toast;  
  
import java.util.List;  
  
*/\*\*  
 \* Этот класс отображает список слов в RecyclerView.  
 \* Слова сохраняются в базе данных комнат.  
 \* Макет для этого действия также отображает FAB, который  
 \* \* позволяет пользователям запускать действие "Новое слово", чтобы добавлять новые слова.  
 \* Пользователи могут удалить слово, проведя по нему пальцем, или удалить все слова  
 \* через меню опций.  
 \* Всякий раз, когда новое слово добавляется, удаляется или обновляется, RecyclerView  
 \* отображение списка слов автоматически обновляется.  
 \*/*public class MainActivity extends AppCompatActivity {  
  
 public static final int *NEW\_WORD\_ACTIVITY\_REQUEST\_CODE* = 1;  
 public static final int *UPDATE\_WORD\_ACTIVITY\_REQUEST\_CODE* = 2;  
  
 public static final String *EXTRA\_DATA\_UPDATE\_WORD* = "extra\_word\_to\_be\_updated";  
 public static final String *EXTRA\_DATA\_ID* = "extra\_data\_id";  
  
 private WordViewModel mWordViewModel;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
  
 Toolbar toolbar = findViewById(R.id.*toolbar*);  
 setSupportActionBar(toolbar);  
  
 // Настройте RecyclerView.  
 RecyclerView recyclerView = findViewById(R.id.*recyclerview*);  
 final WordListAdapter adapter = new WordListAdapter(this);  
 recyclerView.setAdapter(adapter);  
 recyclerView.setLayoutManager(new LinearLayoutManager(this));  
  
 // Настройте WordViewModel.  
 mWordViewModel = ViewModelProviders.*of*(this).get(WordViewModel.class);  
// Получите все слова из базыданных  
// и свяжите их с адаптером.  
 mWordViewModel.getAllWords().observe(this, new Observer<List<Word>>() {  
 @Override  
 public void onChanged(@Nullable final List<Word> words) {  
 // Обновите кэшированную копию слов в адаптере.  
 adapter.setWords(words);  
 }  
 });  
  
 // Настройка плавающей кнопки действия  
 FloatingActionButton fab = findViewById(R.id.*fab*);  
 fab.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 Intent intent = new Intent(MainActivity.this, NewWordActivity.class);  
 startActivityForResult(intent, *NEW\_WORD\_ACTIVITY\_REQUEST\_CODE*);  
 }  
 });  
  
// Добавьте функциональность для пролистывания элементов в  
// RecyclerView, чтобы удалить пролистанный элемент.  
 ItemTouchHelper helper = new ItemTouchHelper(  
 new ItemTouchHelper.SimpleCallback(0,  
 ItemTouchHelper.*LEFT* | ItemTouchHelper.*RIGHT*) {  
 @Override  
 // // Мы не реализуем onMove() в этом приложении.  
 public boolean onMove(RecyclerView recyclerView,  
 RecyclerView.ViewHolder viewHolder,  
 RecyclerView.ViewHolder target) {  
 return false;  
 }  
  
 @Override  
// // Когда пользователь проводит пальцем по слову,  
// удалите это слово из базы данных.  
 public void onSwiped(RecyclerView.ViewHolder viewHolder, int direction) {  
 int position = viewHolder.getAdapterPosition();  
 Word myWord = adapter.getWordAtPosition(position);  
 Toast.*makeText*(MainActivity.this,  
 getString(R.string.*delete\_word\_preamble*) + " " +  
 myWord.getWord(), Toast.*LENGTH\_LONG*).show();  
  
 // Опустите это слово.  
 mWordViewModel.deleteWord(myWord);  
 }  
 });  
// // Прикрепите элемент touch helper к виду recycler.  
 helper.attachToRecyclerView(recyclerView);  
  
 adapter.setOnItemClickListener(new WordListAdapter.ClickListener() {  
  
 @Override  
 public void onItemClick(View v, int position) {  
 Word word = adapter.getWordAtPosition(position);  
 launchUpdateWordActivity(word);  
 }  
 });  
 }  
  
 @Override  
 public boolean onCreateOptionsMenu(Menu menu) {  
 // Раздуть меню; это добавляет элементы на панель действий, если она присутствует.  
 getMenuInflater().inflate(R.menu.*menu\_main*, menu);  
 return true;  
 }  
  
 // В меню опций есть один пункт "Очистить все данные сейчас".  
 // который удаляет все записи в базе данных.  
 @Override  
 public boolean onOptionsItemSelected(MenuItem item) {  
// Обработайте элемент панели действий, нажав здесь. Панель действий будет  
 // автоматически обрабатывать нажатия на кнопку "Домой"/"Вверх", пока  
 // как вы указываете родительское действие в AndroidManifest.xml .  
 int id = item.getItemId();  
  
//отсутствие проверки, упрощаемое утверждение  
 if (id == R.id.*clear\_data*) {  
// Добавьте тост только для подтверждения  
 Toast.*makeText*(this, R.string.*clear\_data\_toast\_text*, Toast.*LENGTH\_LONG*).show();  
  
// Удалите существующие данные.  
 mWordViewModel.deleteAll();  
 return true;  
 }  
 return super.onOptionsItemSelected(item);  
 }  
  
 */\*\*  
 \* \* Когда пользователь вводит новое слово в NewWordActivity,  
 \* это действие возвращает результат этому действию.  
 \* Если пользователь ввел новое слово, сохраните его в базе данных.  
 \*/* public void onActivityResult(int requestCode, int resultCode, Intent data) {  
 super.onActivityResult(requestCode, resultCode, data);  
  
 if (requestCode == *NEW\_WORD\_ACTIVITY\_REQUEST\_CODE* && resultCode == *RESULT\_OK*) {  
 Word word = new Word(data.getStringExtra(NewWordActivity.*EXTRA\_REPLY*));  
 // Сохраните данные.  
 mWordViewModel.insert(word);  
 } else if (requestCode == *UPDATE\_WORD\_ACTIVITY\_REQUEST\_CODE* && resultCode == *RESULT\_OK*) {  
 String word\_data = data.getStringExtra(NewWordActivity.*EXTRA\_REPLY*);  
 int id = data.getIntExtra(NewWordActivity.*EXTRA\_REPLY\_ID*, -1);  
  
 if (id != -1) {  
 mWordViewModel.update(new Word(id, word\_data));  
 } else {  
 Toast.*makeText*(this, R.string.*unable\_to\_update*,  
 Toast.*LENGTH\_LONG*).show();  
 }  
 } else {  
 Toast.*makeText*(  
 this, R.string.*empty\_not\_saved*, Toast.*LENGTH\_LONG*).show();  
 }  
 }  
  
 public void launchUpdateWordActivity( Word word) {  
 Intent intent = new Intent(this, NewWordActivity.class);  
 intent.putExtra(*EXTRA\_DATA\_UPDATE\_WORD*, word.getWord());  
 intent.putExtra(*EXTRA\_DATA\_ID*, word.getId());  
 startActivityForResult(intent, *UPDATE\_WORD\_ACTIVITY\_REQUEST\_CODE*);  
 }  
}

Программный код проекта, файл NewWordActivity.

package com.android.example.roomwordssample;  
  
import android.content.Intent;  
import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity;  
import android.text.TextUtils;  
import android.view.View;  
import android.widget.Button;  
import android.widget.EditText;  
  
import static com.android.example.roomwordssample.MainActivity.*EXTRA\_DATA\_ID*;  
import static com.android.example.roomwordssample.MainActivity.*EXTRA\_DATA\_UPDATE\_WORD*;  
  
*/\*\*  
 \* Этот класс отображает экран, на котором пользователь вводит новое слово.  
 \* \* NewWordActivity возвращает введенное слово вызывающему действию  
 \* (MainActivity), который затем сохраняет новое слово и обновляет список  
 \* отображаемых слов.  
 \*/*public class NewWordActivity extends AppCompatActivity {  
  
 public static final String *EXTRA\_REPLY* = "com.example.android.roomwordssample.REPLY";  
 public static final String *EXTRA\_REPLY\_ID* = "com.android.example.roomwordssample.REPLY\_ID";  
  
 private EditText mEditWordView;  
  
 @Override  
 public void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_new\_word*);  
  
 mEditWordView = findViewById(R.id.*edit\_word*);  
 int id = -1 ;  
  
 final Bundle extras = getIntent().getExtras();  
  
// Если нам передается контент, заполните его для редактирования пользователем.  
 if (extras != null) {  
 String word = extras.getString(*EXTRA\_DATA\_UPDATE\_WORD*, "");  
 if (!word.isEmpty()) {  
 mEditWordView.setText(word);  
 mEditWordView.setSelection(word.length());  
 mEditWordView.requestFocus();  
 }  
 } // В противном случае начните с пустых полей.  
  
  
 final Button button = findViewById(R.id.*button\_save*);  
  
// Когда пользователь нажимает кнопку Сохранить, создайте новое намерение для ответа.  
 // // Намерение ответа будет отправлено обратно вызывающему действию (в данном случае MainActivity).  
 button.setOnClickListener(new View.OnClickListener() {  
 public void onClick(View view) {  
 // Создайте новое намерение для ответа.  
 Intent replyIntent = new Intent();  
 if (TextUtils.*isEmpty*(mEditWordView.getText())) {  
 // Получить новое слово, введенное пользователем. // Ни одно слово не было введено, установите результат соответствующим образом.  
 setResult(*RESULT\_CANCELED*, replyIntent);  
 } else {  
 // Получить новое слово, введенное пользователем.  
 String word = mEditWordView.getText().toString();  
 // Добавьте новое слово в дополнение к намерению ответа.  
 replyIntent.putExtra(*EXTRA\_REPLY*, word);  
 if (extras != null && extras.containsKey(*EXTRA\_DATA\_ID*)) {  
 int id = extras.getInt(*EXTRA\_DATA\_ID*, -1);  
 if (id != -1) {  
 replyIntent.putExtra(*EXTRA\_REPLY\_ID*, id);  
 }  
 }  
 // Установите статус результата таким образом, чтобы он указывал на успех.  
 setResult(*RESULT\_OK*, replyIntent);  
 }  
 finish();  
 }  
 });  
 }  
}

Программный код проекта, файл Word.

package com.android.example.roomwordssample;  
  
import android.arch.persistence.room.ColumnInfo;  
import android.arch.persistence.room.Entity;  
import android.arch.persistence.room.Ignore;  
import android.arch.persistence.room.PrimaryKey;  
import android.support.annotation.NonNull;  
  
*/\*\*  
 \* Класс сущности, представляющий слово в базе данных  
 \*/*@Entity(tableName = "word\_table")  
public class Word {  
  
 @PrimaryKey(autoGenerate = true)  
 private int id;  
  
 @NonNull  
 @ColumnInfo(name = "word")  
 private String mWord;  
  
 public Word(@NonNull String word) {  
 this.mWord = word;  
 }  
  
 /\*  
 \* Этот конструктор аннотируется с помощью @Ignore, потому что Room ожидает только  
 \* один конструктор по умолчанию в классе сущностей.  
 \*/  
  
 @Ignore  
 public Word(int id, @NonNull String word) {  
 this.id = id;  
 this.mWord = word;  
 }  
  
 public String getWord() {  
 return this.mWord;  
 }  
  
 public int getId() {return id;}  
  
 public void setId(int id) {  
 this.id = id;  
 }  
}

Программный код проекта, файл WordDao.

package com.android.example.roomwordssample;  
  
import android.arch.lifecycle.LiveData;  
import android.arch.persistence.room.Dao;  
import android.arch.persistence.room.Delete;  
import android.arch.persistence.room.Insert;  
import android.arch.persistence.room.OnConflictStrategy;  
import android.arch.persistence.room.Query;  
import android.arch.persistence.room.Update;  
  
import java.util.List;  
  
*/\*\*  
 \* Объект доступа к данным (DAO) для слова.  
 \* Каждый метод выполняет операцию с базой данных, такую как вставка или удаление слова,  
 \* выполнение запроса к базе данных или удаление всех слов.  
 \*/*@Dao  
public interface WordDao {  
  
 @Insert(onConflict = OnConflictStrategy.*IGNORE*)  
 void insert(Word word);  
  
 @Query("DELETE FROM word\_table")  
 void deleteAll();  
  
 @Delete  
 void deleteWord(Word word);  
  
 @Query("SELECT \* from word\_table LIMIT 1")  
 Word[] getAnyWord();  
  
 @Query("SELECT \* from word\_table ORDER BY word ASC")  
 LiveData<List<Word>> getAllWords();  
  
 @Update  
 void update(Word... word);  
}

Программный код проекта, файл WordListAdapter.

package com.android.example.roomwordssample;  
  
import android.content.Context;  
import android.support.v7.widget.RecyclerView;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.TextView;  
  
import java.util.List;  
  
*/\*\*  
 \* Адаптер для RecyclerView, который отображает список слов.  
 \*/*public class WordListAdapter extends RecyclerView.Adapter<WordListAdapter.WordViewHolder> {  
  
 private final LayoutInflater mInflater;  
 private List<Word> mWords; // Кэшированная копия слов  
 private static ClickListener *clickListener*;  
  
 WordListAdapter(Context context) {  
 mInflater = LayoutInflater.*from*(context);  
 }  
  
 @Override  
 public WordViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {  
 View itemView = mInflater.inflate(R.layout.*recyclerview\_item*, parent, false);  
 return new WordViewHolder(itemView);  
 }  
  
 @Override  
 public void onBindViewHolder(WordViewHolder holder, int position) {  
 if (mWords != null) {  
 Word current = mWords.get(position);  
 holder.wordItemView.setText(current.getWord());  
 } else {  
 // Охватывает случай, когда данные еще не готовы.  
 holder.wordItemView.setText(R.string.*no\_word*);  
 }  
 }  
  
 */\*\*  
 \* Связывает список слов с этим адаптером  
 \*/* void setWords(List<Word> words) {  
 mWords = words;  
 notifyDataSetChanged();  
 }  
  
 */\*\*  
 \* getItemCount() вызывается много раз, и при первом вызове  
 \* \* words не был обновлен (означает, что изначально он равен null, и мы не можем вернуть null).  
 \*/* @Override  
 public int getItemCount() {  
 if (mWords != null)  
 return mWords.size();  
 else return 0;  
 }  
  
 */\*\*  
 \* Получает слово в заданной позиции.  
 \* Этот метод полезен для определения того, какое слово  
 \* был нажат или проведен в методах, которые обрабатывают пользовательские события.  
 \*/* public Word getWordAtPosition(int position) {  
 return mWords.get(position);  
 }  
  
 class WordViewHolder extends RecyclerView.ViewHolder {  
 private final TextView wordItemView;  
  
 private WordViewHolder(View itemView) {  
 super(itemView);  
 wordItemView = itemView.findViewById(R.id.*textView*);  
 itemView.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 *clickListener*.onItemClick(view, getAdapterPosition());  
 }  
 });  
 }  
 }  
  
 public void setOnItemClickListener(ClickListener clickListener) {  
 WordListAdapter.*clickListener* = clickListener;  
 }  
  
 public interface ClickListener {  
 void onItemClick(View v, int position);  
 }  
  
}

Программный код проекта, файл WordRepository.

package com.android.example.roomwordssample;  
  
import android.app.Application;  
import android.arch.lifecycle.LiveData;  
import android.os.AsyncTask;  
  
import java.util.List;  
  
*/\*\*  
 \* Этот класс содержит код реализации для методов, которые взаимодействуют с базой данных.  
 \* Использование репозитория позволяет нам группировать методы реализации вместе,  
 \* и позволяет WordViewModel быть чистым интерфейсом между остальной частью приложения  
 \* и база данных.  
 \*  
 \* Для вставки, обновления и удаления, а также для более длительных запросов,  
 \* вы должны запустить методы взаимодействия с базой данных в фоновом режиме.  
 \*  
 \* Как правило, все, что вам нужно сделать для реализации метода базы данных  
 \* заключается в вызове его на объекте доступа к данным (DAO), в фоновом режиме, если применимо.  
 \*/*public class WordRepository {  
  
 private WordDao mWordDao;  
 private LiveData<List<Word>> mAllWords;  
  
 WordRepository(Application application) {  
 WordRoomDatabase db = WordRoomDatabase.*getDatabase*(application);  
 mWordDao = db.wordDao();  
 mAllWords = mWordDao.getAllWords();  
 }  
  
 LiveData<List<Word>> getAllWords() {  
 return mAllWords;  
 }  
  
 public void insert(Word word) {  
 new insertAsyncTask(mWordDao).execute(word);  
 }  
  
 public void update(Word word) {  
 new updateWordAsyncTask(mWordDao).execute(word);  
 }  
  
 public void deleteAll() {  
 new deleteAllWordsAsyncTask(mWordDao).execute();  
 }  
  
 // Должен выполняться из основного потока  
 public void deleteWord(Word word) {  
 new deleteWordAsyncTask(mWordDao).execute(word);  
 }  
  
 // Статические внутренние классы ниже здесь для запуска взаимодействия с базой данных в фоновом режиме.  
  
 */\*\*  
 \* Вставляет слово в базу данных.  
 \*/* private static class insertAsyncTask extends AsyncTask<Word, Void, Void> {  
  
 private WordDao mAsyncTaskDao;  
  
 insertAsyncTask(WordDao dao) {  
 mAsyncTaskDao = dao;  
 }  
  
 @Override  
 protected Void doInBackground(final Word... params) {  
 mAsyncTaskDao.insert(params[0]);  
 return null;  
 }  
 }  
  
 */\*\*  
 \* Удаляет все слова из базы данных (не удаляет таблицу).  
 \*/* private static class deleteAllWordsAsyncTask extends AsyncTask<Void, Void, Void> {  
 private WordDao mAsyncTaskDao;  
  
 deleteAllWordsAsyncTask(WordDao dao) {  
 mAsyncTaskDao = dao;  
 }  
  
 @Override  
 protected Void doInBackground(Void... voids) {  
 mAsyncTaskDao.deleteAll();  
 return null;  
 }  
 }  
  
 */\*\*  
 \* Удаляет одно слово из базы данных.  
 \*/* private static class deleteWordAsyncTask extends AsyncTask<Word, Void, Void> {  
 private WordDao mAsyncTaskDao;  
  
 deleteWordAsyncTask(WordDao dao) {  
 mAsyncTaskDao = dao;  
 }  
  
 @Override  
 protected Void doInBackground(final Word... params) {  
 mAsyncTaskDao.deleteWord(params[0]);  
 return null;  
 }  
 }  
  
 */\*\*  
 \* Обновляет слово в базе данных.  
 \*/* private static class updateWordAsyncTask extends AsyncTask<Word, Void, Void> {  
 private WordDao mAsyncTaskDao;  
  
 updateWordAsyncTask(WordDao dao) {  
 mAsyncTaskDao = dao;  
 }  
  
 @Override  
 protected Void doInBackground(final Word... params) {  
 mAsyncTaskDao.update(params[0]);  
 return null;  
 }  
 }  
}

Программный код проекта, файл WordRoomDatabase.

package com.android.example.roomwordssample;  
  
import android.arch.persistence.db.SupportSQLiteDatabase;  
import android.arch.persistence.room.Database;  
import android.arch.persistence.room.Room;  
import android.arch.persistence.room.RoomDatabase;  
import android.content.Context;  
import android.os.AsyncTask;  
import android.support.annotation.NonNull;  
  
*/\*\*  
 \* База данных Word Room. Включает код для создания базы данных.  
 \* После того, как приложение создаст базу данных, все дальнейшие взаимодействия  
 \* с этим происходит через WordViewModel.  
 \*/*@Database(entities = {Word.class}, version = 2, exportSchema = false)  
public abstract class WordRoomDatabase extends RoomDatabase {  
  
 public abstract WordDao wordDao();  
  
 private static WordRoomDatabase *INSTANCE*;  
  
 public static WordRoomDatabase getDatabase(final Context context) {  
 if (*INSTANCE* == null) {  
 synchronized (WordRoomDatabase.class) {  
 if (*INSTANCE* == null) {  
 // Создайте базу данных здесь.  
 *INSTANCE* = Room.*databaseBuilder*(context.getApplicationContext(),  
 WordRoomDatabase.class, "word\_database")  
 // Стирает и перестраивает вместо миграции, если объекта миграции нет.  
 // Миграция не является частью этого практического.  
 .fallbackToDestructiveMigration()  
 .addCallback(*sRoomDatabaseCallback*)  
 .build();  
 }  
 }  
 }  
 return *INSTANCE*;  
 }  
  
 // Этот обратный вызов вызывается при открытии базы данных.  
 // // В этом случае используйте PopulateDbAsync для заполнения базы данных  
 // с исходным набором данных, если в базе данных нет записей.  
 private static RoomDatabase.Callback *sRoomDatabaseCallback* =  
 new RoomDatabase.Callback(){  
  
 @Override  
 public void onOpen (@NonNull SupportSQLiteDatabase db){  
 super.onOpen(db);  
 new PopulateDbAsync(*INSTANCE*).execute();  
 }  
 };  
  
 // Заполнить базу данных исходным набором данных  
 // только в том случае, если в базе данных нет записей.  
 private static class PopulateDbAsync extends AsyncTask<Void, Void, Void> {  
  
 private final WordDao mDao;  
  
 // Начальный набор данных  
 private static String [] *words* = {"Арсентьев", "Двоеглазов", "Казанцева", "Утемова", "Верхнева",  
 "Гирева", "Воротеленко"};  
  
 PopulateDbAsync(WordRoomDatabase db) {  
 mDao = db.wordDao();  
 }  
  
 @Override  
 protected Void doInBackground(final Void... params) {  
 // Если у нас нет слов, то создайте начальный список слов.  
 if (mDao.getAnyWord().length < 1) {  
 for (int i = 0; i <= *words*.length - 1; i++) {  
 Word word = new Word(*words*[i]);  
 mDao.insert(word);  
 }  
 }  
 return null;  
 }  
 }  
}

Программный код проекта, файл WordViewModel.

package com.android.example.roomwordssample;  
  
import android.app.Application;  
import android.arch.lifecycle.AndroidViewModel;  
import android.arch.lifecycle.LiveData;  
  
import java.util.List;  
  
*/\*\*  
 \* WordViewModel обеспечивает интерфейс между пользовательским интерфейсом и уровнем данных приложения,  
 \* представлен репозиторием  
 \*/*public class WordViewModel extends AndroidViewModel {  
  
 private WordRepository mRepository;  
  
 private LiveData<List<Word>> mAllWords;  
  
 public WordViewModel(Application application) {  
 super(application);  
 mRepository = new WordRepository(application);  
 mAllWords = mRepository.getAllWords();  
 }  
  
 LiveData<List<Word>> getAllWords() {  
 return mAllWords;  
 }  
  
 public void insert(Word word) {  
 mRepository.insert(word);  
 }  
  
 public void deleteAll() {  
 mRepository.deleteAll();  
 }  
  
 public void deleteWord(Word word) {  
 mRepository.deleteWord(word);  
 }  
  
 public void update(Word word) {  
 mRepository.update(word);  
 }  
}

На рисунке 1 показано изначальное состояние программы.

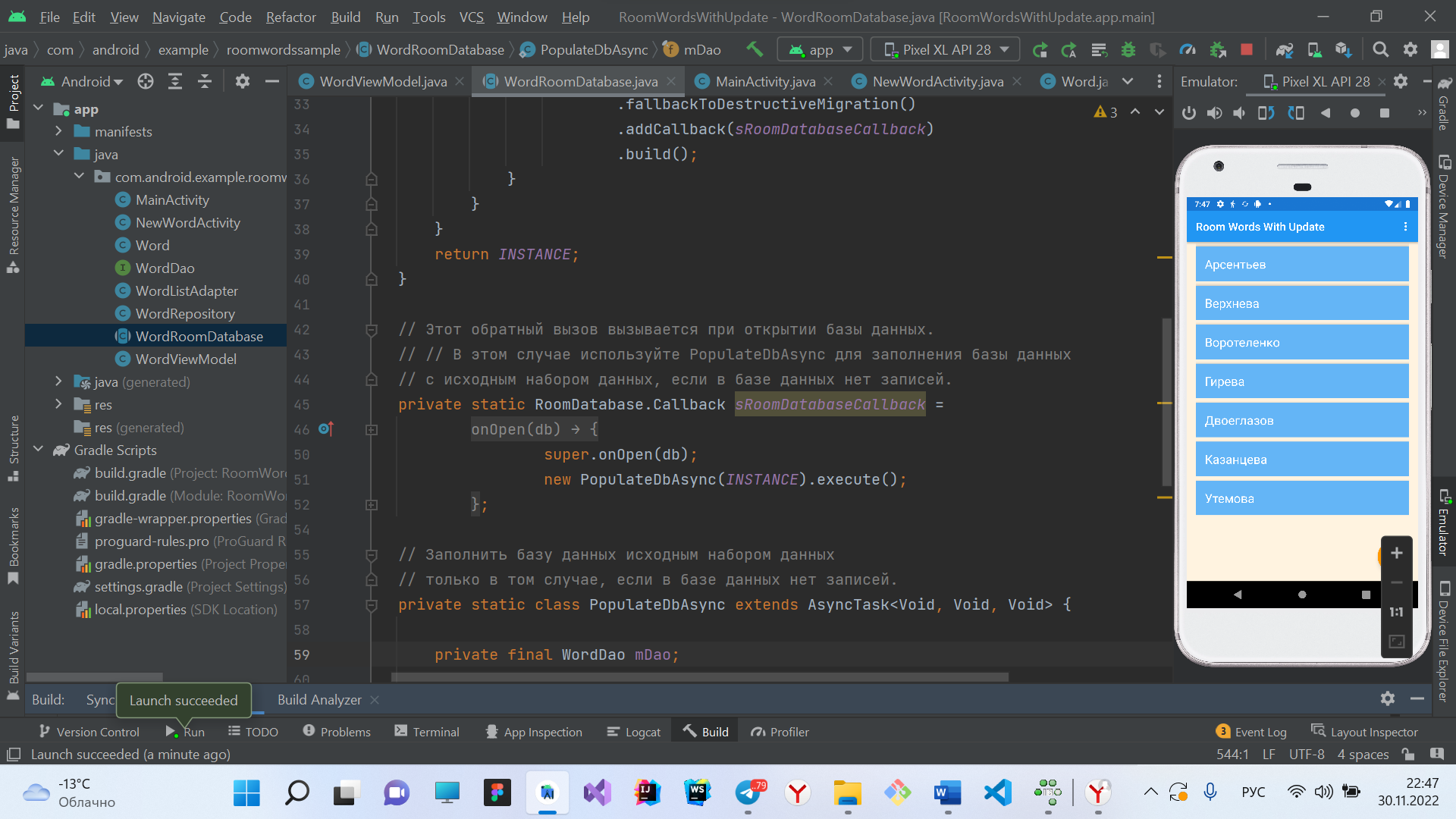


Рисунок 1 – Начальное состояние программы

На рисунке 2 показана страница ввода, нового слова для списка.

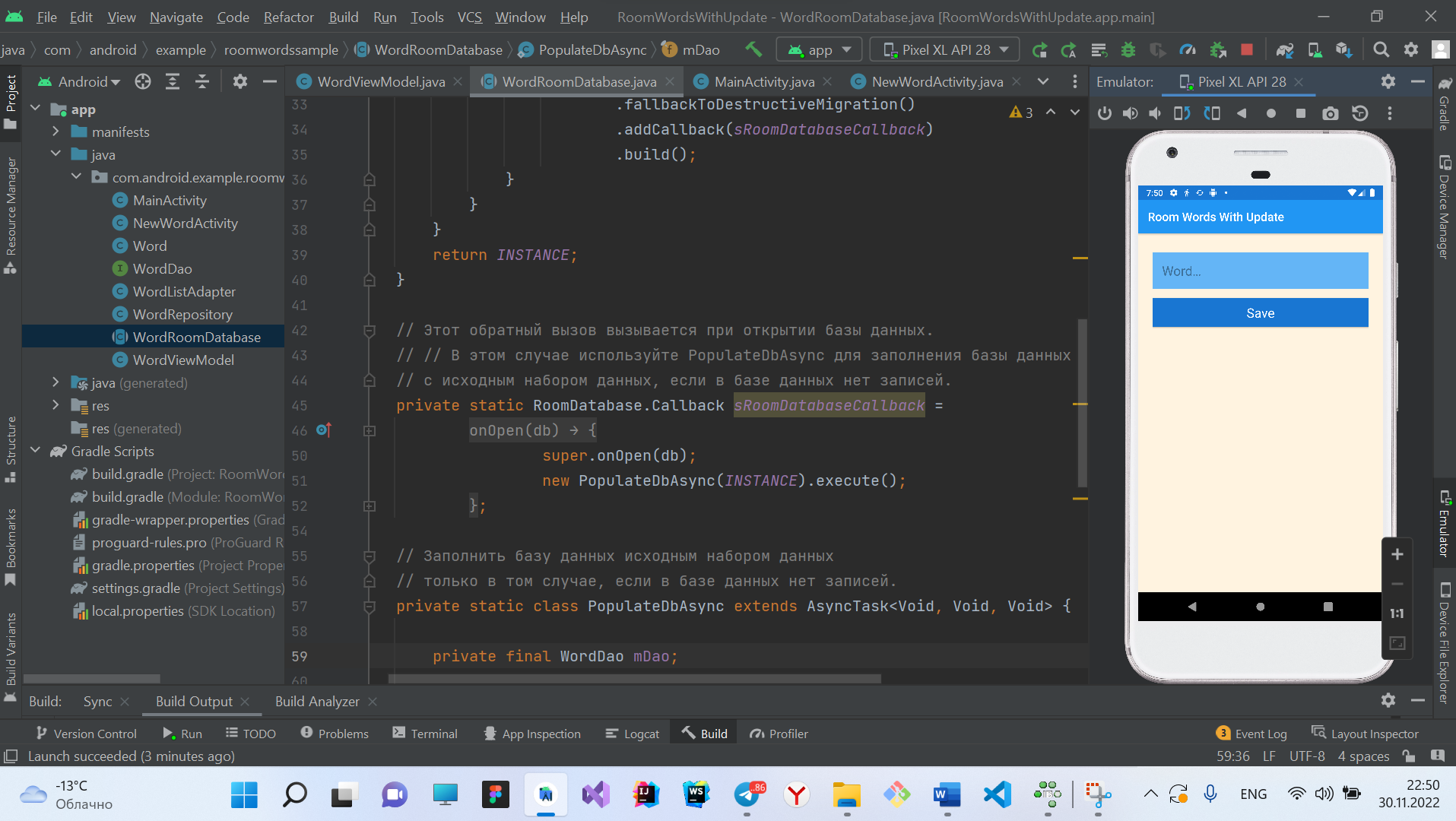


Рисунок 2 – Страница ввода новых слов

На рисунке 3 показано состояние первой страницы после добавления нового слова.

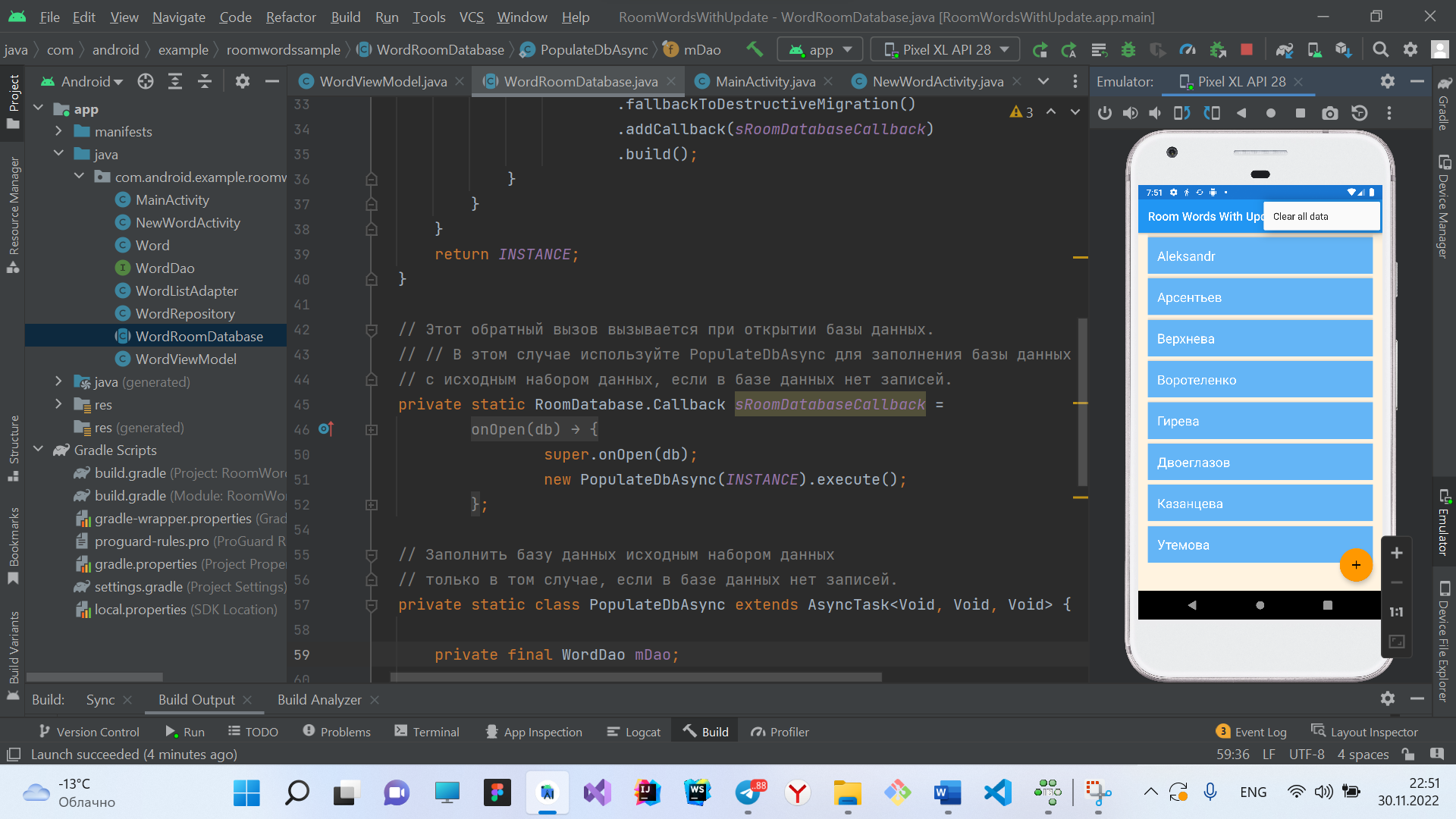


Рисунок 3 – Изменения списка

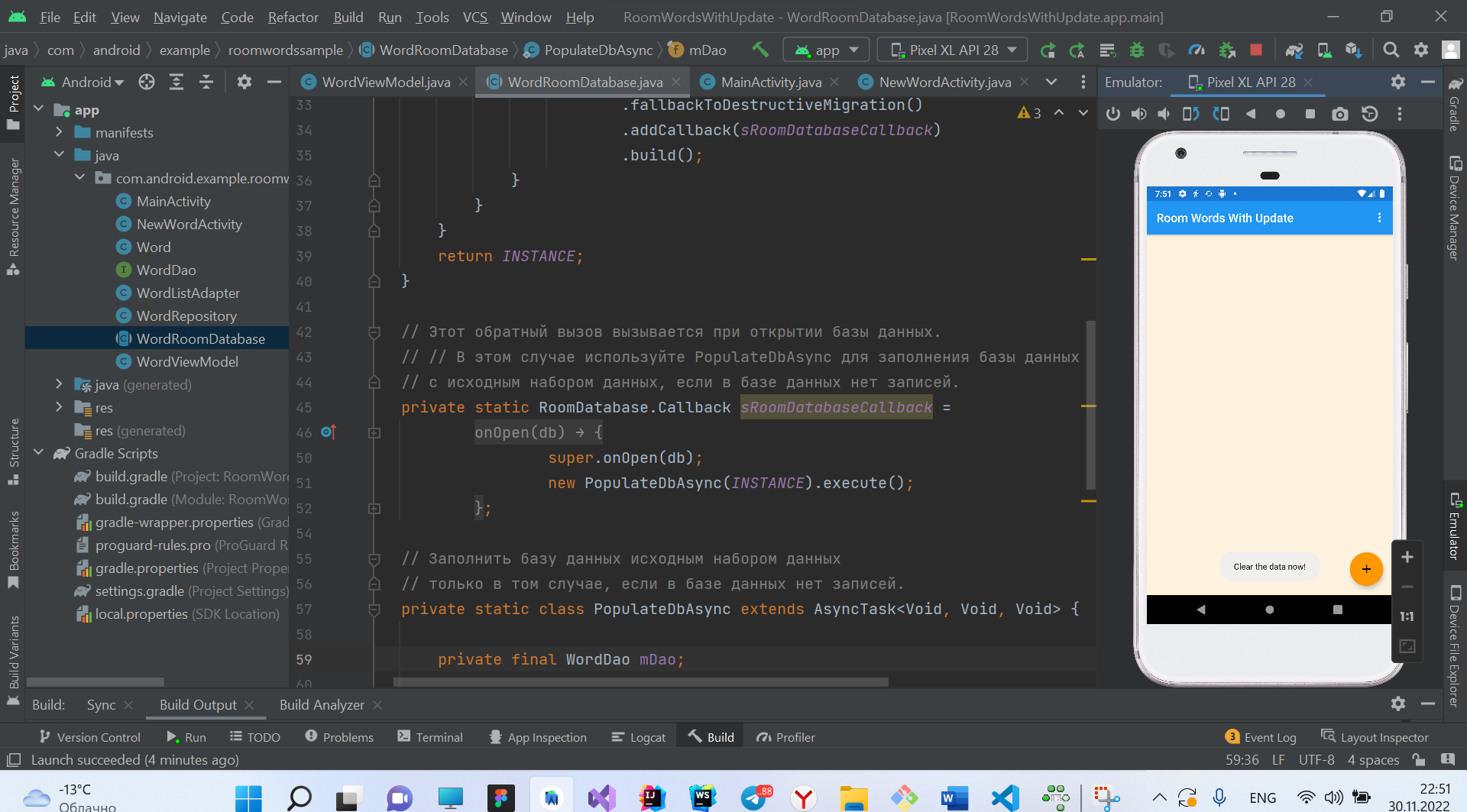
На рисунке 4 показано действия кнопки удалить все значения из списка.

Рисунок 4 – Пустой список

## **Анализ полученных результатов**

В результате выполнения данной лабораторной работы мной был разработан проект, в котором я научился, заполнять базу данных данными только в том случае, если база данных пуста (чтобы пользователи не потеряли изменения, внесенные ими в данные). Как удалить данные из базы данных номеров. Как обновить существующие данные.

.