MINISTRY OF DEVELOPMENT OF INFORMATION TECHNOLOGIES AND COMMUNICATIONS OF THE REPUBLIC OF UZBEKISTAN

TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES NAMED AFTER MUHAMMAD AL-KHWARIZMI

Head of the Department

«___» _____ 2018

GRADUATION WORK

Theme: «Development of mobile applications “Learn by heart” for English learners».

Graduate _____ Amanova M.A.

(signature)

Supervisor _____ Latipova N.

(signature)

Life Safety consultant _____ Axmedova G.N.

(signature)

Reviewer _____

(signature)

TASHKENT – 2018

MINISTRY OF DEVELOPMENT OF INFORMATION TECHNOLOGIES AND COMMUNICATIONS OF THE REPUBLIC OF UZBEKISTAN

TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES NAMED AFTER MUHAMMAD AL-KHWARIZMI

Faculty: Software engineering Department: System and application programming
Specialty: 5330600 –«Software engineering»

APPROVED BY:
Head of the Department _____
« ___ »_____ 2018

ASSIGNMENT

On graduation work of student: Amanova Madina Azizovna

1. Theme of work: Development of mobile applications “Learn by heart” for English learners.
2. Theme approved by University order № 13130.125 from 21.11.2018y.
3. Deadline for completing graduation work: 22.05.2018y.
4. Baseline data for graduation work: Internet resources, theme manuals.
5. Contents of the explanatory note (list of issues to be developed): The analysis of subject area, analysis of the requirements for the development of the system, design of the database structure.
6. List of graphic material: tables, diagrams, user interfaces, presentation.
7. Date of issue of the assignment: 06.12.2017y.

Supervisor: _____
(signature)

Task was accepted: _____
(signature)

8. Consultants for selected sections of the graduation work:

Section	Full name of supervisor	Signature and date	
		Task issued	Task received
Main part	Latipova N.	06.12.2017y.	06.12.2017y.
Life safety	Axmedova	19.05.2018y.	19.05.2018y.

9. Progress chart:

№	Sections of Graduation Work	Deadline	Mark of supervisor
1	Introduction	10.02.2018	
2	Necessity of english learning applications	25.03.2018	
3	Current techniques to Android apps development	10.04.2018	
4	Development of “learn by heart” mobile application	18.05.2018	
5	Life Safety	23.05.2018	
6	Conclusion	30.05.2018	

Graduate: _____«___» _____ 2018.
(signature)

Supervisor : _____«___» _____ 2018.
(signature)

MAZMUNNOMA

Mazkur bitiruv ishi ingliz tilini o’rganuvchilar uchun “Learn by heart” mobil ilovasini ishlab chiqishga bag’ishlangan bo’lib, unda ingliz tilidagi noto’g’ri fe’llar haqida ma’lumotlar keltirilgan bo’lib, foydalanuvchi rasmli va audioli ma’lumotlar orqali o’rganishi va bilimlarini mustahkamlash uchun mashq bo’limlari mavjud. Dastur Android Studio muhitida Java va Sqlite ma’lumotlar bazasida yaratildi.

АННОТАЦИЯ

Эта выпускная работа посвящена разработке мобильного приложения «Learn by heart» для учащихся, изучающих английский язык, более того, оно включает в себя информацию о нерегулярных глаголах, пользователь может знать и приобретать знания через аудио- и графические данные, и есть практическое подразделение, чтобы укреплять навыки. Программное обеспечение создано с помощью системы управления базами данных Java и Sqlite в Android Studio IDE.

ABSTRACT

The final work is devoted to development of “Learn by heart” mobile application for English learners, moreover, it involves information as to irregular verbs, user can know and acquire knowledge via audio and image data, and there is a practice unit so as to strengthen skills. Software is created with Java and Sqlite database management system in Android Studio IDE.

Table of Contents

INTRODUCTION

The mobile app ecosystem shows no sign of shutting down anytime soon, with revenues from app stores forecast to reach $80.6 billion by 2020, up from $44.6 billion in 2016.

With 2.2 million apps in the iOS App Store and 2.8 million apps in the Play Store, creating high quality apps coupled with solid marketing strategies is the only guarantee for success.

But with so much competition, the chances that you will be able to deliver a winner are extremely low unless you follow a documented app development process.

This process will ensure that you don't waste valuable resources and time during the app development phase.

5 Stages in the Mobile App Development Lifecycle

The five steps of this process are as follows.

1. Idea

While every app starts with a core idea, your initial idea probably isn't enough to build an app which makes money or get an audience.

You need to build an app which caters to a large enough market yet is also specific enough to resonate with particular users.

Here's what you should do during the ideation stage.

Discovery and market research

Instead of starting with the design phase, spend time on extensive market research. Doing this at the very beginning can set you on firm footing and give you, your organization, and your investors confidence that your efforts are not going to be in vain.

Some guidelines for market research at this stage include:

Choosing the niche: You want to target a specific audience. So, instead of targeting all users your app could target male gamers.

Target specific user types: To stack the deck in your favor, you could target one aspect of a micro niche. For instance, your app could be only targeted towards tax professionals or lawyers. When you adopt this strategy, you can easily validate your app idea.

Solve a problem you encounter: If there is a problem, there's an app for it. From productivity to fitness to splitting bills to everything in between, apps that solve a recurring problem often get engaged and loyal users. They key here is to approach the solution in a way that others haven't so you can differentiate yourself from the competition.

Use the app store for market intelligence: Google Play and iOS App Store are a market research goldmine. This article gives a fairly comprehensive overview on the process behind using app stores for market research, including tools to use and cues to look for.

Search online: Check out the top funded projects on Kickstarter. Go on Quora and look at the most followed questions. Trawl through Angel List and ProductHunt. Check out Reddit, or go to any industry specific forum. You will get ideas with some degree of validation built in.

Check Crunchbase: Crunchbase shows details about funding patterns of apps and startups. While investors have backed plenty of duds, understanding what gets funded can help you prioritize your decisions.

Use keywords: Google Trends for a particular keyword can give you insights on popular topics. To evaluate the keywords used in the app store, a tool like App Annie is helpful.

In the context of market research and idea validation, check out our proprietary agile validation methodology which delivers better results than the industry standard.

Once you have an idea, use our Idea Grader to find out how your app concept stacks up against different frameworks like Unicorn Theory and Monopoly Theory.

If your app is designed for internal use, you won't have to go through most of the validation tactics outlined above, though you will still need to work with stakeholders and potential users to understand their daily workflow.

Establish app goals and objectives

Once you have validated your idea and understood your market, you can move on to establishing the goals and objectives of your app.

Based on the data collected from the first phase, you should determine the unique hook of your app that will differentiate it from similar apps.

At this stage, it's also useful to drill down and determine what results your users will get and the features your app will include.

Instead of bloating your app with too many features, take a lean approach and use the market insights to keep your app focused.

You should also keep app development trends trends in mind during this stage. For example, if you develop an app in 2017, you should consider app streaming, progressive web apps, or apps with AR/VR capabilities.

At the end of this process ,you should have the scope of work mapped out, and determined how much of the design process will be done in-house.

2. Design

Depending on your app budget and project scope, the design phase can be completed in a single afternoon or can take a team thousands of hours.

I. CHAPTER NECESSITY OF ENGLISH LEARNING APPLICATIONS

1.1 Substance of the English language in education sphere

1. English for spread of information:

a. The English language has so wide presence that, it helped in easy exchange of information by the newspapers, novels, books of social prominence etc. This helped in easy spread of information worldwide.

b. Further there are scientific, cultural and art related organizations which try to coordinate the knowledge worldwide with use of English language.

2. **English for extension of busyness and economy**:

a. Due to English being a common language, it was easy to for businesses on the worldwide scale. For instance the healthcare sector, stock markets, advertisements, software, banking, petroleum products, biotechnology have wide wide presence due to English as common means of communication.

b. There are many world famous authors and scientists who by their ability to write in English language gained worldwide fame. Further since their books like novels, journal were written in English, they had extensive sales. This could have not been possible if written in native language and books were sold in the native country alone.

c. Even there are many institutes, books etc. to teach English to people. These options also act as good revenue sources.

3. **English for spread and advancement of technology:**

a. English is a boon for technology progress and application. You can find a computer or iPad like devices with English commands and they are used by people of all the nations due to ease of use as customers knew the English language.

b. Further the technology could be easily spread as scientists from one country could go to other and help the host nation in development of technology as the means of communication was English.

4. **World wide employment opportunities:**

a. Because of English language, many people started migrating from developing countries to developed ones as they could easily work their due to knowledge of English. Hence you can find many emigrating to United States, Canada and Europe from other countries to get and work there due to their skills in English.

b. To a person with sound knowledge in English it is possible to work from home through a computer for clients from other countries. Some of the jobs include, data entry, medical transcription, free lance writing etc.

Today, everyone recognizes the importance of learning the English language. Moreover, the advent of the Internet has made learning English online a better option than doing it in a traditional classroom. The ability to speak and write English properly allows people to advance in the professional world. Being able to speak in English also widens one's horizon in communicating globally. A proof of the necessity of learning English is the proliferation of websites that offer English lessons online. Another proof is the increasing number of people who prefer this medium of learning everyday.

Why is learning English important to everyone in the modern day world ? Most people study English for work. Businesses worldwide use English as the universal language and medium of communication. To be able to either immigrate or work in English speaking continents like the US, Europe and Australia, one must pass examinations such as IELTS. This is also true if one want to enter a foreign school or university. But even outside the professional world, being able to communicate properly in English allows the ability to portray oneself better. This ability results to gaining better acceptance and understanding as well as respect from other people.

Everyone is cognizant of the importance of learning the English language. The easy access to computers and Internet for everyone and the increasing number of web-based English courses has made a lot of people to study English online. Here are some reasons why many people believe learning English online is better than classroom training :

• FLEXIBILITY, ACCESSIBILITY AND PRACTICALITY : Learning English online allows you to study English at your own pace at your own time. Regardless of how busy your schedule is, you can choose the most convenient time and place that you can take your English lessons. Also, learning English online allows you to take your lessons at your own speed depending on your aptitude and availability. There is no pressure from anybody except yourself. It is also very practical to learn

English online because you do not have to drive to school. You save on travel time and gasoline/transportation expenses. Moreover, the web offers a lot of free English courses for everyone to take advantage of.

• BROADER EXPOSURE TO KNOWLEDGE : Since there are a lot of available online English courses to choose from, you are not limited to the knowledge of one instructor alone. You can learn from various online instructors and training modules from across the globe. Also, learning English online exposes you to different English concepts that are applied in real business operations. In most cases, you can't really learn these concepts from books.

• ADDITIONAL KNOWLEDGE IN USING TECHNOLOGIES : Using the Internet for your English lessons, research information and interaction with other learners enhances your navigating skills through the computer and web technologies. These skills are critical in the 21st century business community that requires working with colleagues globally and across all time zones.

Aside from all the above advantages, learning English online also allows you to choose the best module and learning style that suits you whether it is for business or personal purpose. The Internet offers a wide spectrum of content that are easily accessible at your fingertips. In fact, you can even find interactive lessons that allows you to communicate with fellow learners through chats, emails and the like. This enhances your communication skills all the more.

In today's fast paced world, one must be able to adapt and be flexible while recognizing the need to learn. Web browsers and Internet connections are available everywhere, you just have to take advantage of the resources that are around you. With this in mind, learning English will not be difficult. Not only that, the knowledge and expertise you learn from learning English online will allow you to read the vast and informative news and articles in the English language. Knowledge of the English language keeps you abreast to the rest of the world .

And with the knowledge and skills that you gain from learning English online, limitless personal and business opportunities are made wide open for your taking.

1.2. Strengthen teaching English with effective mobile applications

With the mobile technologies being gradually integrated into learning and the

evolutionary development of smartphone, the market of educational mobile apps, especially for second/foreign language learning, has been rapidly growing. However, research on using mobile apps to foster English language learning remains scant.

The aim of this study is to investigate the relevant learning theories underpinning the current mobile apps for English speaking learning, the pedagogic features of these apps, and the evaluations of the apps mainly from the learners' perspective.

To examine the current mobile apps for English speaking learning, 34 apps were first searched and selected on *Google Play*. Five categories were identified for the English-speaking apps, namely pronunciation, conversation, video lesson, reference, and authentic content.

In summary, mobile apps have encouraged exciting opportunities for personalised and learner-centred environments with flexible access to learning materials anytime and anywhere.

Nowadays mobile technologies have been gradually integrated into learning. The wide use of smartphones and other portable and wireless devices has been significantly changing the ways of learning in many contexts, including language learning. Numerous mobile applications (apps) have been developed to support different aspects of second/foreign language learning, including listening, speaking, reading, writing, vocabulary, and grammar. Although these apps, usually with sound, images, and interactions, are certainly appealing to learners, the second language pedagogy that underpins these resources and activities should never be ignored. Until recently, however, research on using mobile apps to foster English language learning remains scant. No in-depth studies

have been carried out to investigate the features of current English-learning mobile apps in relation to the existing learning theories and paradigms. And the evaluation of using mobile apps for English learning from language learners' perspective is still at the early stage. Therefore, this study is intended to investigate how we might draw on existing learning theories to help us analyse and evaluate the current mobile apps for learning English. Speaking skills and Pronunciation is one of the most challenging and demanding aspects of language for learners to master. However, speaking activities are often compromised due to time constrains in the classroom. As a result, students' inability to express themselves has a negative impact on their confidence and enthusiasm. Mobile apps seem to be the ideal support for speaking learning. It could provide private, stress-free environments which allow unlimited tries until learners feel confident. Hence, in this study, I will narrow down my research scope and focus on mobile apps for learning English speaking when analysing and evaluating the current mobile apps for language learning.

No one could deny the fact that the whole world is going mobile. The widespread ownership of mobile phones and other portable and wireless devices has been dramatically changing our learning, communicating, and even life styles. Use of these mobile technologies turns out to be well aligned with educational goals such as extending learning opportunities, improving student achievement, supporting differentiation of learning needs, goals and learning styles, and deliver authentic learning materials to students who would otherwise have no access to them. Although it seems to be ubiquitous, there is yet no agreed definition of 'mobile learning' or 'm-learning'. It is not a stable concept because the field of mobile learning is undergoing rapid evolution, with increasing availability of new and more sophisticated handheld devices on the market. Another reason is

that the current interpretations of 'mobile' are not explicit enough. Nevertheless, many researchers have highlighted the 'mobility' of mobile learning.

With regard to technologies, 'mobile' generally means personal and portable. Naismith *et al* definemobile learning as learning with wireless devices such as smartphone, personal digital assistant (PDA), iPod, palmtop, laptop, *etc*. Although we could argue that mobile learning involves the use of any portable learning materials, for example, books, portable radios and DVD players, mobile learning has usually been anchored on the use of mobile technology . The type of mobile devices plays an important role in teaching and learning. Research has shown that more than three quarters of all mobile devices used in educational contexts are mobile phones and PDAs.

More recent thinking of mobile learning has emphasized the wider context of learning as part of a mobile lifestyle rather than only focusing on technology aspect.

have referred mobile learning to either formal or informal learning mediated via handheld devices and potentially available anytime anywhere. suggest mobile learning consists of a combined experience as they learn by means of mobile devices: mobility in physical space, mobility of technology, mobility in conceptual space, mobility in social space, and learning dispersed over time. By mobility in physical space, they mean the spatial movement of learners. People can learn anywhere without limitations on location. Regarding mobility of technology, Sharples *et al* not only indicate that the tools and

resources are portable but also mention the transfer attention across devices. This is supported by observation that learners tends to move between using laptop computers, mobile phone, and even touch-screen displays in public places. Kukulska-Hulme even confidently predicts that there would be no need to take a mobile device when technology becomes an integral part of our surroundings.

1.3. Purposes and tasks of final work

At the present day needless to say that learning and teaching foreign languages are so vital, therefore, software developers think about such kind of projects that improve language skills fast and efficiently. That software product helps users to learn how to speak in English that's why the program contains list of speaking topics that include audios and dialogues.

The main objective of the final work is to study how to develop English learning and teaching programs, in particular, Android apps.

To ensure the achievement of the goal the main tasks are identified as follows:

- learning implementation of mobile application development software;
- learning to utilize database management systems into app;
- learning software architecture as well as modern patterns for optimal coding;
- learning programming languages such as Java, XML required in Android Studio IDE;
- installing app on Android snartphones;

II. CHAPTER CURRENT TECHNIQUES TO ANDROID APPS DEVELOPMENT

2.1. Room technology

Room is a new way to save application data in your Android app announced at Google I/O this year. It's part of the new Android Architecture components, a group of libraries from Google that support an opinionated application architecture. Room is offered as a high-level, first-party alternative to Realm, ORMLite, GreenDao and many others.

Room is a high-level interface to the low-level SQLite bindings built into Android, which you can learn more about in Android's documentation. It does most of its work at compile-time by generating a persistence API on top of the built-in SQLite API, so you don't have to touch a single Cursor or ContentResolver. Room lets you use SQL where it's most powerful, for query building, while taking care of schema definition and data manupulation for you.

Using Room

First, add Room to your project. After that, you'll need to tell Room what your data looks like. Suppose you start with a simple model class that looks like this:

```
public class Person {
    String name;
    int age;
    String favoriteColor;}
```

To tell Room about the Person class, you annotate the class with @Entity and the primary key with @PrimaryKey:

```
@Entity

public class Person {

   @PrimaryKey String name;

   int age;

   String favoriteColor;}
```

With those two annotations, Room now knows how to make a table to store instances of Person. One thing to note when you're setting up your models is that every field that's stored in the database needs to either be public or have a getter and setter in the usual Java Beans style (e.g. getName() and setName(String name)).

Your Person class now has all the information Room needs to be able to create the tables, but you don't have a way to actually add, query, or delete a Person's data from the database. That's why you'll have to make a data access object (DAO) next. The DAO is going to give an interface into the database itself, and will take care of manipulating the stored Person data.

Here's a simple DAO for the Person class from before:

```
@Dao

public interface PersonDao {
```

```
    // Adds a person to the database

    @Insert

    void insertAll(Person... people);

    // Removes a person from the database

    @Delete

    void delete(Person person);

    // Gets all people in the database

    @Query("SELECT * FROM person")

    List<Person> getAllPeople();

    // Gets all people in the database with a favorite color

    @Query("SELECT * FROM person WHERE favoriteColor LIKE :color")

    List<Person> getAllPeopleWithFavoriteColor(String color);

}
```

The first thing to notice is that PersonDao is an interface, not a class. This interface is straightforward enough that Room could write the class for you, so that's exactly what it does! The other interesting detail is the SQL statements in the @Query() annotations. The SQL statements tell Room what information

you want to get out of the database, and they can be as simple or as complicated as you want. They're also validated at compile-time, so you can have confidence that your syntax is correct without having to run your app to see if you get a runtime exception. So if you change the method signature of List<Person> getAllPeopleWithFavoriteColor(String color) to List<Person> getAllPeopleWithFavoriteColor(int color), Room will give you a compile error:

```
incompatible types: int cannot be converted to String
```

And if you make a typo in the SQL statement, like writing favoriteColors (plural) instead of favoriteColor (singular), Room will also give you a compiler error:

```
There is a problem with the query: [SQLITE_ERROR] SQL error or missing database (no such column: favoriteColors)
```

You may notice that you can't get an instance of the PersonDao because it's an interface. To be able to use your DAO classes, you'll need to make a Database class. Behind the scenes, this class will be responsible for maintaining the database itself and providing instances of the DAOs you declared before.

You can create your database class with just a couple of lines of code:

```
@Database(entities = {Person.class /*, AnotherEntityType.class, AThirdEntityType.class */}, version = 1)

public abstract class AppDatabase extends RoomDatabase {

    public abstract PersonDao getPersonDao();}
```

This describes the structure of the database, but the database itself will live in a single file. To get an AppDatabase instance saved in a file named populus-database, you'd write:

```
AppDatabase db = Room.databaseBuilder(getApplicationContext(),
        AppDatabase.class, "populus-database").build();
```

If you wanted to get all of the Person data that's in the database, you could then write:

```
List<Person> everyone = db.getPersonDao().getAllPeople();
```

Keep in mind that RoomDatabase instances are expensive, so you'll want to create just one and use it everywhere. Dependency injection makes it easy to manage this instance.

Benefits of Using Room

Unlike most ORMs, Room uses an annotation processor to perform all of its data persistence magic. This means that neither your application classes nor model classes need to extend from anything in Room, unlike many other ORMs including Realm and SugarORM. As you saw when making mistakes with the @Query() annotations above, you also get compile-time validation of your database schema and SQL query statements, which can save you a lot of hassle compared to the runtime exceptions you'd get if you were using the native SQLite interface.

Room also allows you to observe changes to the data by integrating with both the Architecture Components' LiveData API as well as RxJava 2. This means

that if you have a complicated schema where changes in the database need to appear in multiple places of your app, Room makes receiving change notifications seamless. This powerful addition can be enabled with a one-line change in your DAO. All that you have to do is change your @Query methods to return a LiveData or Observable object.

For example, this method:

```
@Query("SELECT * FROM person")

List<Person> getAllPeople();
```

becomes this:

```
@Query("SELECT * FROM person")

LiveData<List<Person>> /* or Observable<List<Person>> */ getAllPeople();
```

Room’s Biggest Limitation: Relationships

The biggest limitation of Room is that it will not handle relationships to other entity types for you automatically like some other ORMs will. That means if you want to keep track of people’s pets like this:

```
@Entity

public class Person {

    @PrimaryKey String name;
```

```
    int age;

    String favoriteColor;

    List<Pet> pets;}

@Entity

public class Pet {

    @PrimaryKey String name;

    String breed;

}
```

Then Room will give a compiler error since it doesn't know how to store the relationship between a Person and its Pets:

```
Cannot figure out how to save this field into database. You can consider adding a type converter for it.
```

The compiler error suggests a type converter, which converts objects into primitives that can be directly stored in SQL. Since List cannot be reduced into a primitive, you'll need to do something different. This is a one-to-many relationship, where one Person can have many Pets. Room can't model relationships like this, but it can handle the reverse relationship – each Pet has a single owner. To model this, remove the pets field in Person, and add an ownerId field to the Pet class as shown:

```
@Entity

public class Person {

    @PrimaryKey String name;

    int age;
```

```
    String favoriteColor;
}
@Entity(foreignKeys = @ForeignKey(
        entity = Person.class,
        parentColumns = "name",
        childColumns = "ownerId"))
public class Pet {
    @PrimaryKey String name;
    String breed;
    String ownerId; // this ID points to a Person
}
```

This will cause Room to enforce a foreign key constraint between the entities. Room won't infer one-to-many and many-to-one relationships, but it gives you tools to express these relationships, fulfilling its promise as a high-level interface to SQLite.

To get all of the pets owned by a specific person, you can use a query that finds all pets with a given owner ID. For example, you can add the following method to your DAO:

```
@Query("SELECT * FROM pet WHERE ownderId IS :ownerId")

List<Pet> getPetsForOwner(String ownerId);
```

If you want the pets field to stay in the Person model object, then there's a few more steps. The first is to tell Room to ignore the pets field – otherwise you'll get the previously mentioned compiler error again. You can do that by annotating the pets field with the @Ignore annotation like this:

```
@Entity

public class Person {

    @PrimaryKey String name;

    int age;

    String favoriteColor;

    @Ignore List<Pet> pets;

}
```

This tells Room not to read or write that field to and from the database. This means that every time you get a Person from the DAO, the pets field will be set to null, so you'll have to populate it yourself. One way to do this is to have your Repository query for all Pets that a Person owns and set the pets field before passing a Person to the rest of your application.

Should You Use Room?

If you've already set up data persistence in your app and are happy with it, then you should keep it. Every ORM and the native SQLite implementation are going to keep working just as they have before – Room is just another data persistence option.

If you've been using SQLite or are considering using it, though, you should absolutely try Room. It has all the power you need to perform advanced queries while removing the need for you to write SQL statements to maintain the database yourself.

Alpha 3 of Room is available right now, but it's very likely that the APIs will change before the final release, so you may want to avoid using it in a production app until then.

2.2. butterknife

Tips to use ButterKnife—Dependency Injection in Android

What is Dependency Injection?

It's a design pattern in Android that allows writing codes for low coupling and also makes it possible to change them any time.

Benefits:

· Promotes loose coupling between classes and subsystems

· Easy to use and modify the components

. Testing the functionalities is easy too

Libraries available in Android:

1. ButterKnife

2. Roboguice

3. Android Annotations

4. Dagger 2

Dependency Injection with ButterKnife :

ButterKnife helps to represent the views from an activity or fragment and also helps in handling various events like onClick(), onLongClick() etc. through annotations.

How to Use ButterKnife Library?

Step 1: AddDependency

Add the following dependency to our build.gradle.

compile ‘com.jakewharton:butterknife:6.1.0’

Step 2 : Use Annotations

Instead of using findViewById() method, add @InjectView annotation before the declaration of a variable.

For example:

```
1    @bindView(R.id._view)
2    TextView _view;
```

Step 3 : Implementation

Resource Binding

Bind pre-defined resources with @BindBool, @BindColor, @BindDimen, @BindDrawable, @BindInt, @BindString, which binds an R.bool ID (or your specified type) to its corresponding field.

```
1    class TestActivity extends Activity {
2      @BindString(R.string.name) String name;
3      @BindDrawable(R.drawable.user_img) Drawableuser_img;
4      @BindColor(R.color.green) int green; // int or ColorStateList field
5      @BindDimen(R.dimen.latitude) Float latitude;
6    }
```

Views Binding

Multiple views can be grouped into a list or array.

```
01    class TestActivity extends Activity {
02      @BindView(R.id.name) TextView name;
03      @BindView(R.id.address) TextView address;
04      @BindView(R.id.email) TextView email;
05
06      @Override public void onCreate(Bundle savedInstanceState) {
07    super.onCreate(savedInstanceState);
08    setContentView(R.layout.test_activity);
```

```
09    ButterKnife.bind(this);
10
11      }
12    }
```

Non-Activity Binding

Binding can be done on arbitrary objects by supplying the view root.

Listener Binding

Listeners can also automatically be configured onto methods.

```
1    @OnClick(R.id.done)
2    public void done(View view) {
3    // TODO perform your action...  }
4    @OnClick(R.id.display_data)
5    public void DisplayData() {
6      // TODO perform your action...
7    }
```

Conclusion:

The dependency injection (DI) has become an increasingly popular tool in Android App development, and for good reasons. Injections reduce the amount of coding you perform and hence, debugging too, facilitating smoother development process and creation of better apps.

While it may be tempting to toss in dependencies to a variety of libraries, it's also important to keep in mind the potential toll that dependency injections can have on your app's performance.

2.3. Material design

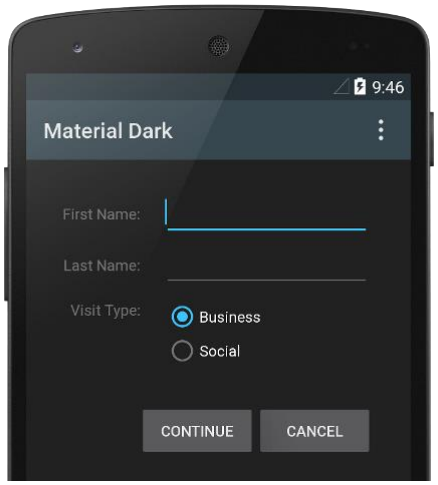
Material design is a comprehensive guide for visual, motion, and interaction design across platforms and devices. To use material design in your Android apps, follow the guidelines defined in the material design specification and use the new components and styles available in the material design support library. This page provides an overview of the patterns and APIs you should use.

Android provides the following features to help you build material design apps:

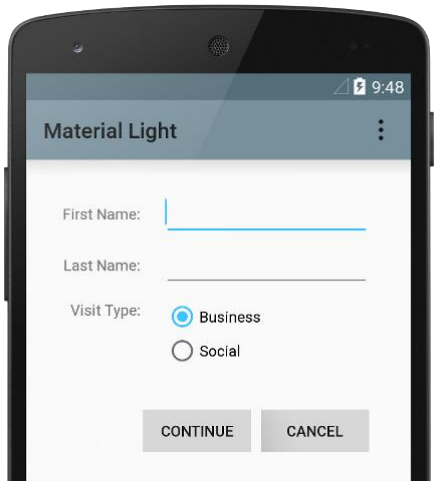
- A material design app theme to style all your UI widgets
- Widgets for complex views such as lists and cards
- New APIs for custom shadows and animations

Material theme and widgets

To take advantage of the material features such as styling for standard UI widgets, and to streamline your app's style definition, apply a material-based theme to your app.



Dark material theme



Light material theme

For more information, see how to apply the material theme.

To provide your users a familiar experience, use material's most common UX patterns:

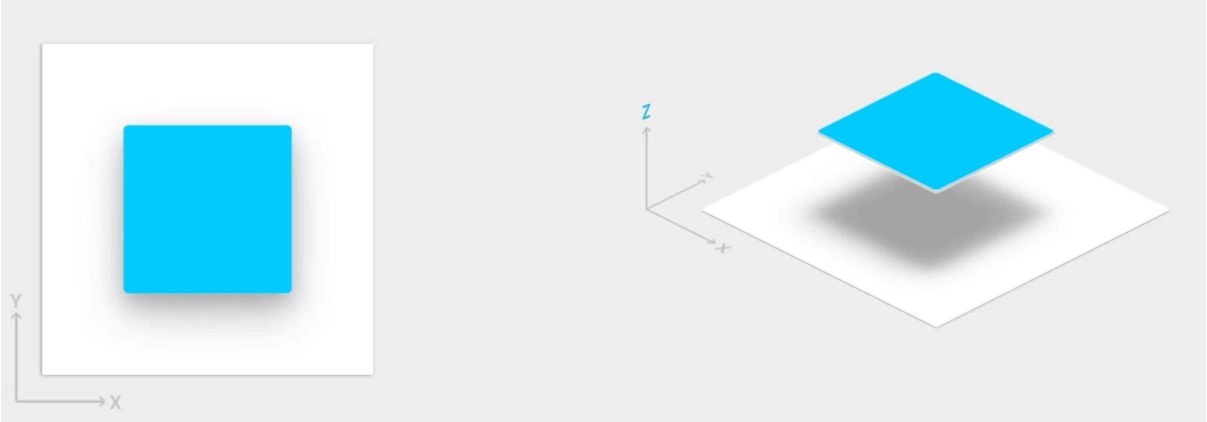
- Promote your UI's main action with a Floating Action Button (FAB).
- Show your brand, navigation, search, and other actions with the App Bar.
- Show and hide your app's navigation with the Navigation Drawer.
- Use one of many other material components for your app layout and navigation, such as collapsing toolbars, tabs, a bottom nav bar, and more. To see them all, check out the Material Components for Android catalog

And whenever possible, use predefined material icons. For example, the navigation "menu" button for your navigation drawer should use the standard "hamburger" icon. See Material Design Icons for a list of available icons. You can also import SVG icons from the material icon library with Android Studio's Vector Asset Studio.

Elevation shadows and cards

In addition to the X and Y properties, views in Android have a Z property. This new property represents the elevation of a view, which determines:

- The size of the shadow: views with higher Z values cast bigger shadows.
- The drawing order: views with higher Z values appear on top of other views.



Elevation is often applied when your layout includes a card-based layout, which helps you display important pieces of information inside cards that provide a material look. You can use the `CardView` widget to create cards with a default elevation. For more information, see Create a Card-Based Layout.

For information about adding elevation to other views, see Create Shadows and Clip Views.

Animations

The new animation APIs let you create custom animations for touch feedback in UI controls, changes in view state, and activity transitions.

These APIs let you:

- Respond to touch events in your views with **touch feedback** animations.
- Hide and show views with **circular reveal** animations.
- Switch between activities with custom **activity transition** animations.
- Create more natural animations with **curved motion**.
- Animate changes in one or more view properties with **view state change** animations.
- Show animations in **state list drawables** between view state changes.

Touch feedback animations are built into several standard views, such as buttons. The new APIs let you customize these animations and add them to your custom views.

For more information, see Animations Overview.

Drawables

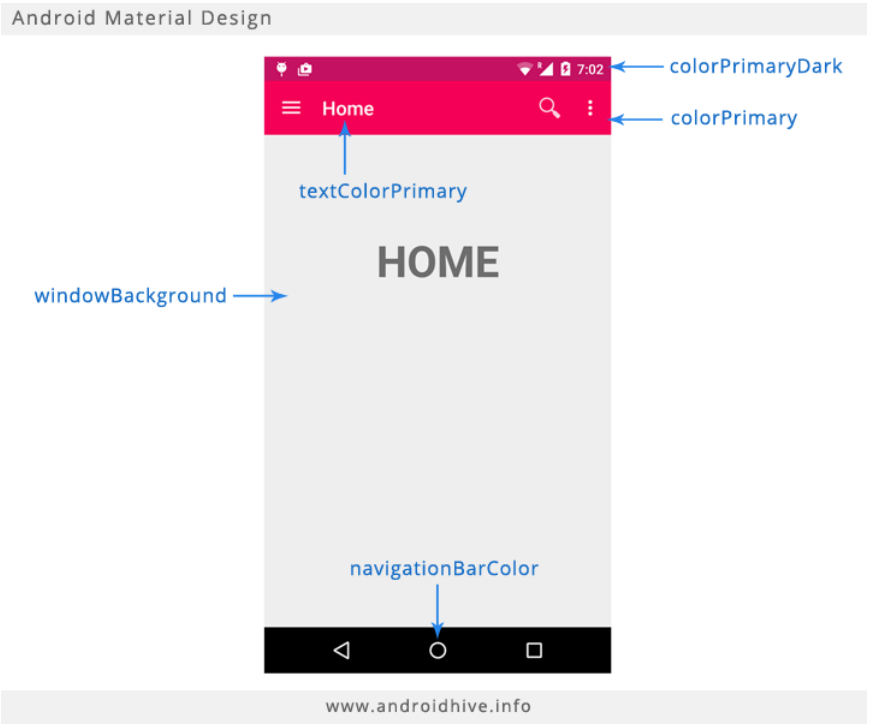
These new capabilities for drawables help you implement material design apps:

- **Vector drawables** are scalable without losing definition and are perfect for single-color in-app icons. Learn more about vector drawables.
- **Drawable tinting** lets you define bitmaps as an alpha mask and tint them with a color at runtime. See how to add tint to drawables.
- **Color extraction** lets you automatically extract prominent colors from a bitmap image. See how to select colors with the Palette API.

In 2014, Google developed a new visual design language called Material Design for Android Lollipop and higher versions. The visual specifics in material design are amusing, and the material objects have x, y, and z dimensions, which

allows you to create an incredible 3D world. Material Design is not about how to use dazzling colors, the best images, nor the elevation of the object; it is about how we create the amazing experience to users with the positive brand reality.

Google has proposed some rules and regulations while adding Material Design to application to improvise its standards. Instead of using a palette selection tool that pulls colors to the content of an app, using Material Design makes the Android application's graphic layout into a more simplified and standard format. To be noted, the material design is not only being used for rectangular or tablet screen; it should also be used for circular watch screen and other screens. So, if we create a grid, then it precepts all the spacing and should match to all the types of screens, which is a must for apps that are identified everywhere.



Overall, Material Design is straightforward, clear, and brilliant. Because of these dazzling features, it has become imperative for a broad number of gadgets.

Goals of Material Design

- Material Design aims to design the application UI like a magical paper, things that appear like real, appreciable objects.
- Animations have been pulled to make the experience more lively by safeguarding that the maximum amount of content is always visible.
- With Material Design, Google is also determined to robotize the experience for users.
- Mobile rules are fundamental, but touch, voice, mouse, and keyboard are all excellent input methods.

The materials take energy from the users — from their fingers, from their mouse click, from their touch — and use it to transform and animate.

In material design, software elements are treated as real things. For example, take paper and ink. Every pixel drawn in an application is similar to a dot of ink on a

piece of paper. Assume that paper is plain and doesn't have any color whereas the ink can be of any color. So, the content color of a paper depends on the color of the ink. Likewise, in Android applications, it can be a menu, button, or image.

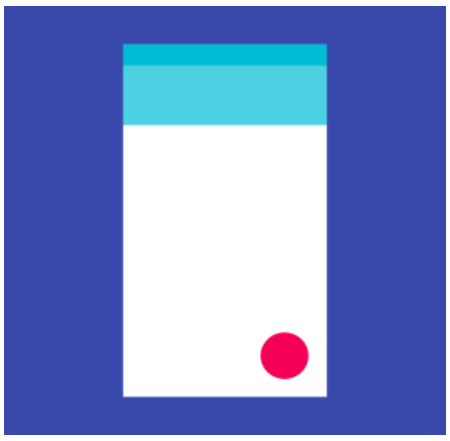
Also, the paper can be of any size. It might fill the whole screen, or it might even shrink to small square or round shape. The ink will not have any restrictions; it will be throughout the paper. It just has to fit inside the paper to be visible. The papers can change its shape, split, move, join, and re-size. Likewise, every application made in material design will have all these characteristics

Material Is the Metaphor

A material metaphor is a bring together theory of a rationalized space and a system of motion. A metaphor is a figure of speech that specifies flashy effect to one thing by observing another thing. It is open to imagination and magic.

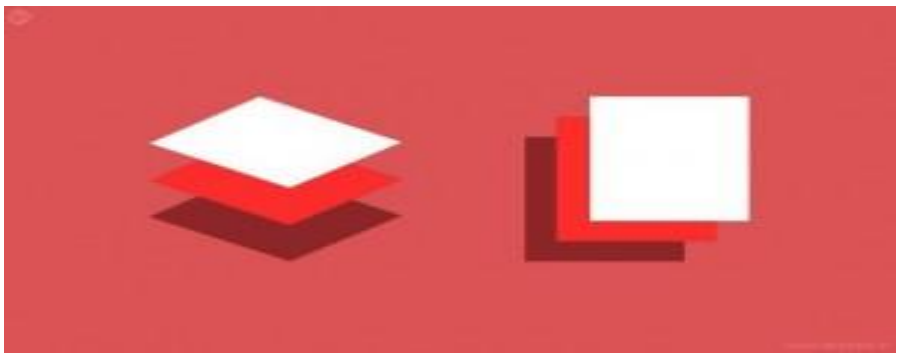
Surfaces Are Spontaneous and Natural

Surfaces and edges provide visual hints that are familiarized in our knowledge of reality. The use of ordinary material attributes conveys to a primal part of our brain and advice us to quickly understand its need.



Dimensionality Supports Interaction

The basics of light, surface, and movement are keys to transfer how objects cooperate. Sensible lighting shows bond, divides space, and demonstrates moving parts.



One Flexible Design

A single underlying design system establishes interactions and space. Each device follows a different view of the same fundamental system. Each view is made custom-fit to the size and interaction appropriate for that device. Colors, iconography, hierarchy, and spatial relationships stand constantly.

Content Is Bold, Graphic, and Wilful

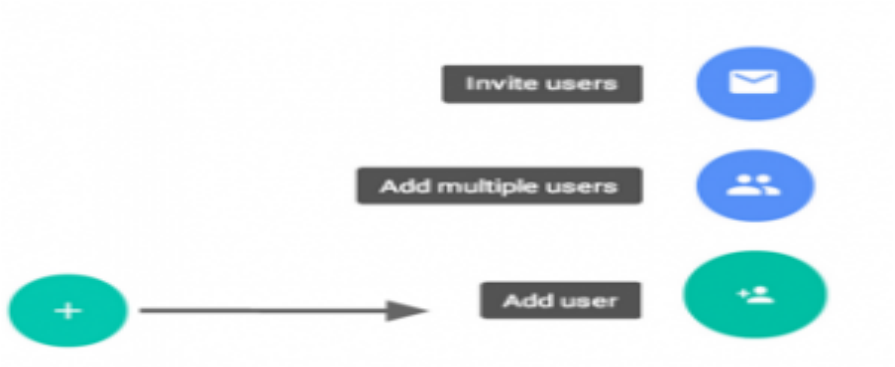
Bold content provides grouping, meaning, and focus. Cautious color choices, edge-to-edge imagery, and intentional white space create captivation and clarity.

Color, Surface, and Iconography Highlight Actions

User action is all about the significance of experience design. Color in material design is inspired by bold complexion, deep shadows, and brilliant highlights. The whole design is reconstructed by the change of points in the immediate actions.

Users Introduce Alterations and Changes

Alterations in the UI extract their energy from user actions. Motion that forces from touch respects and emphasizes the user as the best mover. It means that the widgets or material take the energy from users' fingers during the mouse click or on touch and that energy is used to animate to show it as reality.



Animation Is Choreographed on a Common Step

All action takes place in one surrounding. When objects are restructured and transformed, the user will be given the experience without collapsing the continuity of it.

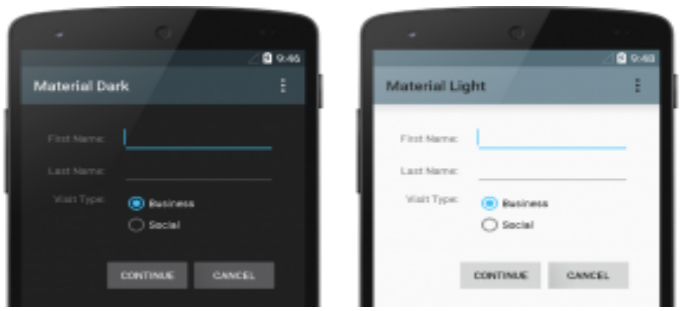
Motion Provides Meaning

Motion is meaningful and convenient. It helps focus attention and preserve continuity. The following elements assist in material design for apps of Android Version 5.0 (Lollipop) or higher.

Themes

The material theme is defined as:

- @android:style/Theme.Material: Dark version.
- @android:style/Theme.Material.Light: Light version.
- @android:style/Theme.Material.Light.DarkActionBar.



To use the material theme in your apps, customize the color palette as shown below:

```
<resources>
<!-- inherit from the material theme -->
<style name="AppTheme" parent="android:Theme.Material">
<!-- Main theme colors -->
<!-- your app branding color for the app bar -->
<item name="android:colorPrimary">@color/primary</item>
<!-- darker variant for the status bar and contextual app bars -->
<item name="android:colorPrimaryDark">@color/primary_dark</item>
<!-- theme UI controls like checkboxes and text fields -->
<item name="android:colorAccent">@color/accent</item>
</style>
</resources>
```

The following example describes how to add material design to a button.

styles.xml:

```
<resources>
<!-- Base application theme. -->
<style name="AppTheme.PopupOverlay" parent="ThemeOverlay.AppCompat.Light" />
<!-- Customize your theme here. -->
<style name="MyButton" parent="Theme.AppCompat.Light">
<item name="colorControlHighlight">@color/calbutton_focus</item>
```

```
<item name="colorButtonNormal">@color/background_color</item>
</style>
</resources>
```

activity_main.xml:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height=" match_parent ">
<Button
android:id="@+id/button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:theme="@style/MyButton"
android:layout_gravity="center"
android:text="Click"
android:textAllCaps="true"
android:textColor="@color/white"/>
</LinearLayout>
```

Cards and Lists

Cards and Lists are the two new widgets in Android with material design styles and animation. To create cards and Lists, RecyclerView can be used, which is introduced from Android version 5.0 (Lollipop). It is an adoption of ListView, which supports various layout types and contributes performance improvements. Part of data can be shown inside the card with a constant look over apps in CardView.

An example shown below demonstrates how to add a CardView in your layout.

build.gradle:

```
dependencies {
```

```
// CardView
compile 'com.android.support:cardview-v7:23.3.+'
}
```

activity_card.xml:

```
<android.support.v7.widget.CardView
android:id="@+id/card_view"
android:layout_width="200dp"
android:layout_height="200dp"
card_view:cardCornerRadius="3dp">
...
</android.support.v7.widget.CardView>
```

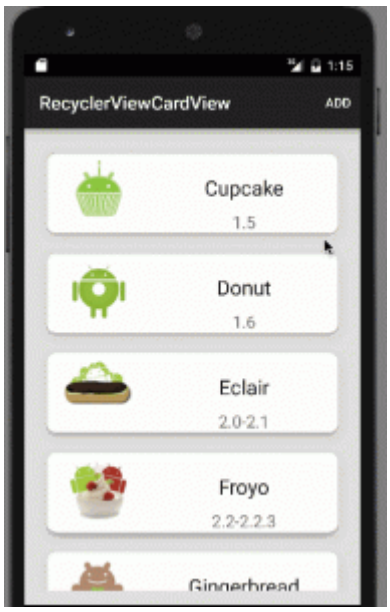
To use the RecyclerView widget in your layout, necessary attributes are shown below:

build.gradle:

```
dependencies {
// RecyclerView
compile 'com.android.support:recyclerview-v7:23.1.1
}
```

activity_main.xml:

```
<android.support.v7.widget.RecyclerView
android:id="@+id/recycler_view"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:clipToPadding="false"
android:scrollbars="vertical" />
```



Floating Action Button

Another interesting widget introduced in material design is floating action button. This button floats on UI in a circular shape with an action attached to it. By default, its behavior is to animate on the screen as an expanding piece of material.

We can also provide shadows and elevation to the buttons. The distance between surfaces and the depth of its shadow signifies elevation. To set the elevation of a view, use the android:elevation attribute in your layouts. The bounds of a view's background drawable determine the default shape of its shadow.

In addition to the X and Y properties, views in Android material design now have a Z property. This new property serves as the elevation of a view, which concludes the size of the shadow i.e., a view with greater Z values launches bigger shadows.

```
< android.support.design.widget.FloatingActionButton
android:id=”@+id/my_floatbutton”
android:layout_width=”wrap_content”
android:layout_height=”wrap_content”
android: layout_gravity="top|end”
android: src="@android:drawable/ic_add”
```

```
android:background=”@color/white”
android:elevation="5dp" />
```

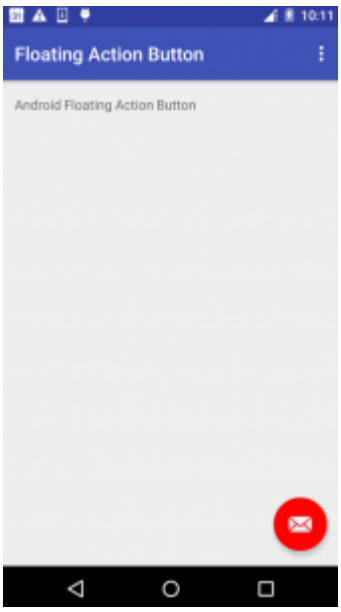
build.gradle:

```
dependencies{
compile 'com.android.support:appcompat-v7:23.1.1'
compile 'com.android.support:design:23.1.1'
}
```

activity_main.xml:

```
<android.support.design.widget.FloatingActionButton
android:id="@+id/fab"
android: layout_width="wrap_content"
android: layout_height="wrap_content"
android: layout_gravity="bottom|end" // position the floating button
android: layout_margin="@dimen/fab_margin"
android: src="@android:drawable/ic_dialog_email"/>
```

You can also define your own background color for floating button using app:backgroundTint. The size of the button can also be defined by using the app:fabSize attribute.



Collapsing Toolbar Layout

A new widget called CollapsingToolbarLayout was also introduced from Android version 5.0 (Lollipop). This comes with an amazing animation; whenever a user scrolls up, the control provides the fabulous animating effect. According to the Android documentation, CollapsingToolbarLayout is a wrapper for Toolbar, which implements a collapsing app bar. It makes the header image collapse into the Toolbar, adjusting its title size and it is designed to be used as a direct child of an AppBarLayout.

To add CollapsingToolbarLayout to your layout, see the following,

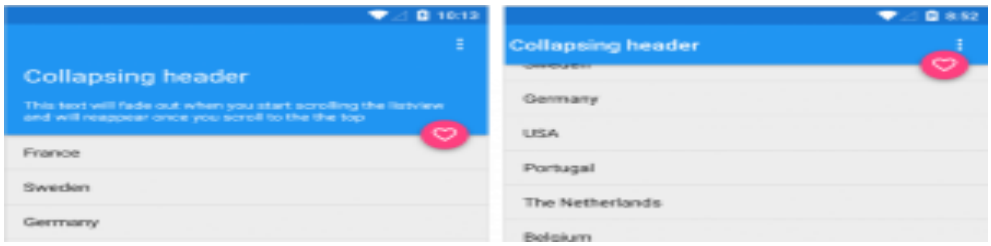
build.gradle:

```
dependencies{
compile 'com.android.support:appcompat-v7:23.1.1'
compile 'com.android.support:design:23.1.1'
}
```

activity_main.xml:

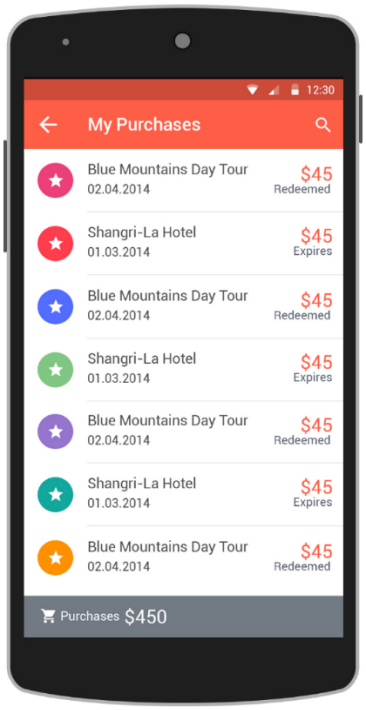
```
<android.support.design.widget.CollapsingToolbarLayout
android:id="@+id/collapsing_toolbar"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:fitsSystemWindows="true"
app:contentScrim="?attr/colorPrimary"
app:expandedTitleMarginEnd="64dp"
app:expandedTitleMarginStart="48dp"
app:layout_scrollFlags="scroll|exitUntilCollapsed">
.....
.....
</android.support.design.widget.CollapsingToolbarLayout>
```



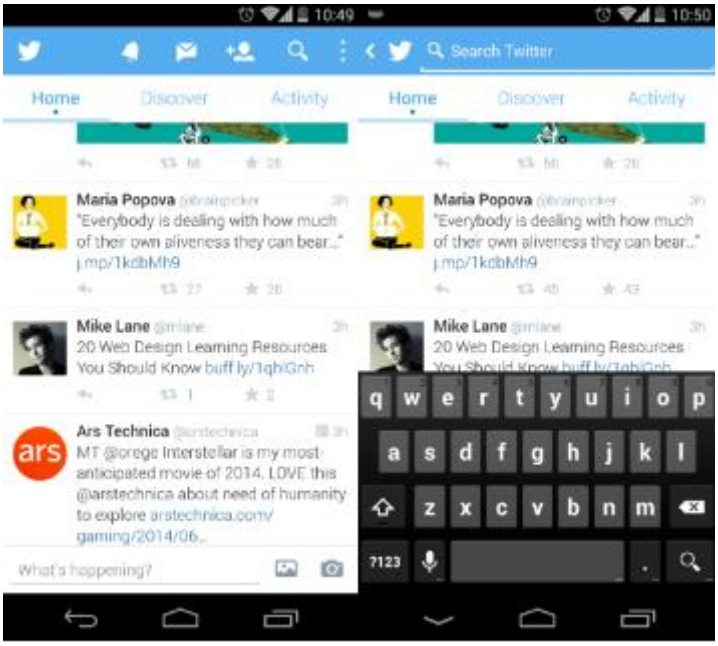
Conclusion

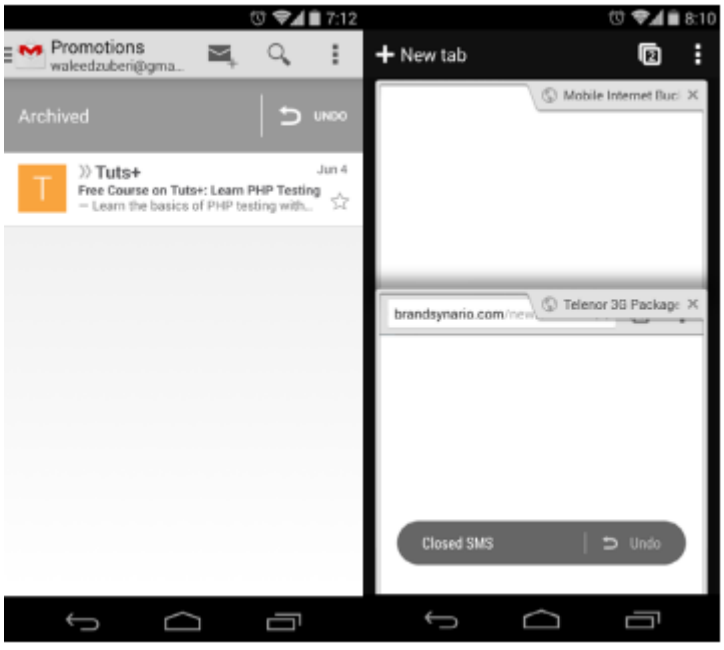
Google developed Material Design to bring together the user experience from different Google platforms. Material Design makes the user interaction smoother, simpler, and more intuitive. When you think about material design, it has so many technologies, which will only create the impression for users while using apps during interactions. The physical world is a very big part of Material Design. All in all, what do you think of Material Design in Android? Don’t you think it’s the best part to unite and enhance the user experience while using the Android application?

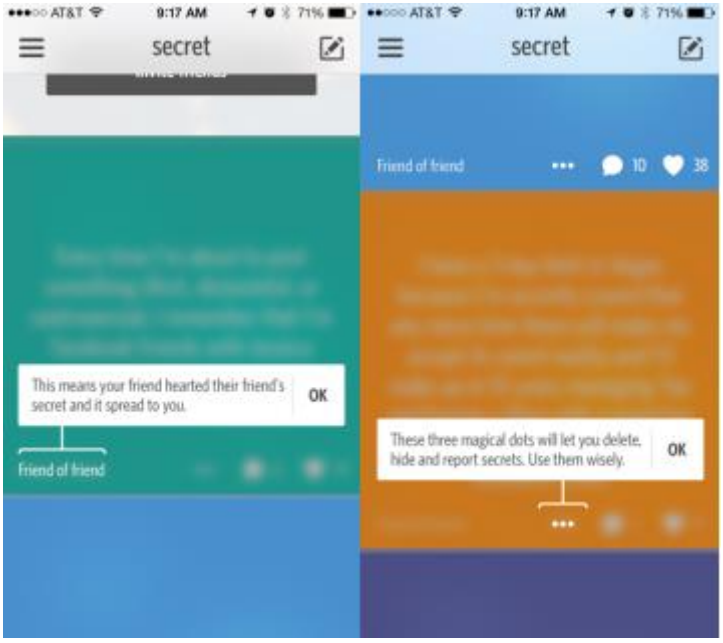

12:30
My Purchases
Blue Mountains Day Tour
02.04.2014
$45
Redeemed
Shangri-La Hotel
01.03.2014
$45
Expires
Blue Mountains Day Tour
02.04.2014
$45
Redeemed
Shangri-La Hotel
01.03.2014
$45
Expires
Blue Mountains Day Tour
02.04.2014
$45
Redeemed
Shangri-La Hotel
01.03.2014
$45
Expires
Blue Mountains Day Tour
02.04.2014
$45
Redeemed
Purchases $450

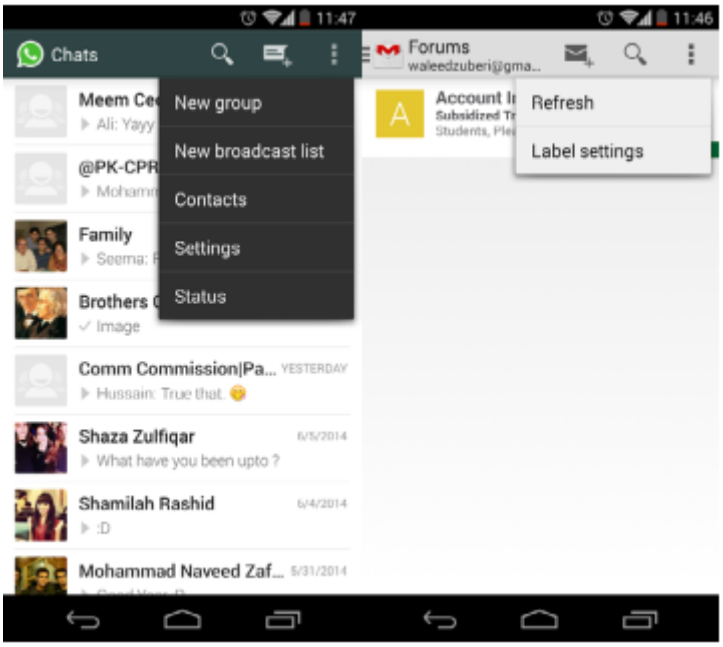
III. CHAPTER DEVELOPMENT OF "LEARN BY HEART" MOBILE APPLICATION

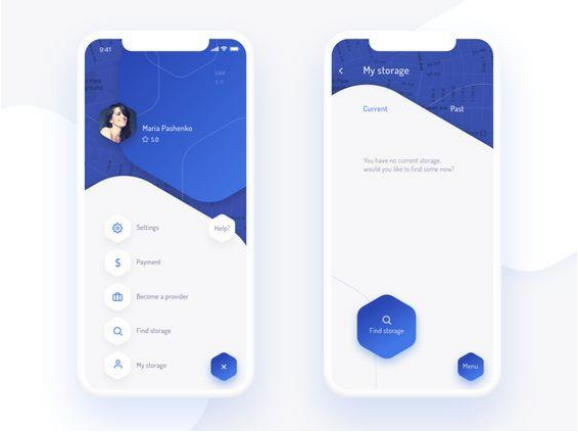
3.1. Application development progress

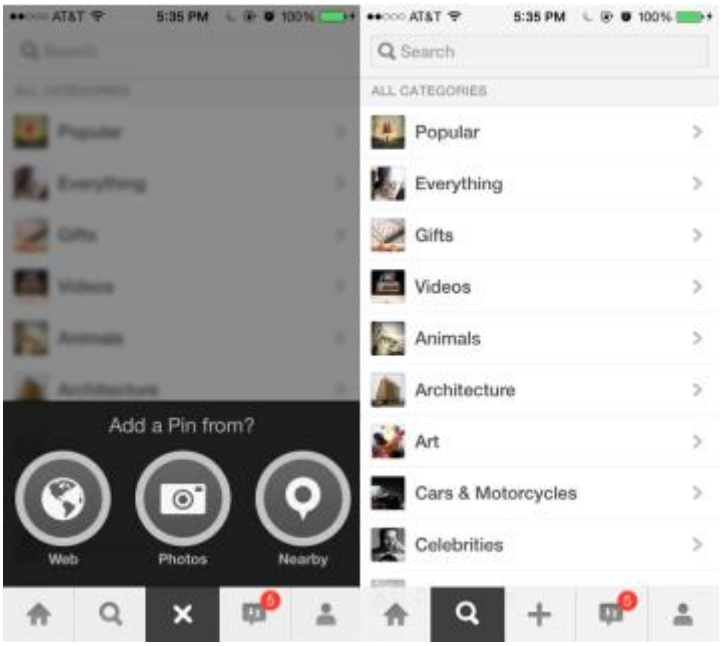


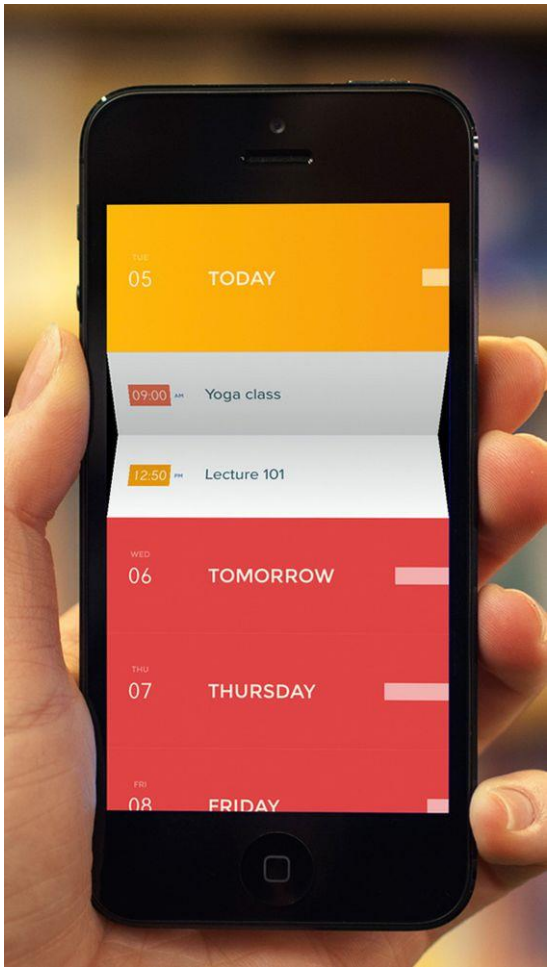



AT&T 9:17 AM 71%
secret
This means your friend hearted their friend's secret and it spread to you.
OK
Friend of friend
AT&T 9:17 AM 71%
secret
Friend of friend
10
38
These three magical dots will let you delete, hide and report secrets. Use them wisely.
OK


11:47
Chats
New group
New broadcast list
Contacts
Settings
Status
Family
Brothers
Image
Comm Commission|Pa...
YESTERDAY
Hussain: True that
Shaza Zulfiqar
What have you been upto ?
Shamilah Rashid
:D
Mohammad Naveed Zaf...
11:46
Forums
Refresh
Label settings


My storage
Current
Past





Many mobile operating system platforms share the market rather equally; developing a mobile application on one platform will not suffice in maximizing access to users. A key factor in choosing mobile application features is its portability to all major platforms. For example, the Android platform is open

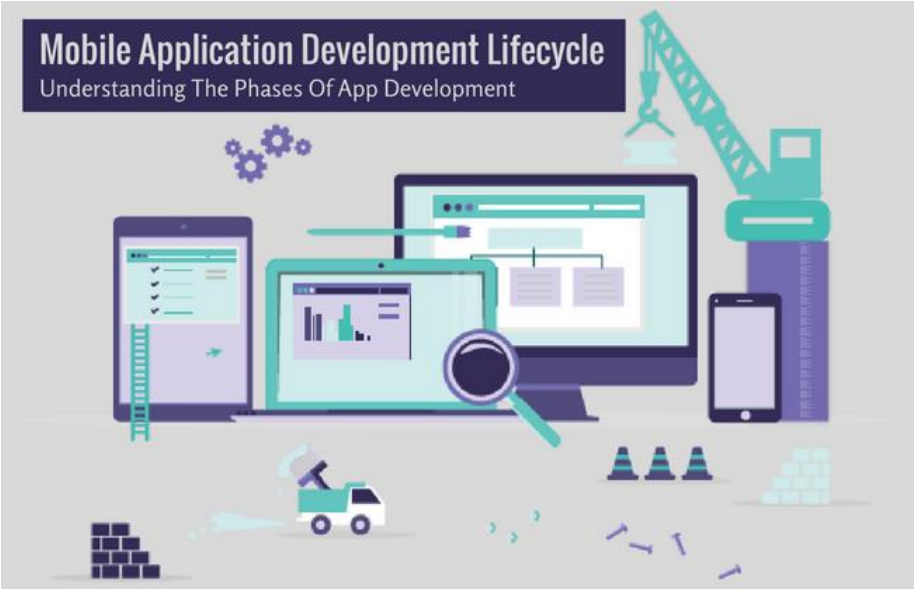
source making it developer-friendly. But other mobile platforms like iOS, Windows, BlackBerry are proprietary: they are closed and restrict developers' access to internals and thereby limiting features that can be implemented on the platform. The limitation does not exist in the same magnitude in desktop and laptop application development.

3.2. User instruction

With desktop or laptop applications often sold to the user on a CD, the user registration is not tied to the sale and is optional. Mobile applications are more often than not downloaded from the application store; user registration goes hand in hand with the download. Hence, it is easier to provide updates for mobile applications than for desktop or laptop applications. Crucial Phases Of Mobile App Development Lifecycle let's take a look at each phase of the mobile app development process and carefully understand how to build a great app.

The number of app users and apps developed each day simply seems to be accelerating. In the app market, every enterprise wants to leverage mobile app development to expand its business. According to a survey from B2B Research, around 15% of SMBs currently own a mobile app and this adoption rate is expected to increase to 50% by 2017.

With the mobile app adoption graph sloping upward, the competition in the app economy will get fierce. Therefore, the decision of quickly getting an app developed for your business could be difficult.



Moreover, Google Play Store and Apple iTunes currently consists around 2.2 million and 2 million mobile apps respectively.

IV. LIFE SAFETY

4.1. Ecological problems

4.2. Computer impact on human health

CONCLUSION

This makes it more imperative for businesses to thoroughly understand the process of app development and determine whether the app they want to develop will be a right fit for the market niche they choose and their marketing goals. Having said this, let's take a look at each phase of the mobile app development process and carefully understand how to build a great app.

Phase #1 – Preplan your App: When creating a mobile app, preplanning is the most critical phase where a lot of your time goes into researching whether your app already exists in the market or not and how you can build it better. You are required to define your app for the users to explain how it will benefit them, what problems will it resolve, what features it includes, etc. With the help of this information, you can then create the use cases which will also guide you through your project.

Phase #2: Define the Concept: Defining the concept of your app in visual terms is the first step towards discovering how your app will evolve. This is where you will have to involve your technical team as they will help you determine the feasibility of your idea along with the time and cost estimates. Thereafter, with the help of the use case information from phase 1, you can create rough sketches on whiteboard or templates to understand your app's screen representation and uncover usability issues.

Phase #3 – Assess Technical Feasibility: Apart from having cool interactions and easy-to-understand visuals, ensure that the backend systems support your app's functionality. You can follow the below steps to assess the technical feasibility of your app:

Your app should be able to access usable data. You can either source a public API or build your own abstract layer.

Determine the devices and platforms (iOS, Android, and Windows) you want your app to run on. You can either choose native app development or consult Xamarin mobile app developers to get insights on cross-platform app development.

Refine the project definition and develop a go-to marketing strategy.

Phase #4 – Create a Prototype: Defining the touch experience is difficult without knowing how it actually works. Create a prototype using the most common use cases to see how the app works. Just use a rough wireframe and allow the stakeholders to touch the prototype to provide feedback.

Phase #5 – Design your App: Prior to coding, you must design. A user experience (UX) designer will create the interaction architecture of the design elements and a user interface (UI) designer will create the look and feel of your app. This phase involves multiple steps and reviews wrapped up with an end-result that includes a visual direction or a blueprint that informs the developers of the envisioned final product.

Phase #6 – Develop Mobile App with Agile Practices: Now that your app design is ready, it's time to build the app. Prefer an agile development approach as it provides collaboration, transparency, and rapid iteration benefits that are required for adapting to change.

Phase #7 – Test the App: Finally, your app is built and now you want to test it. There are two ways to test your app:

UAT testing – Ask some of your target users/audience to test it and provide feedback or approval to see if your solution works.

BETA testing – Allow the beta users to test your app either via an open solicitation for participants or enrollment of groups identified previously. Their feedback will help you figure out whether your app's functionality operates well or not.

Phase #8 – Launch your App: Your app is now ready to be launched in the real world environment. Create some buzz using attractive write-ups and articles to

announce your app launch. Market your release with email blasts and on social media channels to generate downloads, ratings, and gain some momentum.

Now that you know how to carefully create a mobile app, kick-start it and do share your app development experience with us through the comments.

BIBLIOGRAPHY

Used internet resources

1. https://www.studyread.com/importance-english-language-education/
2. https://www.reachouttoasia.org/.../Importance_of_English_in_education_and_the_relation_with_the_culture.doc
3. https://www.bignerdranch.com/blog/room-data-storage-for-everyone/
4. http://blog.andolasoft.com/2016/10/tips-use-butterknife-dependency-injection-android.html

Appendix

```
public class MainActivity extends AppCompatActivity {
    private PageItemAdapter adapter;
    @BindView(R.id.pager)
    ViewPager pager;
    @BindView(R.id.toolbar)
    Toolbar toolbar;
    @BindView(R.id.drawer_layout)
    DrawerLayout drawer;
    @BindView(R.id.navigation)
    NavigationView navigationView;
    @BindView(R.id.tab_layout)
    TabLayout tabLayout;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ButterKnife.bind(this);
        pager.addOnLayoutChangeListener((v, left, top, right, bottom,
oldLeft, oldTop, oldRight, oldBottom) -> {

Picasso.get().load(R.drawable.fon).resize(pager.getWidth(),
pager.getHeight()).centerCrop().into((LoadImageBackground) (bitmap,
from) -> pager.setBackground(new BitmapDrawable(getResources(),
bitmap)));
        });

        setSupportActionBar(toolbar);
        ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
                this, drawer, toolbar,
R.string.navigation_drawer_open, R.string.navigation_drawer_close);
        drawer.addDrawerListener(toggle);
        toggle.syncState();

navigationView.setNavigationItemSelectedListener(this::onNavigationIte
mSelected);
        adapter = new PageItemAdapter(getSupportFragmentManager());
        pager.setAdapter(adapter);
        tabLayout.setupWithViewPager(pager);
    }

    public boolean onNavigationItemSelected(@NonNull MenuItem item) {
        return false;
    }

}
public class TabActivity extends AppCompatActivity {
    private static final String TAG = "MR";
    @BindView(R.id.pager)
    ViewPager pager;
    @Inject
    PageVerbsItemAdapter adapter;
    @Inject
    VerbAdapter verbAdapter;
    @BindView(R.id.toolbar)
```

```
    Toolbar toolbar;


    @Override
    protected void onCreate(Bundle savedInstanceState) {

DaggerMyComponent.builder().setActivity(this).build().inject(this);
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tab);
        ButterKnife.bind(this);
        setSupportActionBar(toolbar);
        pager.setAdapter(adapter);
    }
}
public final class Utils {
    public static final int[] colors = {
            R.color.colorItem1,
            R.color.colorItem4,
            R.color.colorItem3,
            R.color.colorItem2,
            R.color.colorItem6,
            R.color.colorItem10,
            R.color.colorItem11,
            R.color.colorItem8,
            R.color.colorItem5,
            R.color.colorItem9,
            R.color.colorItem7,
            R.color.colorItem12,
    };

    public static View getViewByResId(@NonNull ViewGroup viewGroup,
@LayoutRes int id) {
        return LayoutInflater.from(viewGroup.getContext()).inflate(id,
viewGroup, false);
    }

    public static boolean copyDatabaseFromAssets(Context mContext,
String mDataBaseName) {
        try {
            InputStream inputStream =
mContext.getAssets().open(mDataBaseName);
            String outFileName = "/data/data/" +
mContext.getPackageName() + "/databases/" + mDataBaseName;
            OutputStream outputStream = new
FileOutputStream(outFileName);
            byte[] buff = new byte[1024];
            int length;
            while ((length = inputStream.read(buff)) > 0) {
                outputStream.write(buff, 0, length);
            }
            outputStream.flush();
            outputStream.close();
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
```

```
    }

    public static Bitmap viewToBitmap(View view) {
        view.setDrawingCacheEnabled(true);
        view.buildDrawingCache(true);
        Bitmap b = Bitmap.createBitmap(view.getDrawingCache());
        view.setDrawingCacheEnabled(false); // clear drawing cache
        return b;
    }

    public static void shareImage(Context context, Bitmap bitmap,
String title) {
        Intent shareIntent = new Intent();
        File file = new
File(context.getExternalFilesDir(Environment.DIRECTORY_PICTURES),
"share_image_" + System.currentTimeMillis() + ".png");
        FileOutputStream out = null;
        try {
            out = new FileOutputStream(file);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        bitmap.compress(Bitmap.CompressFormat.PNG, 90, out);
        try {
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        // **Warning:** This will fail for API >= 24, use a
FileProvider as shown below instead.
//        bmpUri = Uri.fromFile(file);
        shareIntent.setAction(Intent.ACTION_SEND);
        shareIntent.putExtra(Intent.EXTRA_TEXT, title);
        shareIntent.putExtra(Intent.EXTRA_STREAM, Uri.fromFile(file));
        shareIntent.setType("image/*");
        shareIntent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
        context.startActivity(Intent.createChooser(shareIntent, "Share
images..."));
    }

}
@Entity(tableName = "verbs")
public class VerbsData {
    @PrimaryKey
    private int id;
    private String infinitive;
    @ColumnInfo(name = "transcription1")
    private String infinitiveTranscription;
    @ColumnInfo(name = "past_simple")
    private String pastSimple;
    @ColumnInfo(name = "transcription2")
    private String pastSimpleTranscription;
    @ColumnInfo(name = "past_participle")
    private String pastParticiple;
    @ColumnInfo(name = "transcription3")
    private String pastParticipleTranscription;
    @ColumnInfo(name = "as_regular")
```

```
    private int asRegular;
    private int part;
    private String description;
    private String russian;
    private String uzbek;

    public VerbsData() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getInfinitive() {
        return infinitive;
    }

    public void setInfinitive(String infinitive) {
        this.infinitive = infinitive;
    }

    public String getInfinitiveTranscription() {
        return infinitiveTranscription;
    }

    public void setInfinitiveTranscription(String
infinitiveTranscription) {
        this.infinitiveTranscription = infinitiveTranscription;
    }

    public String getPastSimple() {
        return pastSimple;
    }

    public void setPastSimple(String pastSimple) {
        this.pastSimple = pastSimple;
    }

    public String getPastSimpleTranscription() {
        return pastSimpleTranscription;
    }

    public void setPastSimpleTranscription(String
pastSimpleTranscription) {
        this.pastSimpleTranscription = pastSimpleTranscription;
    }

    public String getPastParticiple() {
        return pastParticiple;
    }

    public void setPastParticiple(String pastParticiple) {
        this.pastParticiple = pastParticiple;
```

```
    }

    public String getPastParticipleTranscription() {
        return pastParticipleTranscription;
    }

    public void setPastParticipleTranscription(String
pastParticipleTranscription) {
        this.pastParticipleTranscription =
pastParticipleTranscription;
    }

    public int getAsRegular() {
        return asRegular;
    }

    public void setAsRegular(int asRegular) {
        this.asRegular = asRegular;
    }

    public int getPart() {
        return part;
    }

    public void setPart(int part) {
        this.part = part;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public String getRussian() {
        return russian;
    }

    public void setRussian(String russian) {
        this.russian = russian;
    }

    public String getUzbek() {
        return uzbek;
    }

    public void setUzbek(String uzbek) {
        this.uzbek = uzbek;
    }

    public Bundle getBundle() {
        Bundle bundle = new Bundle();
        bundle.putInt("id", id);
        bundle.putString("infinitive",infinitive);
```

```
bundle.putString("infinitiveTranscription",infinitiveTranscription);
        bundle.putString("pastSimple",pastSimple);

bundle.putString("pastSimpleTranscription",pastSimpleTranscription);
        bundle.putString("pastParticiple",pastParticiple);

bundle.putString("pastParticipleTranscription",pastParticipleTranscrip
tion);
        bundle.putInt("asRegular",asRegular);
        bundle.putInt("part",part);
        bundle.putString("description",description);
        bundle.putString("uzbek",uzbek);
        return bundle;
    }

    public static VerbsData getInstance(Bundle bundle){
        VerbsData data = new VerbsData();
        data.id = bundle.getInt("id");
        data.infinitive = bundle.getString("infinitive");
        data .infinitiveTranscription=
bundle.getString("infinitiveTranscription");
        data.pastSimple = bundle.getString("pastSimple");
        data.pastParticipleTranscription =
bundle.getString("pastSimpleTranscription");
        data.pastParticiple = bundle.getString("pastParticiple");
        data.pastParticipleTranscription =
bundle.getString("pastParticipleTranscription");
        data.asRegular = bundle.getInt("asRegular");
        data.part = bundle.getInt("part");
        data.description = bundle.getString("description");
        data.uzbek = bundle.getString("uzbek");
        return data;
    }
}
public class PageFragment extends Fragment {
    private Unbinder unbinder;
    @BindView(R.id.top_layout)
    View view;
    @BindView(R.id.dialog_infinitive)
    TextView infinity;
    @BindView(R.id.dialog_past_simple)
    TextView pastSimple;
    @BindView(R.id.dialog_past_participle)
    TextView pastParticiple;
    @BindView(R.id.dialog_transcription1)
    TextView transcription1;
    @BindView(R.id.dialog_transcription2)
    TextView transcription2;
    @BindView(R.id.dialog_transcription3)
    TextView transcription3;
    @BindView(R.id.dialog_description)
    TextView description;
    @BindView(R.id.dialog_translation)
    TextView translation;
    @BindView(R.id.dialog_example1)
    TextView example1;
```

```
    @BindView(R.id.dialog_example2)
    TextView example2;
    @BindView(R.id.dialog_example3)
    TextView example3;

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
@Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
        return inflater.inflate(R.layout.dialog_verb_item, container,
false);
    }

    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle
savedInstanceState) {
        unbinder = ButterKnife.bind(this, view);
        if (getArguments() != null) {
            loadDataToView(VerbsData.getInstance(getArguments()));
        }
    }

    private void loadDataToView(VerbsData data) {
        infinity.setText(data.getInfinitive());
        pastSimple.setText(data.getPastSimple());
        pastParticiple.setText(data.getPastSimple());
        transcription1.setText(data.getInfinitiveTranscription());
        transcription2.setText(data.getPastSimpleTranscription());
        transcription3.setText(data.getPastParticipleTranscription());
        description.setText(data.getDescription());
//                translation.setText(data.);
//        example1.setText(data.get);
//                example2
//        example3
    }

    @Override
    public void onDestroyView() {
        unbinder.unbind();
        super.onDestroyView();
    }

    public static PageFragment getInstance(VerbsData data) {
        PageFragment fragment = new PageFragment();
        fragment.setArguments(data.getBundle());
        return fragment;
    }
}
public class VerbDialog extends BaseVerbDialog {
    @BindView(R.id.top_layout)
    View view;
    @BindView(R.id.dialog_infinitive)
    TextView infinity;
    @BindView(R.id.dialog_past_simple)
    TextView pastSimple;
    @BindView(R.id.dialog_past_participle)
    TextView pastParticiple;
```

```
    @BindView(R.id.dialog_transcription1)
    TextView transcription1;
    @BindView(R.id.dialog_transcription2)
    TextView transcription2;
    @BindView(R.id.dialog_transcription3)
    TextView transcription3;
    @BindView(R.id.dialog_description)
    TextView description;
    @BindView(R.id.dialog_translation)
    TextView translation;
    @BindView(R.id.dialog_example1)
    TextView example1;
    @BindView(R.id.dialog_example2)
    TextView example2;
    @BindView(R.id.dialog_example3)
    TextView example3;
    private ImageButton selectedButton;
    private VerbsData data;


    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle
savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        defaultColorButtons();
        loadDataToView();
    }

    @OnClick({R.id.dialog_practice_white, R.id.dialog_practice_blue,
R.id.dialog_practice_yellow, R.id.dialog_practice_green,
R.id.dialog_practice_red})
    void clickCircleButton(ImageButton button) {
        if (selectedButton != null)
selectedButton.setImageResource(0);
        button.setImageResource(R.drawable.ic_check_black_24dp);
        selectedButton = button;
    }

    @OnClick(R.id.dialog_close)
    void closeDialog() {
        dismiss();
    }

    @OnClick(R.id.dialog_flashcard)
    void openTabActivity() {
        startActivity(new Intent(getContext(), TabActivity.class));
        closeDialog();
    }

    @OnClick(R.id.dialog_share)
    void share() {
        shareImage(getActivity(), viewToBitmap(view),
infinity.getText().toString());
    }

    private void defaultColorButtons() {
```

```
        int[] colors = {R.color.color1, R.color.color2,
R.color.color3, R.color.color4, R.color.color5};
        for (int i = 0; i < circleButtons.size(); i++) {
            ImageButton button = circleButtons.get(i);
            final Drawable drawable = button.getBackground();

drawable.setColorFilter(getResources().getColor(colors[i]),
PorterDuff.Mode.SRC_IN);
        }
    }

    public void setData(VerbsData data) {
        this.data = data;
    }

    private void loadDataToView() {
        infinity.setText(data.getInfinitive());
        pastSimple.setText(data.getPastSimple());
        pastParticiple.setText(data.getPastSimple());
        transcription1.setText(data.getInfinitiveTranscription());
        transcription2.setText(data.getPastSimpleTranscription());
        transcription3.setText(data.getPastParticipleTranscription());
        description.setText(data.getDescription());
//                translation.setText(data.);
//        example1.setText(data.get);
//                example2
//        example3
    }

    @OnClick(R.id.dialog_volume_infinitive)
    public void dialogVolumeInfinitive() {
        Speaker.getSpeaker().speech(infinity.getText().toString());
    }

    @OnClick(R.id.dialog_volume_past_simple)
    public void dialogVolumePastSimple() {
        Speaker.getSpeaker().speech(pastSimple.getText().toString());
    }

    @OnClick(R.id.dialog_volume_past_participle)
    public void dialogVolumePastParticiple() {

Speaker.getSpeaker().speech(pastParticiple.getText().toString());
    }
}
public class Database {
    private static AppDatabase database;


    public static void init(Context context) {
        if (database == null) {
            database = Room.databaseBuilder(context,
AppDatabase.class, "data.db")
                    .addCallback(new RoomDatabase.Callback() {
                                    @Override
                        public void onOpen(@NonNull
SupportSQLiteDatabase db) {
```

```
                                    Utils.copyDatabaseFromAssets(context,
"data.db");
                            }
                        })
                        .openHelperFactory(new
FrameworkSQLiteOpenHelperFactory())
                        .allowMainThreadQueries()
                        .build();
            }
        }

        public static VerbsDao getDatabase() {
            return database.getVerbs();
        }


}
@Component(modules = MyModule.class)
public interface MyComponent {

    void inject(TabActivity activity);

    @Component.Builder
    interface Builder {
        @BindsInstance
        Builder setActivity(@Activity TabActivity activity);

        MyComponent build();
    }
}
@Module
public class MyModule {

    @Provides
    public List<VerbsData> getVerbs() {
        return Database.getDatabase().getVerbs();
    }

    @Provides
    public PageVerbsItemAdapter getPageVerbsItemAdapter(@Activity
TabActivity activity) {
        return new
PageVerbsItemAdapter(activity.getSupportFragmentManager(),
getVerbs());
    }
}
public class App extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        Database.init(getApplicationContext());
        Speaker.init(getApplicationContext());
    }
}
public class PageItemAdapter extends FragmentPagerAdapter {
    private String titles[] = {"All verbs", "Favourites", "Practice"};
```

```
    public PageItemAdapter(FragmentManager fm) {
        super(fm);
    }

    @Override
    public Fragment getItem(int position) {
        return new VerbsItemFragment();
    }

    @Override
    public int getCount() {
        return 3;
    }

    @Nullable
    @Override
    public CharSequence getPageTitle(int position) {
        return titles[position].toUpperCase();
    }
}
public class PageVerbsItemAdapter extends FragmentPagerAdapter {
    private List<VerbsData> data;

    @Inject
    public PageVerbsItemAdapter(FragmentManager fm, List<VerbsData>
data) {
        super(fm);
        this.data = data;
    }

    @Override
    public Fragment getItem(int position) {
        return PageFragment.getInstance(data.get(position));
    }

    @Override
    public int getCount() {
        return data == null ? 0 : data.size();
    }
}
```