**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 27**

по дисциплине «Разработка мобильных приложений»

**Тема: Unit 4 Настройки «Room, Data life, ViewModel»**

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

Создать приложение с Activity, которое отображает слова в RecyclerView. Создать Enity элемент, представляющий объекты word. Определить сопоставление SQL-запросов с методами Java в DAO (объект доступа к данным). Использовать LiveData для внесения изменений в данные, видимые в пользовательском интерфейсе, с помощью наблюдателей. Добавить базу данных Room в приложение для локального сохранения данных и инициализируйте базу данных. Абстрактный сервер данных как Repositoryкласс с API, который не зависит от способа хранения или получения данных. Использовать ViewModel, чтобы отделить все операции с данными от пользовательского интерфейса. Добавить второй Activity, который позволяет пользователю добавлять новые слова.

.

## Результаты проделанной работы

Программный код файл, InstantTaskExecutorRule.

package com.example.android.roomwordssample;  
import android.arch.core.executor.ArchTaskExecutor;  
import android.arch.core.executor.TaskExecutor;  
import org.junit.rules.TestWatcher;  
import org.junit.runner.Description;  
  
*/\*\*  
 \* Правило тестирования JUnit, которое заменяет фоновый исполнитель, используемый компонентами архитектуры, на  
 \* другой, который выполняет каждую задачу синхронно.  
 \* Вы можете использовать это правило для тестов на стороне хоста, которые используют компоненты архитектуры.  
 \*/*public class InstantTaskExecutorRule extends TestWatcher {  
 @Override  
 protected void starting(Description description) {  
 super.starting(description);  
 ArchTaskExecutor.*getInstance*().setDelegate(new TaskExecutor() {  
 @Override  
 public void executeOnDiskIO(Runnable runnable) {  
 runnable.run();  
 }  
  
 @Override  
 public void postToMainThread(Runnable runnable) {  
 runnable.run();  
 }  
  
 @Override  
 public boolean isMainThread() {  
 return true;  
 }  
 });  
 }  
  
 @Override  
 protected void finished(Description description) {  
 super.finished(description);  
 ArchTaskExecutor.*getInstance*().setDelegate(null);  
 }  
}

Программый код, файл проекта LiveDataTestUtil.

package com.example.android.roomwordssample;  
import android.arch.lifecycle.LiveData;  
import android.arch.lifecycle.Observer;  
import android.support.annotation.Nullable;  
import java.util.concurrent.CountDownLatch;  
import java.util.concurrent.TimeUnit;  
  
public class LiveDataTestUtil {  
  
 */\*\*  
 \* \* Получите значение из объекта LiveData. Мы ждем выдачи LiveData в течение 2 секунд.  
 \* Как только мы получили уведомление через OnChanged, мы прекращаем наблюдение.  
 \*/* public static <T> T getValue(final LiveData<T> liveData) throws InterruptedException {  
 final Object[] data = new Object[1];  
 final CountDownLatch latch = new CountDownLatch(1);  
 Observer<T> observer = new Observer<T>() {  
 @Override  
 public void onChanged(@Nullable T o) {  
 data[0] = o;  
 latch.countDown();  
 liveData.removeObserver(this);  
 }  
 };  
 liveData.observeForever(observer);  
 latch.await(2, TimeUnit.*SECONDS*);  
 //отсутствие проверки снято  
 return (T) data[0];  
 }  
}

Программный код файл проекта, WordDaoTest.

package com.example.android.roomwordssample;  
import static junit.framework.Assert.*assertEquals*;  
import static junit.framework.Assert.*assertTrue*;  
import android.arch.persistence.room.Room;  
import android.content.Context;  
import android.support.test.InstrumentationRegistry;  
import android.support.test.runner.AndroidJUnit4;  
  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Rule;  
import org.junit.Test;  
import org.junit.runner.RunWith;  
  
import java.io.IOException;  
import java.util.List;  
  
  
*/\*\*  
 \* Это не полный набор тестов. Для простоты большинство ваших образцов не  
 \* включите тесты. Однако при строительстве помещения полезно убедиться, что оно работает, прежде чем  
 \* добавление пользовательского интерфейса.  
 \*/*@RunWith(AndroidJUnit4.class)  
public class WordDaoTest {  
  
 @Rule  
 public InstantTaskExecutorRule instantTaskExecutorRule = new InstantTaskExecutorRule();  
  
 private WordDao mWordDao;  
 private WordRoomDatabase mDb;  
  
 @Before  
 public void createDb() {  
 Context context = InstrumentationRegistry.*getTargetContext*();  
 // Используя базу данных в памяти, потому что информация, хранящаяся здесь, исчезает, когда процесс  
// завершается.  
 mDb = Room.*inMemoryDatabaseBuilder*(context, WordRoomDatabase.class)  
 // Разрешение запросов основного потока, только для тестирования.  
 .allowMainThreadQueries()  
 .build();  
 mWordDao = mDb.wordDao();  
 }  
  
 @After  
 public void closeDb() throws IOException {  
 mDb.close();  
 }  
  
 @Test  
 public void insertAndGetWord() throws Exception {  
 Word word = new Word("word");  
 mWordDao.insert(word);  
 List<Word> allWords = LiveDataTestUtil.*getValue*(mWordDao.getAlphabetizedWords());  
 *assertEquals*(allWords.get(0).getWord(), word.getWord());  
 }  
  
 @Test  
 public void getAllWords() throws Exception {  
 Word word = new Word("aaa");  
 mWordDao.insert(word);  
 Word word2 = new Word("bbb");  
 mWordDao.insert(word2);  
 List<Word> allWords = LiveDataTestUtil.*getValue*(mWordDao.getAlphabetizedWords());  
 *assertEquals*(allWords.get(0).getWord(), word.getWord());  
 *assertEquals*(allWords.get(1).getWord(), word2.getWord());  
 }  
  
 @Test  
 public void deleteAll() throws Exception {  
 Word word = new Word("word");  
 mWordDao.insert(word);  
 Word word2 = new Word("word2");  
 mWordDao.insert(word2);  
 mWordDao.deleteAll();  
 List<Word> allWords = LiveDataTestUtil.*getValue*(mWordDao.getAlphabetizedWords());  
 *assertTrue*(allWords.isEmpty());  
 }  
}

На рисунке 1 показано изначальное состояние программы.

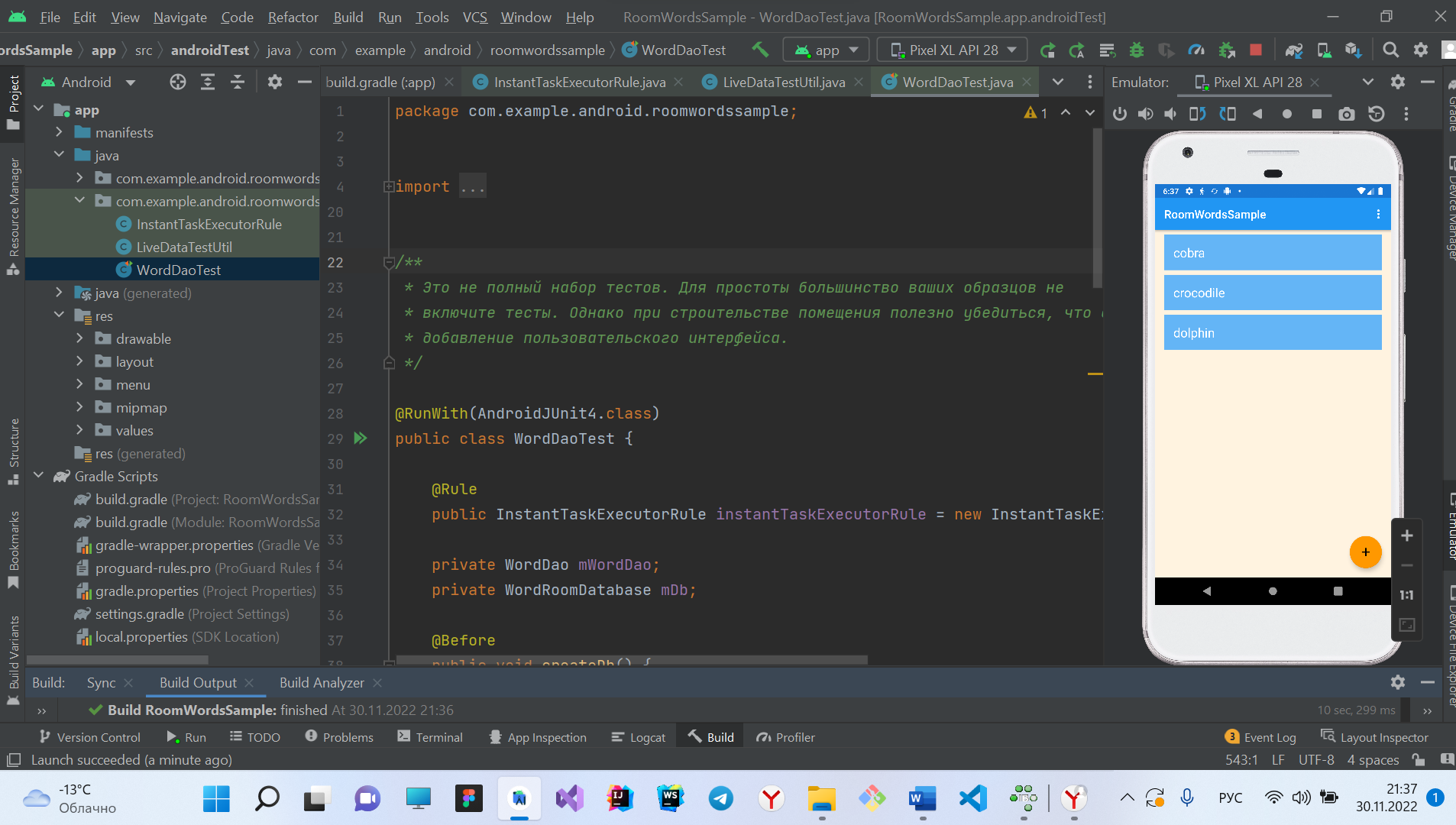
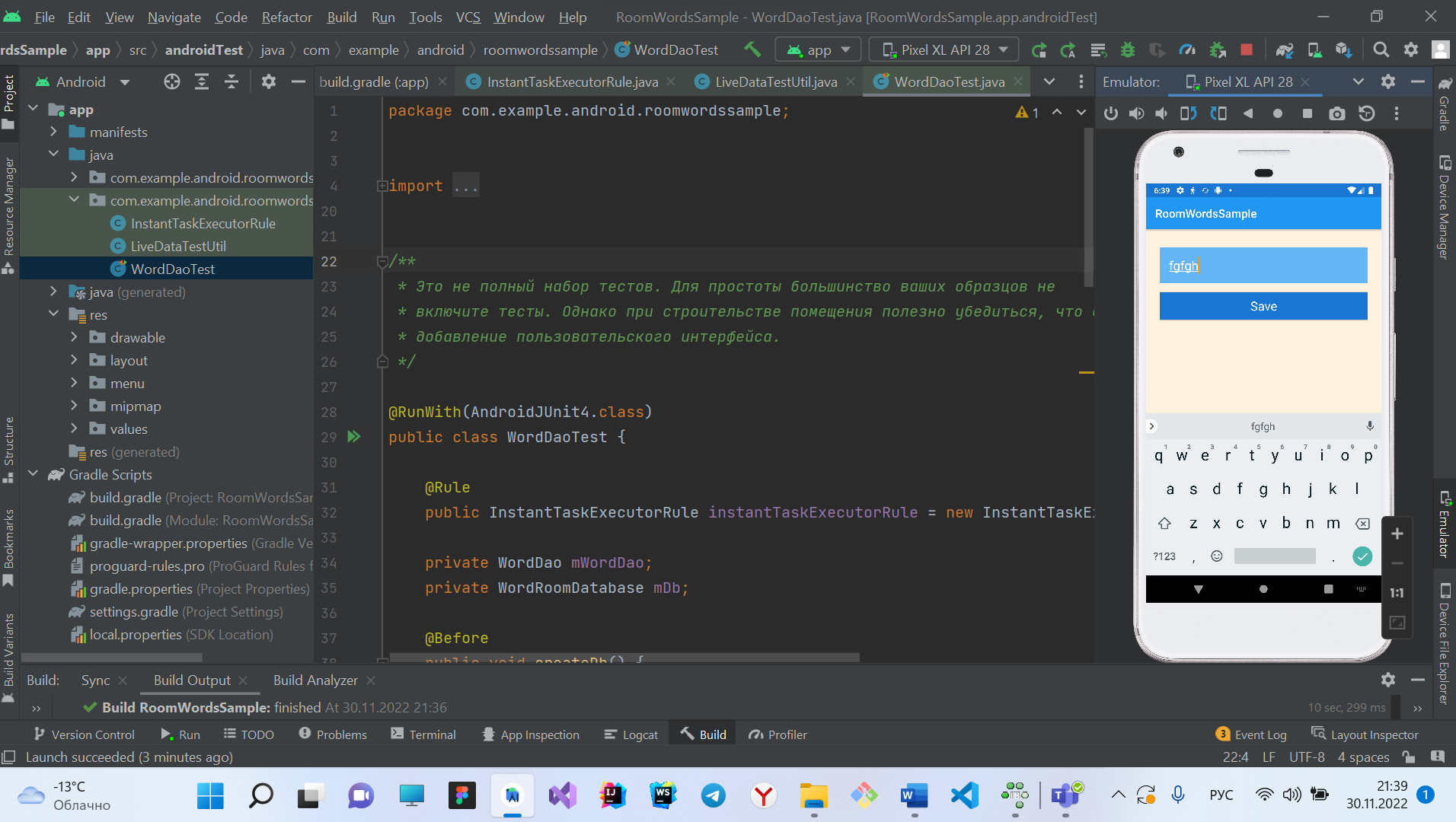


Рисунок 1 – Изначальное состояние программы

На рисунке 2 показано вторая страница программы, после нажатия на кнопку Fab.

Рисунок 2 – Вторая страница приложения

На рисунке 3 показана изначальная страница приложения, после того как мы добавили в список новое слово.

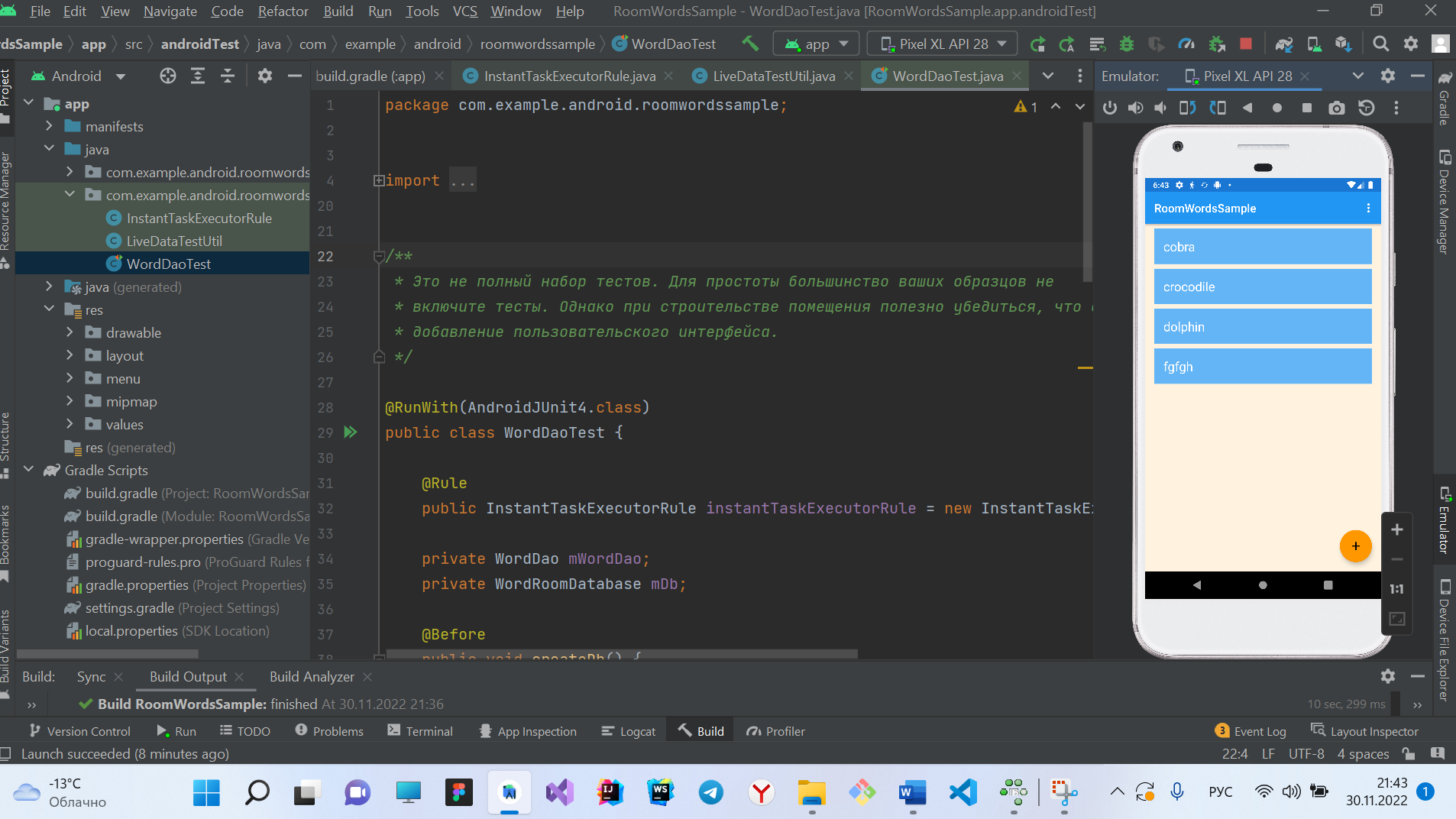


Рисунок 3 – Изменения на стартовой страницы программы

## **Анализ полученных результатов**

В результате выполнения данной лабораторной работы мной был разработан проект, в котором я научился, как спроектировать и создать приложение с использованием некоторых компонентов архитектуры Android. Вы будете использовать комнату, ViewModel, и LiveData.

.