1 . Вступление

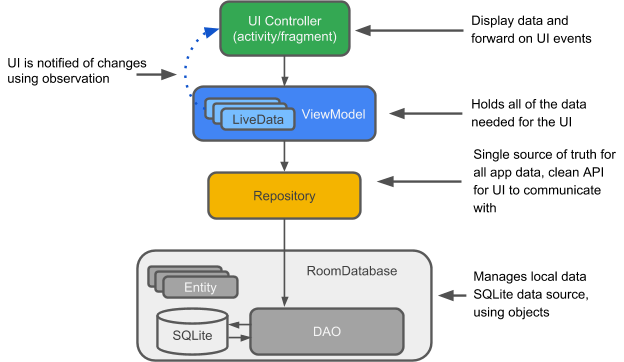
Цель [компонентов архитектуры](https://developer.android.com/topic/libraries/architecture/index.html) - дать рекомендации по архитектуре приложений, с библиотеками для таких общих задач, как управление жизненным циклом и постоянство данных.

Компоненты архитектуры помогают структурировать приложение таким образом, чтобы он был надежным, проверяемым и поддерживаемым с меньшим количеством шаблонов.

Компоненты архитектуры обеспечивают простой, гибкий и практичный подход, который избавляет вас от некоторых распространенных проблем, поэтому вы можете сосредоточиться на создании отличных впечатлений.

**Каковы рекомендуемые компоненты архитектуры?**

Чтобы ввести терминологию, вот краткое введение в компоненты архитектуры и то, как они работают вместе. Обратите внимание, что этот codelab фокусируется на подмножестве компонентов, а именно LiveData, ViewModel и Room. Каждый компонент объясняется больше, поскольку вы его используете. На этой диаграмме показана базовая форма этой архитектуры.



**Сущность.** При работе с компонентами архитектуры это аннотированный класс, который описывает таблицу базы данных.

**База данных SQLite:** на устройстве данные хранятся в базе данных SQLite. Для простоты дополнительные параметры хранилища, такие как веб-сервер, опущены. Библиотека сохранения состояний создает и поддерживает эту базу данных для вас.

**DAO:** объект доступа к данным. Отображение SQL-запросов к функциям. Раньше вы должны были четко определить их в своем [SQLiteOpenHelper](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html" \t "_blank)классе. Когда вы используете DAO, вы вызываете методы, и Room заботится обо всем остальном.

**База данных:** база данных поверх базы данных SQLite, которая заботится о мирских задачах, с которыми вы привыкли обрабатывать SQLiteOpenHelper. Держатель базы данных, который служит точкой доступа к базовой базе данных SQLite. База данных Room использует DAO для выпуска запросов к базе данных SQLite.

**Репозиторий:** класс, который вы создаете, например, используя WordRepositoryкласс. Вы используете репозиторий для управления несколькими источниками данных.

**ViewModel:** Предоставляет данные для пользовательского интерфейса. Действует как центр связи между репозиторием и пользовательским интерфейсом. Скрывает, откуда данные берутся из пользовательского интерфейса. В экземплярах ViewModel сохраняются изменения конфигурации.

**LiveData:** Класс держателя данных, который можно [наблюдать](https://en.wikipedia.org/wiki/Observer_pattern) . Всегда хранит / кэширует последнюю версию данных. Уведомляет своих наблюдателей, когда данные были изменены. LiveDataэто жизненный цикл. Компоненты пользовательского интерфейса просто наблюдают соответствующие данные и не останавливаются или не возобновляют наблюдение. LiveData автоматически управляет всем этим, так как он знает о соответствующих изменениях жизненного цикла во время наблюдения.

Ищете больше? Ознакомьтесь с полным [руководством по архитектуре приложений](https://developer.android.com/topic/libraries/architecture/guide.html) .

**Что вы будете строить**

Вы создадите приложение, использующее компоненты архитектуры Android, и реализует архитектуру из [руководства по архитектуре приложений](https://developer.android.com/topic/libraries/architecture/guide.html) для этих компонентов. Пример приложения хранит список слов в базе данных Room и отображает его в a RecyclerView. Приложение является голой кости, но достаточно сложным, что вы можете использовать его в качестве шаблона для разработки.

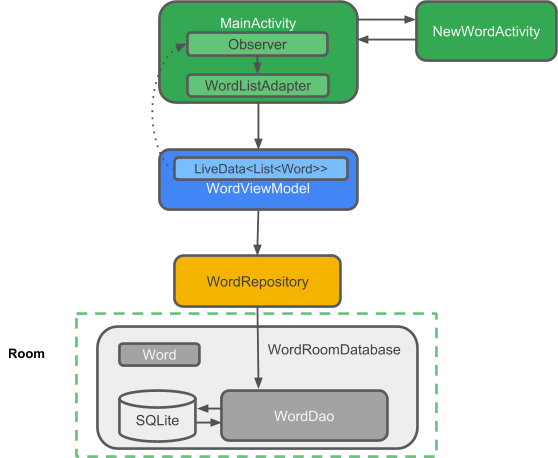
В этом кодебеле вы создаете приложение, которое выполняет следующие действия:

* Работает с базой данных для получения и сохранения данных и предварительно заполняет базу данных несколькими словами.
* Отображает все слова RecyclerViewв MainActivity.
* Открывает второе действие, когда пользователь нажимает кнопку +. Когда пользователь вводит слово, он добавляет слово в базу данных и список.

|  |  |  |
| --- | --- | --- |
| https://codelabs.developers.google.com/codelabs/android-room-with-a-view/img/2a518dec65b5b3dd.png | https://codelabs.developers.google.com/codelabs/android-room-with-a-view/img/ba2d72cea775f744.png | https://codelabs.developers.google.com/codelabs/android-room-with-a-view/img/ee5319926f0482eb.png |

Обзор **архитектуры** RoomWordSample

На следующей диаграмме показаны все части приложения. Каждый из закрывающих ящиков (кроме базы данных SQLite) представляет собой класс, который вы создадите.



**Что вы узнаете**

* Как создавать и создавать приложения с использованием библиотек компонентов Architecture Library и Lifecycles.

Существует множество шагов по использованию компонентов архитектуры и реализации рекомендуемой архитектуры. Самое главное - создать ментальную модель того, что происходит, и понять, как сочетаются части и как потоки данных. Когда вы работаете через этот codelab, не просто скопируйте и вставьте код, но попытайтесь начать строить это внутреннее понимание.

**Что вам понадобится**

* [Android Studio 3.0 или новее](https://developer.android.com/studio/index.html) и знания о том, как ее использовать. Убедитесь, что Android Studio обновлена, а также ваш SDK и Gradle. В противном случае вам придется подождать, пока все обновления не будут выполнены.
* Android-устройство или эмулятор.

Вы должны быть хорошо знакомы с языком программирования Java, объектно-ориентированными концепциями дизайна и Основами разработки Android. Особенно:

* RecyclerView и адаптеры
* База данных SQLite и язык запросов SQLite
* Threading и AsyncTask
* Это помогает познакомиться с архитектурными шаблонами программного обеспечения, которые отделяют данные от пользовательского интерфейса, например MVP или MVC. Этот codelab реализует архитектуру, определенную в [Руководстве по архитектуре приложений](https://developer.android.com/topic/libraries/architecture/guide.html) .

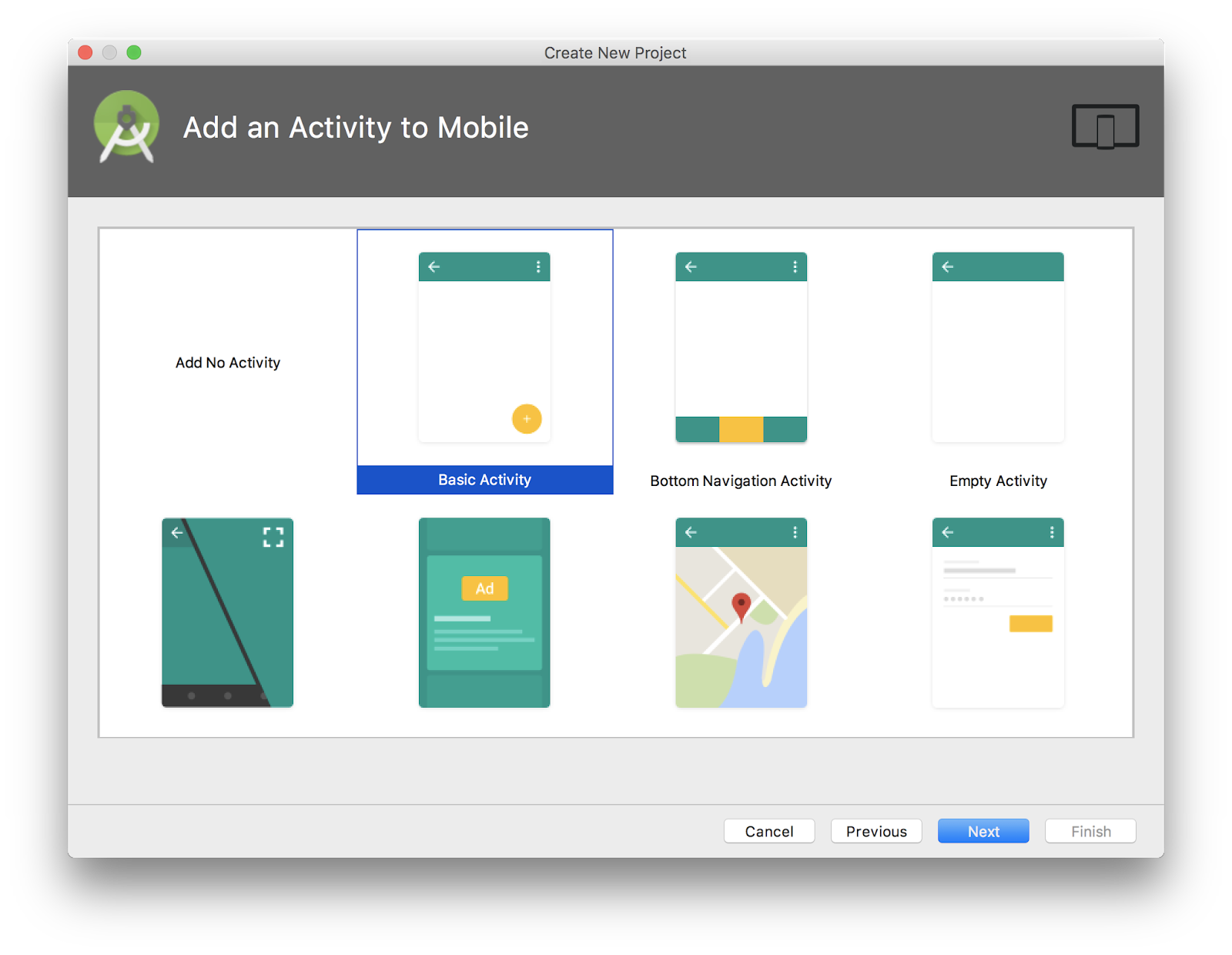
Этот codelab ориентирован на Android Architecture Components. Для простоты копирования и вставки вам предлагаются нестандартные концепции и код.

Этот codelab предоставляет весь код, необходимый для создания полного приложения.

2 . Создайте приложение

Откройте Android Studio и создайте приложение следующим образом:

* Вызовите приложение RoomWordSample
* Целевой SDK 26+
* Снимите отметку с обоих, **включите поддержку Kotlin** и **включите поддержку C ++** .
* Проверка **телефона и планшетного** формфактора только и минимальный SDK API 26.
* Выберите основную операцию.



3 . Обновить файлы градиента

Вы должны добавить библиотеки компонентов в свои файлы gradle.

Добавьте следующий код в ваш build.gradle( **модуль: приложение** ) файл, в конце блока зависимостей.

*// Room components*

implementation "android.arch.persistence.room:runtime:$rootProject.roomVersion"

annotationProcessor "android.arch.persistence.room:compiler:$rootProject.roomVersion"

androidTestImplementation "android.arch.persistence.room:testing:$rootProject.roomVersion"

*// Lifecycle components*

implementation "android.arch.lifecycle:extensions:$rootProject.archLifecycleVersion"

annotationProcessor "android.arch.lifecycle:compiler:$rootProject.archLifecycleVersion"

В файле build.gradle( **Project: RoomWordsSample** ) добавьте номера версий в конец файла, как указано в приведенном ниже коде.

Получите самые последние номера версий на странице « [Добавить компоненты на ваш проект»](https://developer.android.com/topic/libraries/architecture/adding-components.html) .

ext {

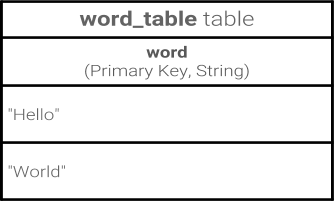
roomVersion = '1.1.1'

archLifecycleVersion = '1.1.1'

}

4 . Создание объекта

Данные для этого приложения - это слова, и каждое слово является [Entity](https://developer.android.com/training/data-storage/room/defining-data.html" \t "_blank) . Создайте класс, Wordкоторый описывает слово Entity. Вам нужен конструктор и метод «getter» для класса модели данных, потому что так Roomизвестно, чтобы создавать экземпляры ваших объектов.



Вот код:

public class Word {

private String mWord;

public Word(@NonNull String word) {this.mWord = word;}

public String getWord(){return this.mWord;}

}

Чтобы сделать Wordкласс значимым для базы данных Room, вам необходимо его аннотировать. Аннотации определяют, как каждая часть этого класса относится к записи в базе данных. Комната использует эту информацию для генерации кода.

* @Entity(tableName = **"word\_table"**)  
  Каждый @Entityкласс представляет собой объект в таблице. Аннотируйте объявление своего класса, чтобы указать, что это объект. Укажите имя таблицы, если вы хотите, чтобы она отличалась от имени класса.
* @PrimaryKey  
  Каждому объекту нужен первичный ключ. Чтобы все было просто, каждое слово действует как собственный первичный ключ.
* @NonNull  
  Обозначает, что значение возвращаемого параметра, поля или метода никогда не может быть нулевым.
* @ColumnInfo(name = **"word"**)  
  Укажите имя столбца в таблице, если вы хотите, чтобы он отличался от имени переменной-члена.
* Каждое поле, которое хранится в базе данных, должно быть общедоступным или иметь метод «getter». Этот пример предоставляет getWord()метод.

Вы можете найти полный список аннотаций в [сводной ссылке пакета номера](https://developer.android.com/reference/android/arch/persistence/room/package-summary.html) .

Обновите Wordкласс с помощью аннотаций, как показано в этом коде. Если вы введете аннотации, Android Studio будет автоматически импортировать.

@Entity(tableName = "word\_table")

public class Word {

@PrimaryKey

@NonNull

@ColumnInfo(name = "word")

private String mWord;

public Word(String word) {this.mWord = word;}

public String getWord(){return this.mWord;}

}

**Когда вы копируете код вставки, если вы получаете ошибки для аннотаций,** вам может потребоваться вручную импортировать аннотации.

import android.arch.persistence.room.ColumnInfo;

import android.arch.persistence.room.Entity;

import android.arch.persistence.room.PrimaryKey;

import android.support.annotation.NonNull;

**Совет.** Вы можете [автогенерировать](https://developer.android.com/reference/android/arch/persistence/room/PrimaryKey.html" \t "_blank) уникальные ключи, аннотируя первичный ключ следующим образом:

@Entity(tableName = "word\_table")  
public class Word {  
  
@PrimaryKey(autoGenerate = true)  
private int id;  
  
@NonNull  
private String word;  
//..other fields, getters, setters  
}

См. Раздел « [Определение данных с использованием объектов« Комната »](https://developer.android.com/training/data-storage/room/defining-data.html) .

5 . Создание DAO

**Что такое DAO?**

В [DAO](https://developer.android.com/training/data-storage/room/accessing-data.html) (объект доступа к данным) вы указываете SQL-запросы и связываете их с вызовами методов. Компилятор проверяет SQL и генерирует запросы из удобных аннотаций для обычных запросов, например @Insert.

DAO должен быть интерфейсом или абстрактным классом.

По умолчанию все запросы должны выполняться в отдельном потоке.

Комната использует DAO для создания чистого API для вашего кода.

**Внедрение DAO**

DAO для этого codelab является базовым и предоставляет запросы для получения всех слов, вставки слова и удаления всех слов.

1. Создайте новый интерфейс и вызовите его WordDao.
2. Аннотируйте класс, @Daoчтобы определить его как класс DAO для комнаты.
3. Объявите способ вставить одно слово: void insert(Word word);
4. Аннотировать метод с помощью @Insert. Вам не нужно предоставлять SQL! (Есть также @Deleteи @Updateаннотации для удаления и обновления строки, но вы не используете их в этом приложении.)
5. Объявить метод , чтобы удалить все слова: void deleteAll();.
6. Нет никакой удобной аннотации для удаления нескольких объектов, поэтому аннотируйте метод с общим @Query.
7. Предоставьте SQL-запрос в качестве строкового параметра @Query. Используйте @Queryдля чтения и сложных запросов и обеспечения SQL.  
   **@Query("DELETE FROM word\_table")**
8. Создайте метод , чтобы получить все слова: getAllWords();.   
   Есть метод возвращает Listиз Words.  
   **List<Word> getAllWords();**
9. Аннотировать метод с помощью SQL-запроса:   
   @Query(**"SELECT \* from word\_table ORDER BY word ASC"**)

Вот полный код:

@Dao

public interface WordDao {

@Insert

void insert(Word word);

@Query("DELETE FROM word\_table")

void deleteAll();

@Query("SELECT \* from word\_table ORDER BY word ASC")

List<Word> getAllWords();

}

**Совет.** Для этого приложения упорядочение слов не является строго необходимым. Однако, по умолчанию, заказ не гарантируется, и заказ делает тестирование простым.

**Совет.** При вставке данных вы можете предоставить стратегию конфликта.

В этом кодебеле вам не нужна стратегия конфликта, потому что это ваш первичный ключ, а поведение по умолчанию SQL - ABORTэто то, что вы не можете вставить в базу два элемента с одним и тем же основным ключом.

Если таблица имеет более одного столбца, вы можете использовать

@Insert(onConflict = OnConflictStrategy.REPLACE)

для замены строки.

Узнайте больше о [DAO Room](https://developer.android.com/training/data-storage/room/accessing-data.html) .

## 6 . Класс LiveData

При изменении данных вы обычно хотите предпринять некоторые действия, например, отобразить обновленные данные в пользовательском интерфейсе. Это означает, что вы должны наблюдать данные, чтобы при их изменении вы могли реагировать. В зависимости от того, как хранятся данные, это может быть сложно. Наблюдение за изменениями данных по нескольким компонентам вашего приложения может создавать явные жесткие пути зависимостей между компонентами. Это затрудняет, среди прочего, тестирование и отладку.

[LiveData](https://developer.android.com/topic/libraries/architecture/livedata.html), класс [библиотеки жизненного цикла](https://developer.android.com/topic/libraries/architecture/lifecycle.html) для наблюдения данных, решает эту проблему. Используйте возвращаемое значение типа [LiveData](https://developer.android.com/reference/android/arch/lifecycle/LiveData.html" \t "_blank)в описании вашего метода, и Room генерирует весь необходимый код для обновления, LiveDataкогда база данных обновляется.

Если вы используете LiveDataнезависимо от комнаты, вам необходимо управлять обновлением данных. LiveDataне имеет общедоступных методов обновления хранимых данных.

Если вы, разработчик, хотите обновить сохраненные данные, вы должны использовать[MutableLiveData](https://developer.android.com/reference/android/arch/lifecycle/MutableLiveData.html)вместо него LiveData. MutableLiveDataКласс имеет два открытых методов , которые позволяют установить значение LiveDataобъекта, [setValue(T)](https://developer.android.com/reference/android/arch/lifecycle/MutableLiveData.html#setValue(T))и [postValue(T)](https://developer.android.com/reference/android/arch/lifecycle/MutableLiveData.html#postValue(T)). Обычно, MutableLiveDataиспользуется в [ViewModel](https://developer.android.com/reference/android/arch/lifecycle/ViewModel.html" \t "_blank), а затем ViewModelединственное выставляет неизменные LiveDataобъекты наблюдателям.

В WordDao, измените getAllWords()подпись метода так, чтобы возвращаемое было List<Word>завернуто LiveData.

@Query("SELECT \* from word\_table ORDER BY word ASC")

LiveData<List<Word>> getAllWords();

Позже в этом кодебеле вы создаете Observerданные в onCreate()методе MainActivityи переопределяете метод наблюдателя onChanged(). При LiveDataизменениях наблюдатель уведомляется и onChanged()выполняется. Затем вы обновите кешированные данные в адаптере, и адаптер обновит то, что видит пользователь.

См. [LiveData](https://developer.android.com/topic/libraries/architecture/livedata.html" \t "_blank)Документацию, чтобы узнать больше о других способах использования LiveDataили просмотреть эти [компоненты Architecture: LifeData и Lifecycle](https://www.youtube.com/watch?v=jCw5ib0r9wg) .

7 . Добавить базу данных о номерах

**Что такое база данных Room?**

Комната - это слой базы данных поверх базы данных SQLite. Комната заботится о мирских задачах, с которыми вы привыкли обращаться [SQLiteOpenHelper](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html" \t "_blank).

* Комната использует DAO для выпуска запросов в свою базу данных.
* По умолчанию, чтобы избежать плохой производительности пользовательского интерфейса, Room не позволяет вам выполнять запросы базы данных в основном потоке. [LiveData](https://developer.android.com/" \t "_blank)применяет это правило, автоматически выполняя запрос асинхронно по фоновому потоку, когда это необходимо.
* В комнате предусмотрены проверки выполнения SQLite.
* Класс вашей комнаты должен быть абстрактным и распространяться RoomDatabase.
* Обычно вам нужен только один экземпляр базы данных Room для всего приложения.

**Внедрение базы данных Room**

1. Создайте public abstractкласс, который расширяет RoomDatabase и вызывает его WordRoomDatabase.  
   **public abstract class**WordRoomDatabase **extends**RoomDatabase {}
2. Аннотируйте класс как базу данных Room, объявите объекты, которые принадлежат базе данных, и установите номер версии. Листинг сущностей будет создавать таблицы в базе данных.  
   @Database(entities = {Word.**class**}, version = 1)
3. Определите DAO, которые работают с базой данных. Предоставьте абстрактный метод «getter» для каждого @Dao.  
   **public abstract**WordDao wordDao();

Вот код:

@Database(entities = {Word.class}, version = 1)

public abstract class WordRoomDatabase extends RoomDatabase {

public abstract WordDao wordDao();

}

1. Сделать WordRoomDatabaseс [одноплодной](https://en.wikipedia.org/wiki/Singleton_pattern) , чтобы предотвратить наличие нескольких экземпляров базы данных открыт одновременно.

Вот код:

private static volatile WordRoomDatabase INSTANCE;

static WordRoomDatabase getDatabase(final Context context) {

if (INSTANCE == null) {

synchronized (WordRoomDatabase.class) {

if (INSTANCE == null) {

*// Create database here*

}

}

}

return INSTANCE;

}

1. Добавьте код для получения базы данных. Этот код использует конструктор базы данных Room для создания [RoomDatabase](https://developer.android.com/reference/android/arch/persistence/room/RoomDatabase.html" \t "_blank)объекта в контексте приложения из WordRoomDatabaseкласса и называет его "word\_database".

*// Create database here*

INSTANCE = Room.databaseBuilder(context.getApplicationContext(),

WordRoomDatabase.class, "word\_database")

.build();

Вот полный код для класса:

@Database(entities = {Word.class}, version = 1)

public abstract class WordRoomDatabase extends RoomDatabase {

public abstract WordDao wordDao();

private static volatile WordRoomDatabase INSTANCE;

static WordRoomDatabase getDatabase(final Context context) {

if (INSTANCE == null) {

synchronized (WordRoomDatabase.class) {

if (INSTANCE == null) {

INSTANCE = Room.databaseBuilder(context.getApplicationContext(),

WordRoomDatabase.class, "word\_database")

.build();

}

}

}

return INSTANCE;

}

}

В этом кодебеле вы не обновляете сущности и номера версий. При изменении схемы базы данных вам необходимо обновить номер версии и определить, как обрабатывать миграцию.

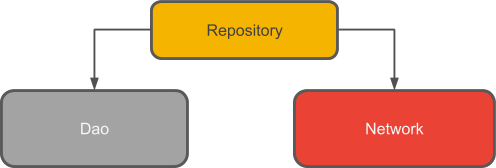
Для примера, уничтожение и воссоздание базы данных - прекрасная стратегия миграции. Для реального приложения вы должны реализовать стратегию миграции. См. Раздел [Понимание миграции с помощью комнаты](https://medium.com/google-developers/understanding-migrations-with-room-f01e04b07929) .

В Android Studio, если вы получаете ошибки при вставке кода или во время процесса сборки, выберите « **Сборка»> «Очистить проект»** . Затем выберите **Build> Rebuild Project** , а затем снова создайте. Если вы используете предоставленный код, не должно быть ошибок, когда вам будет предложено создать приложение.

8 . Создать репозиторий

Что такое репозиторий?

Репозиторий - это класс, который абстрагирует доступ к нескольким источникам данных. Репозиторий не является частью библиотек компонентов архитектуры, но является рекомендуемой передовой практикой для разделения кода и архитектуры. RepositoryКласс обрабатывает данные операции. Он предоставляет чистый API для остальной части приложения для данных приложения.



**Зачем использовать репозиторий?**

Репозиторий управляет потоками запросов и позволяет использовать несколько бэкэндов. В наиболее распространенном примере репозиторий реализует логику решения о том, следует ли извлекать данные из сети или использовать результаты, кэшированные в локальной базе данных.

**Реализация хранилища**

1. Создайте открытый класс WordRepository.
2. Добавьте переменные-члены для DAO и список слов.

private WordDao mWordDao;

private LiveData<List<Word>> mAllWords;

1. Добавьте конструктор, который получает дескриптор базы данных и инициализирует переменные-члены.

WordRepository(Application application) {

WordRoomDatabase db = WordRoomDatabase.getDatabase(application);

mWordDao = db.wordDao();

mAllWords = mWordDao.getAllWords();

}

1. Добавьте обертку для getAllWords(). Комната выполняет все запросы в отдельном потоке. Наблюдаемый LiveDataуведомит наблюдателя, когда данные изменились.

LiveData<List<Word>> getAllWords() {

return mAllWords;

}

1. Добавьте обертку для insert()метода. Вы должны вызывать это в потоке, отличном от UI, или ваше приложение выйдет из строя. Room гарантирует, что вы не выполняете длительных операций в основном потоке, блокируя пользовательский интерфейс.

public void insert (Word word) {

new insertAsyncTask(mWordDao).execute(word);

}

1. В этом нет ничего волшебного AsyncTask, поэтому здесь вам нужно скопировать.

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;

}

}

Вот полный код:

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);

}

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;

}

}

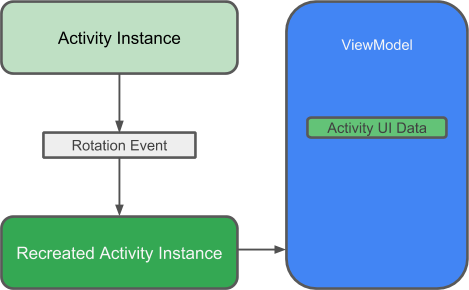
}

Для этого простого примера репозиторий не делает многого. См. [BasicSample](https://github.com/googlesamples/android-architecture-components/tree/master/BasicSample" \t "_blank) для более сложной реализации.

9 . Создайте ViewModel

**Что такое ViewModel?**

ViewModel«S роль заключается в предоставлении данных в пользовательский интерфейс и выжить изменения конфигурации. A ViewModelдействует как центр связи между репозиторием и пользовательским интерфейсом. Вы также можете использовать ViewModelдля обмена данными между фрагментами. ViewModel является частью [библиотеки жизненного цикла](https://developer.android.com/topic/libraries/architecture/lifecycle.html) .



Для вводного руководства по этой теме см [ViewModel](https://developer.android.com/topic/libraries/architecture/viewmodel.html" \t "_blank).

**Зачем использовать ViewModel?**

A ViewModelхранит данные пользовательского интерфейса вашего приложения в сознательном жизненном цикле, который сохраняет изменения конфигурации. Разделение данных пользовательского интерфейса вашего приложения с вашего Activityи Fragmentклассов позволяет лучше следовать принципу единой ответственности: ваши действия и фрагменты отвечают за вывод данных на экран, в то время как вы ViewModelможете позаботиться о хранении и обработке всех данных, необходимых для пользовательского интерфейса.

В этом случае ViewModelиспользуйте LiveDataпеременные данные, которые пользователь будет использовать или отображать. Использование LiveDataимеет несколько преимуществ:

* Вы можете поместить наблюдателя в данные (вместо опроса для изменений) и обновить   
  интерфейс только после фактического изменения данных.
* Репозиторий и пользовательский интерфейс полностью разделены ViewModel. Нет вызовов из базы данных ViewModel, что делает код более проверяемым.

**Внедрение ViewModel**

1. Создайте класс, WordViewModelкоторый расширяет AndroidViewModel.

public class WordViewModel extends AndroidViewModel {}

1. Добавьте переменную частного члена для хранения ссылки на репозиторий.

private WordRepository mRepository;

1. Добавьте приватную LiveDataпеременную-член для кеширования списка слов.

private LiveData<List<Word>> mAllWords;

1. Добавьте конструктор, который получает ссылку на репозиторий и получает список слов из репозитория.

public WordViewModel (Application application) {

super(application);

mRepository = new WordRepository(application);

mAllWords = mRepository.getAllWords();

}

1. Добавьте метод «getter» для всех слов. Это полностью скрывает реализацию от пользовательского интерфейса.

LiveData<List<Word>> getAllWords() { return mAllWords; }

1. Создайте insert()метод оболочки, который вызывает метод репозитория insert(). Таким образом, реализация insert()полностью скрыта от пользовательского интерфейса.

public void insert(Word word) { mRepository.insert(word); }

Вот полный код для WordViewModel:

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); }

}

**Предупреждение:** никогда не передавать контекст в ViewModelэкземпляры. Не храните Activity, Fragmentили Viewэкземпляры или их Contextв ViewModel.

Например, Activityможет быть уничтожено и создано много раз в течение жизненного цикла, ViewModelкогда устройство вращается. Если вы сохраните ссылку на Activityв ViewModel, вы в конечном итоге со ссылками , которые указывают на разрушенное Activity. Это утечка памяти.

Если вам нужен контекст приложения, используйте AndroidViewModel, как показано в этом кодедебе.

**Важно:**ViewModel это не замена onSaveInstanceState()метода, потому что ViewModelон не выдерживает остановки процесса. Подробнее [здесь](https://medium.com/google-developers/viewmodels-persistence-onsaveinstancestate-restoring-ui-state-and-loaders-fc7cc4a6c090) .

Чтобы узнать больше, просмотрите видеокомпоненты [Architecture Components: ViewModel](https://www.youtube.com/watch?v=c9-057jC1ZA" \t "_blank) .

10 . Добавить XML-макет

Затем вам нужно добавить XML-макет для списка и элементов.

Этот codelab предполагает, что вы знакомы с созданием макетов в XML, поэтому мы просто предоставляем вам код.

Добавьте стиль для элементов списка в values/styles.xml:

*<!-- The default font for RecyclerView items is too small.*

*The margin is a simple delimiter between the words. -->*

<style name="word\_title">

<item name="android:layout\_width">match\_parent</item>

<item name="android:layout\_height">26dp</item>

<item name="android:textSize">24sp</item>

<item name="android:textStyle">bold</item>

<item name="android:layout\_marginBottom">6dp</item>

<item name="android:paddingLeft">8dp</item>

</style>

Добавить layout/recyclerview\_item.xmlмакет:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical" android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<TextView

android:id="@+id/textView"

style="@style/word\_title"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:background="@android:color/holo\_orange\_light" />

</LinearLayout>

В layout/content\_main.xml, замените : TextViewa RecyclerView:

<android.support.v7.widget.RecyclerView

android:id="@+id/recyclerview"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:background="@android:color/darker\_gray"

tools:listitem="@layout/recyclerview\_item" />

Ваша плавающая кнопка действия (FAB) должна соответствовать доступному действию. В layout/activity\_main.xmlфайле, дают FloatingActionButtonна +значок символа:

1. В layout/activity\_main.xmlфайле выберите « **Файл»> «Создать»> «Вектор»** .
2. Выберите **значок материала** .
3. Нажмите значок робота Android в поле **Значок:** выберите +ресурс («добавить»).
4. Измените код файла макета следующим образом.

android:src="@drawable/ic\_add\_black\_24dp"

## 11 . Добавить RecyclerView

Вы собираетесь отображать данные в a RecyclerView, что немного лучше, чем просто бросать данные в a TextView. Это codelab предполагает , что вы знаете , как [RecyclerView](https://www.google.com/url?q=https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html&sa=U&ved=0ahUKEwj0sqGO0s3XAhVBwVQKHVakBFIQFggEMAA&client=internal-uds-cse&cx=000521750095050289010:zpcpi1ea4s8&usg=AOvVaw0Ze9aHV4aNxBt4fPVYvjKD), [RecyclerView.LayoutManager](https://developer.android.com/reference/android/support/v7/widget/RecyclerView.LayoutManager.html), [RecyclerView.ViewHolder](https://www.google.com/url?q=https://developer.android.com/reference/android/support/v7/widget/RecyclerView.ViewHolder.html&sa=U&ved=0ahUKEwjK4Jy00s3XAhUIxlQKHX9jAhkQFggEMAA&client=internal-uds-cse&cx=000521750095050289010:zpcpi1ea4s8&usg=AOvVaw3j_sAR3AnGb855QSn82WEf)и [RecyclerView.Adapter](https://www.google.com/url?q=https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html&sa=U&ved=0ahUKEwiji9DU0s3XAhWCj1QKHSzjDscQFggEMAA&client=internal-uds-cse&cx=000521750095050289010:zpcpi1ea4s8&usg=AOvVaw3CkQoDAgrAMidLMxDz-TC6)работу.

Обратите внимание, что mWordsпеременная в адаптере кэширует данные. В следующей задаче вы добавляете код, который автоматически обновляет данные.

Также обратите внимание, что этот getItemCount()метод должен изящно учитывать, что данные еще не готовы и mWordsостаются null. В более сложном приложении вы можете отображать данные заполнителя или что-то еще, что было бы значимым для пользователя.

Добавьте класс, WordListAdapterкоторый расширяется RecyclerView.Adapter. Вот код.

public class WordListAdapter extends RecyclerView.Adapter<WordListAdapter.WordViewHolder> {

class WordViewHolder extends RecyclerView.ViewHolder {

private final TextView wordItemView;

private WordViewHolder(View itemView) {

super(itemView);

wordItemView = itemView.findViewById(R.id.textView);

}

}

private final LayoutInflater mInflater;

private List<Word> mWords; *// Cached copy of words*

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 {

*// Covers the case of data not being ready yet.*

holder.wordItemView.setText("No Word");

}

}

void setWords(List<Word> words){

mWords = words;

notifyDataSetChanged();

}

*// getItemCount() is called many times, and when it is first called,*

*// mWords has not been updated (means initially, it's null, and we can't return null).*

@Override

public int getItemCount() {

if (mWords != null)

return mWords.size();

else return 0;

}

}

Добавьте RecyclerViewв onCreate()методе MainActivity.

В onCreate()методе:

RecyclerView recyclerView = findViewById(R.id.recyclerview);

final WordListAdapter adapter = new WordListAdapter(this);

recyclerView.setAdapter(adapter);

recyclerView.setLayoutManager(new LinearLayoutManager(this));

Запустите приложение, чтобы убедиться, что все работает. Нет элементов, потому что вы еще не подключили данные, поэтому приложение должно отображать серый фон без элементов списка.

## https://codelabs.developers.google.com/codelabs/android-room-with-a-view/img/193d01725acbe6cc.png

## 12 . Заполнение базы данных

В базе данных нет данных. Вы добавите данные двумя способами: добавьте некоторые данные при открытии базы данных и добавьте их Activityдля добавления слов.

Чтобы удалить весь контент и повторно заполнить базу данных всякий раз, когда приложение запущено, вы создаете [RoomDatabase.Callback](https://developer.android.com/reference/android/arch/persistence/room/RoomDatabase.Callback.html" \t "_blank)и переопределяете onOpen(). Поскольку вы не можете выполнять операции с базой данных в потоке пользовательского интерфейса, onOpen()создает и выполняет AsyncTaskдобавление контента в базу данных.

Вот код для создания обратного вызова в WordRoomDatabaseклассе:

private static RoomDatabase.Callback sRoomDatabaseCallback =

new RoomDatabase.Callback(){

@Override

public void onOpen (@NonNull SupportSQLiteDatabase db){

super.onOpen(db);

new PopulateDbAsync(INSTANCE).execute();

}

};

Вот код для AsyncTask, который удаляет содержимое базы данных, затем заполняет его двумя словами «Hello» и «World». Не стесняйтесь добавлять больше слов!

private static class PopulateDbAsync extends AsyncTask<Void, Void, Void> {

private final WordDao mDao;

PopulateDbAsync(WordRoomDatabase db) {

mDao = db.wordDao();

}

@Override

protected Void doInBackground(final Void... params) {

mDao.deleteAll();

Word word = new Word("Hello");

mDao.insert(word);

word = new Word("World");

mDao.insert(word);

return null;

}

}

Наконец, добавьте обратный вызов к последовательности сборки базы данных непосредственно перед вызовом .build().

.addCallback(sRoomDatabaseCallback)

## 13 . Добавить NewWordActivity

Добавьте эти строковые ресурсы в values/strings.xml:

<string name="hint\_word">Word...</string>

<string name="button\_save">Save</string>

<string name="empty\_not\_saved">Word not saved because it is empty.</string>

Добавьте этот цветной ресурс в value/colors.xml:

<color name="buttonLabel">#d3d3d3</color>

Добавьте эти ресурсы измерения в values/dimens.xml:

<dimen name="small\_padding">6dp</dimen>

<dimen name="big\_padding">16dp</dimen>

Создайте activity\_new\_word.xmlфайл в папке макета:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical" android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<EditText

android:id="@+id/edit\_word"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:fontFamily="sans-serif-light"

android:hint="@string/hint\_word"

android:inputType="textAutoComplete"

android:padding="@dimen/small\_padding"

android:layout\_marginBottom="@dimen/big\_padding"

android:layout\_marginTop="@dimen/big\_padding"

android:textSize="18sp" />

<Button

android:id="@+id/button\_save"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:background="@color/colorPrimary"

android:text="@string/button\_save"

android:textColor="@color/buttonLabel" />

</LinearLayout>

Для создания нового действия используйте шаблон Empty Activity NewWordActivity. Убедитесь, что активность добавлена ​​в манифест Android!  
<**activity android:name=".NewWordActivity"**></**activity**>

Вот код действия:

public class NewWordActivity extends AppCompatActivity {

public static final String EXTRA\_REPLY = "com.example.android.wordlistsql.REPLY";

private EditText mEditWordView;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_new\_word);

mEditWordView = findViewById(R.id.edit\_word);

final Button button = findViewById(R.id.button\_save);

button.setOnClickListener(new View.OnClickListener() {

public void onClick(View view) {

Intent replyIntent = new Intent();

if (TextUtils.isEmpty(mEditWordView.getText())) {

setResult(RESULT\_CANCELED, replyIntent);

} else {

String word = mEditWordView.getText().toString();

replyIntent.putExtra(EXTRA\_REPLY, word);

setResult(RESULT\_OK, replyIntent);

}

finish();

}

});

}

}

## 14 . Подключение к данным

Последним шагом является подключение пользовательского интерфейса к базе данных путем сохранения новых слов, которые вводит пользователь, и отображения текущего содержимого базы данных слова в RecyclerView.

Чтобы отобразить текущее содержимое базы данных, добавьте наблюдателя, который наблюдает LiveDataза ним ViewModel. Всякий раз, когда данные изменяются, onChanged()вызывается обратный вызов, который вызывает метод адаптера setWord()для обновления кэшированных данных адаптера и обновления отображаемого списка.

В MainActivity, создайте переменную-член для ViewModel:

private WordViewModel mWordViewModel;

Используйте [ViewModelProviders](https://developer.android.com/reference/android/arch/lifecycle/ViewModelProviders.html" \t "_blank)для связи ViewModelс вашим контроллером пользовательского интерфейса. Когда ваше приложение начнется сначала, оно ViewModelProvidersсоздаст ViewModel. Когда действие уничтожается, например, путем изменения конфигурации, ViewModelсохраняется. Когда активность будет воссоздана, ViewModelProvidersвозвратите существующее ViewModel. См [ViewModel](https://developer.android.com/topic/libraries/architecture/viewmodel.html" \t "_blank).

В onCreate(), получите ViewModelот ViewModelProvider.

mWordViewModel = ViewModelProviders.of(this).get(WordViewModel.class);

Также в onCreate(), добавьте наблюдателя для LiveDataвозвращенного getAllWords().   
В onChanged()метод срабатывает , если наблюдаемые изменения данных и активность находится на переднем плане.

mWordViewModel.getAllWords().observe(this, new Observer<List<Word>>() {

@Override

public void onChanged(@Nullable final List<Word> words) {

*// Update the cached copy of the words in the adapter.*

adapter.setWords(words);

}

});

В MainActivity, добавьте onActivityResult()код для NewWordActivity.

Если операция возвращается с RESULT\_OK, вставьте возвращаемое слово в базу данных, вызвав insert()метод WordViewModel.

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));

mWordViewModel.insert(word);

} else {

Toast.makeText(

getApplicationContext(),

R.string.empty\_not\_saved,

Toast.LENGTH\_LONG).show();

}

}

Определите отсутствующий код запроса:

public static final int NEW\_WORD\_ACTIVITY\_REQUEST\_CODE = 1;

В MainActivity,начале, NewWordActivityкогда пользователь удаляет FAB. Замените код в onClick()обработчике кликов FAB с помощью этого кода:

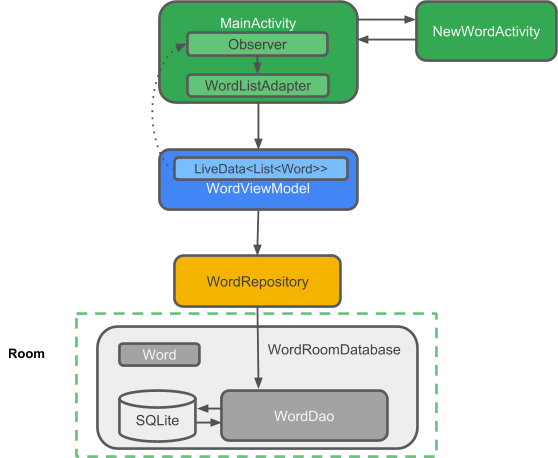
Intent intent = new Intent(MainActivity.this, NewWordActivity.class);

startActivityForResult(intent, NEW\_WORD\_ACTIVITY\_REQUEST\_CODE);

ЗАПУСТИТЕ ВАШУ ПРИЛОЖЕНИЕ !!!

Когда вы добавляете слово в базу данных NewWordActivity, пользовательский интерфейс будет автоматически обновляться.

## 15 . Резюме



Теперь, когда у вас есть рабочее приложение, давайте вспомним, что вы создали. Вот структура приложения снова, с самого начала.

У вас есть приложение , которое отображает слова в списке ( MainActivity, RecyclerView, WordListAdapter).

Вы можете добавить слова в список ( NewWordActivity).

Слово является экземпляром Wordкласса сущности.

Слова кешируются в RecyclerViewAdapterвиде Listслова ( mWords). Этот список слов автоматически обновляется и обновляется при изменении слов в базе данных.

## Поток данных для автоматического обновления пользовательского интерфейса (реактивный интерфейс)

Автоматическое обновление возможно, потому что мы используем LiveData. В MainActivityнем есть Observerтот, который наблюдает слова LiveData из базы данных и уведомляется, когда они меняются. Когда происходит изменение, метод наблюдателя onChange()выполняется и обновляется mWordsв WordListAdapter.

Данные можно наблюдать, потому что это так LiveData. И наблюдается то, LiveData<List<Word>>что возвращается методом WordViewModelобъекта getAllWords().

WordViewModelСкрывает все о бэкэнде из UI layer. Он предоставляет методы для доступа к слою данных и возвращает его, LiveDataчтобы MainActivityможно было установить отношения наблюдателя. Viewsи Activities(и Fragments) взаимодействуют только с данными через ViewModel. Таким образом, неважно, откуда берутся данные.

В этом случае данные поступают из a Repository. ViewModelНе нужно знать , что это Repository взаимодействует с. Он просто должен знать, как взаимодействовать с ним Repository, то есть с помощью методов, открытых им Repository.

Репозиторий управляет одним или несколькими источниками данных. В WordListSampleприложении этот бэкэнд является базой данных Room. Комната представляет собой оболочку и реализует базу данных SQLite. Комната делает для вас много работы, что вам приходилось делать самому. Например, Room делает все, что вы делали с SQLiteOpenHelperклассом.

Метод DAO сопоставляет запрос к запросам базы данных, поэтому, когда репозиторий вызывает метод, например getAllWords()Room, может выполняться **.SELECT \* from word\_table ORDER BY word ASC**

Поскольку результат, возвращаемый из запроса, наблюдается LiveData, каждый раз, когда данные в Комнате меняются, выполняется Observerинтерфейс, onChanged()и пользовательский интерфейс обновляется.

16 . Код решения

**[Необязательно] Загрузите код решения**

Нажмите следующую ссылку, чтобы загрузить код решения для этой codelab:

[Скачать исходный код](https://github.com/googlecodelabs/android-room-with-a-view" \t "_blank)

Распакуйте загруженный zip-файл. Это распакует корневую папку, android-room-with-a-view-masterсодержащую полное приложение. Приложение очень модульное, поэтому для каждого шага нет отдельной папки.