Aleksandr Evseev

ANDROID APPLICATION DEVELOPMENT

Case: Studia – The diary application

Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCES
Information Technology
February 2019

ABSTRACT

Centria University of Applied Sciences	Date February 2019	Author Aleksandr Evseev

Degree programme Information Technology	
Name of thesis ANDROID APPLICATION DEVELOPMENT. Studia, The diary application	
Instructor Dr Grzegorz Szewczyk	**Pages** 37
Supervisor Dr Grzegorz Szewczyk	

The world of technologies is developing drastically, making improvements in many spheres of our life. Health care, education, production, construction, and many other industries emerge with engineering science in order to achieve greater, more accurate results. Computational power is increasing daily; people no longer require personal computers to carry out everyday tasks as the same performance is now possible with smartphones and other mobile gadgets. Considerable improvements in data storage are also visible as all the information is now available online in so-called clouds. Developments in network transmissions and embedded systems result in greater concern on mobile technologies.

One of the ways to interact with these complex computational networks are smartphones and other devices. They are running specially designed Operating Systems made to be able to execute mobile applications, which are designed and realized by programmers throughout the several stages the overall process consists of.

This thesis is dedicated to this application development process. Even though, software designing became simpler in many ways as there are many innovations brought to the field including better development environments, numerous open source libraries and frameworks available online and overall improvements in sphere of knowledge, software engineering is still a precise and complex process which requires many preparations. Throughout this work, a procedure of creating an application for an android platform will be shown.

Key words
Android, Application Design, Architecture Components, Mobile Technologies, Software Engineering

ABBREVIATIONS

DAO	Database Access Object
DB	Database
LLC	Limited Liability Company
SDK	Software Development Kit
UI	User Interface

CONTENTS

GRAPHS

FIGURES

TABLES

1 INTRODUCTION

Today the mobile application market is available for anyone who has the desire to publish their products. After a certain validation procedure the software can be released to public. The vendor leaves the marketing to the company provider including product description, advertisement, response to customer feedback and pricing policy. Product popularity and ratings heavily depend on users who are downloading and using it, as they can leave a review of the application. Eventually, the software will be offered to other customers depending on the product's number of downloads and ratings. It is a very important task to continue support and development of the application throughout its appearance on the market by improving it, adding new features and responding to customer's wishes.

In this work, a project is designed for the Android platform and aimed to be released on the Play Store which belongs to Google LLC and therefore the analysis of the target group and further explanations are correlated. The application development itself is quite intricated, it requires many stages which mostly are summarized as a requirements collection, designing, prototyping, realization and testing. There are numerous approaches designed to structure it, mostly by revising the flow, order or repetitiveness of those general actions. The two solutions that are most commonly in use are Waterfall and Agile methodologies.

The Waterfall model requires from the customer a full impact and a complete agreement on the product at the very beginning of development as, further on, any extra requirements or modifications might have fatal results on the whole project (Lotz 2018).

The Agile method allows customers to participate in product creation throughout the development. According to this model, features are sorted by priorities and, as new requirements arrive, they are given a priority and designed accordingly. This solution allows for new features to be considered throughout the time of development but doesn't ensure that they will be successfully embedded (Lotz 2018).

This thesis project was designed with accordance to the Waterfall model's approach, however an Android environment is made to be modular and with proper programming techniques, parts of the project could be made reusable therefore allowing further modifications or additions to be implemented. Following the aforementioned approach requires numerous preparations and design techniques to be

carried out before actual development is started and the further chapters of this thesis guide through them.

2 APPLICATION IDEA

The Android application that is described and designed in this work is, in general, a diary / note taking solution. It is made for people with hobbies or other activities to keep track of their progress and previous work.

An entry created by the user would have a title, content and could be supported with media such as pictures. Nevertheless, the main idea would be a categorical division. In order to create any entry a user would first have to select the category it is related to as, later on, all further operations and statistics would be mostly based on this categorical separation.

For example, considering a category called “Programming”, after the user makes several entries into this category about some work that was done or the description of some project that was created, he would be able to observe statistics stating different information such as the last date an entry was made, the number of entries overall, and others. But if desired, the user can conduct a search, a sorting or look into previously made entries.

The application is therefore summed up into two main definitions:

1. A diary tool that allows the input, organization and tracking of what work was done before.
2. A solution to receive different statistics as a motivational purpose.

3 MARKET RESEARCH

Before the application development and prototyping, a research was carried out to determine whether there are similar solutions available for the user and if this thesis' application has a potential to find its customer base. This chapter brings a comparison with the products available on the market. The "Google Play Store" from Google LLC is a software selling platform that was taken for the analysis. Search requests were made with the following keywords: "Journal" and "Diary". Similar features were marked with a tick sign in the row "Common features" in each table below.

Application Name	Diary Book – Journal With Lock, Photos, Themes			
Company Name	Lucidify Labs			
	Synchronization	Calendar support	Passcode protection	Statistics report
Common features			√	
Comment	Provides wide range of UI customization including different color pallet, several fonts. Supports rich text input, including possibility to style text input and upload media.			

Application Name	Journal it! – Bullet Journal, Diary, Habit Tracker			
Company Name	de-studio			
	Synchronization	Calendar support	Passcode protection	Statistics report
Common features	√	√		
Comment	Supports to-do lists, personal mood tracker, comments and includes widgets. The solution has wide range of features and is suitable for many needs.			

Application Name	Daybook – Diary, Journal, Note			
Company Name	Bighead Techies			
	Synchronization	Calendar support	Passcode protection	Statistics report
Common features	√	√	√	
Comment	Stands out with a simple, user-friendly interface. Main functionality is built around date organization. Includes a special feature to dictate note into text.			

Application Name	Diaro – Diary, Journal, Notes, Mood Tracker			
Company Name	Pixel Crater Ltd			
	Synchronization	Calendar support	Passcode protection	Statistics report
Common features	√	√	√	
Comment	Highlights a possibility to attach unlimited number of photos. Features searching capability and location based attachments.			

Application Name	My Dark Diary			
Company Name	Zheko			
	Synchronization	Calendar support	Passcode protection	Statistics report
Common features			√	
Comment	Simple, modern, minimalistic interface. Possibility to backup and restore data. Support for tags, checklists. Company accents on being trusted by 1 million people. Application stands out in its simplicity as of orientation to perform one task well.			

Application Name	Daylio – Diary, Journal, Mood Tracker			
Company Name	Daylio			
	Synchronization	Calendar support	Passcode protection	Statistics report
Common features	√	√	√	√
Comment	Application offers user a possibility to analyse emotional behaviour according to daily inputs with person's current mood and factors that could affect it, like activities that were done today or location where they happened. Another powerful feature of the application is a possibility to review statistics with many custom diagrams.			

Each company that was selected for the research is summarized and presented in form of a table and can be found above. The information that each table represents concludes whether application contains a common functionality and if it stands out from the rest in certain way. As it can be observed most products are either designed for everyday usage or specialized on different themes and therefore an idea of creating a journaling, note taking application with statistical feedback came to life.

4 APPLICATION DESIGN

The following chapter includes all the design techniques used to plan and develop the project. The actual process is made accordingly; every procedure considers the results of the preceding one. The blueprint of the application, as a final product, would allow to proceed to the actual development.

4.1 Requirements

A requirements collection is the starting point in any software designing process. They are collected and translated from the customer's view on the project. As is the case for the Waterfall method that was mentioned in chapter one, a complete list of requirements is settled and confirmed only at the beginning of the development.

4.1.1 Functional Requirements

In order for the user to carry out a certain task that the application was designed for, a set of functions that is required to be implemented in the product is settled and agreed among the developer team and stakeholders. Mainly functional requirements outline actions of the system that are carried out under certain conditions (Altexsoft 2018).

FR001 Application shall contain entities that are sorted under categories.
FR002 Application shall be able to perform different searching and sorting operations on Entities.
FR003 Application shall be able to provide different statistics on usage.
FR004 Entities might include optional media such as pictures.
FR005 Entities can be marked as Special to highlight them.
FR006 Application initially should include category "General" that cannot be deleted.

4.1.2 Non-Functional Requirements

Non-functional requirements define limitations of the system and patterns of its behaviour. Usually specifications are divided into categories such as usability, security, reliability, performance, availability and scalability (Altexsoft 2018).

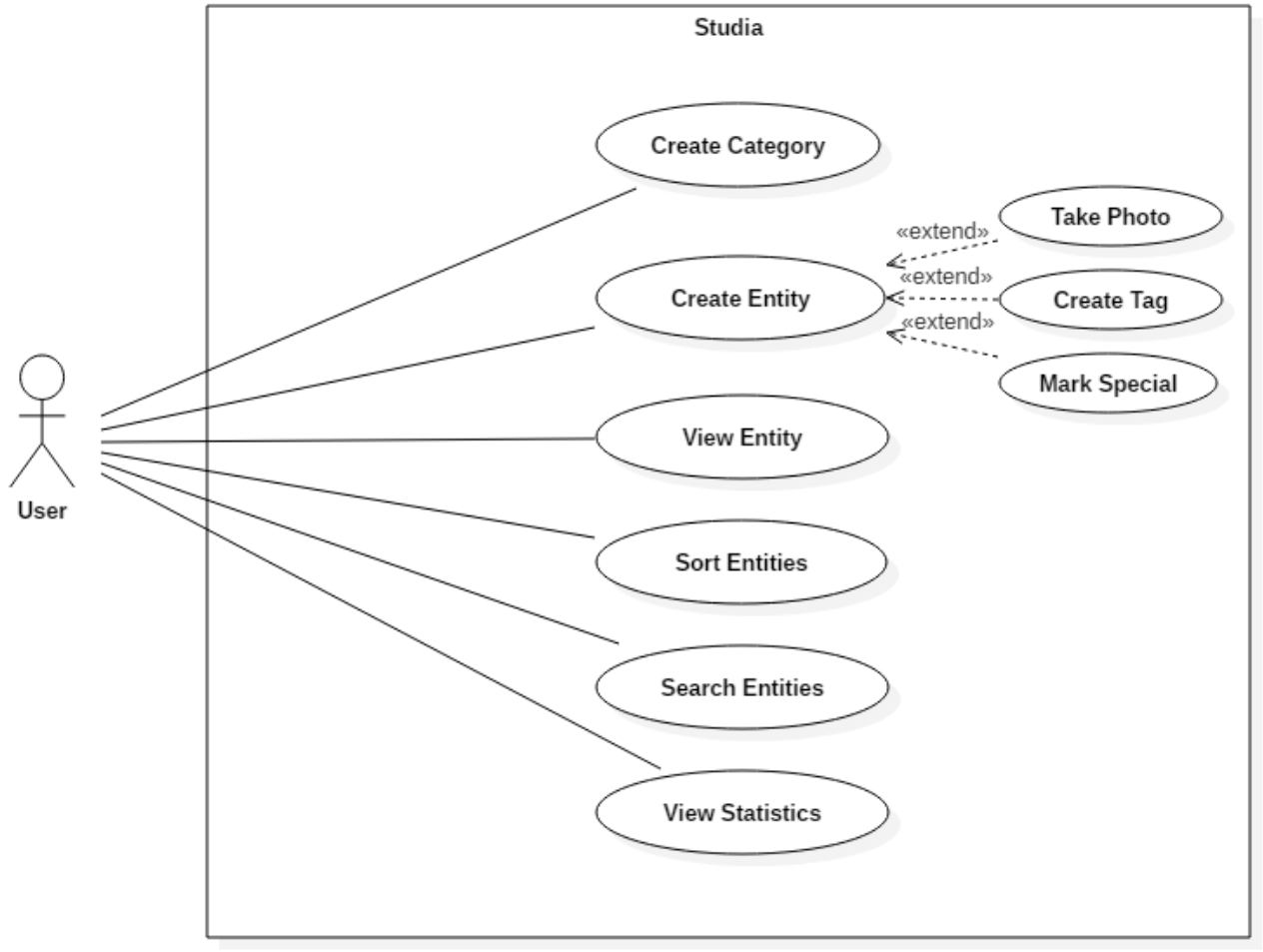
NFR001 By default entities should be sorted by date of creation.
NFR002 Sorting can be done by date, priority, date of modification.
NFR003 Searching can be run on entity's name, content, tag or date of creation.
NFR004 Statistics should include such data as frequency of input of records, the last day a record was entered, amount of days records was taken in a row without a break, number of records taken, all according to the categories where they belong.

4.2 Use Cases

Use cases define relationships between a system and actors that interact with it. Actors are the parts of the system responsible for triggering certain tasks as, for example, a user or a database. The relation between the system and the actors result in goals or functions that are to be implemented in the system. By translation of the use cases diagram the developer can form a correlated class diagram which would allow to procced to the actual application development. The translation is applied using the “Class selection process” where main classes needed for engineering are highlighted. (Sommerville 2016).

4.2.1 Use Cases Diagram

The diagram is created to resolve the main functionality of the application. It includes actors, use cases and displays relation between them (GRAPH 1).



GRAPH 1. Use cases diagram

4.2.2 Use Cases Descriptions

Each use case is summarized in form of a table and can be found below, it contains the information about different characteristics of the use case such as actors that activate the use case, a description that contains a short explanation, a trigger that activates the use case, a normal flow and alternative flow, exceptions that might happen and a short algorithm of actions that is carried out afterwards and other parameters. Use cases are numbered and brought for further analyses in following chapter.

Use Case ID:	**001**
Use Case Name:	**Create Category**
Actors:	User
Description:	User creates new category.
Trigger:	
Pre-conditions	
Normal Flow:	User is provided with a form to input new category's characteristics: a name and a colour. Application processes it to create new category. [**Exception:** Input Is Invalid], [**Exception:** Duplicated Entry]
Alternative Flow:	
Exceptions:	**Input Is Invalid:** Category's characteristics are invalid: User is notified and asked to input characteristics again. **Duplicated Entry:** Category with the same characteristics already exists: User is notified and asked to input characteristics again.
Post-Conditions:	Category is valid and created.
Includes	
Frequency of Use:	
Special Require-ments:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**002**
Use Case Name:	**Create Entity**
Actors:	User
Description:	User creates new entity.
Trigger:	
Pre-conditions	
Normal Flow:	User is provided with an interface to create a new entity with two mandatory fields required: title and content. By inputting the required fields and possibly supplying it with a photo, a tag or a mark "special" user creates new Entity. It is also possible to select a category from a list. [**Exception:** Input Is Invalid]
Alternative Flow:	User does not select category and an entity is assigned by application to cate-gory "General".

Use Case ID:	**002**
Use Case Name:	**Create Entity**
Exceptions:	**Input is Invalid:** Entity characteristics are invalid: User is notified and asked to input characteristics again.
Post-Conditions:	Entity is valid and added to the related category list.
Includes	
Frequency of Use:	
Special Require-ments:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**003**
Use Case Name:	**View Entity**
Actors:	User
Description:	User selects an entity from the provided list to view or modify its content. A view includes different controllers to operate on entity's content as for example image viewer or content text box.
Trigger:	
Pre-conditions	
Normal Flow:	Application provides an adaptive view displaying entities content.
Alternative Flow:	
Exceptions:	
Post-Conditions:	
Includes	
Frequency of Use:	
Special Require-ments:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**004**
Use Case Name:	**Sort Entities**
Actors:	User
Description:	User performs sorting on the list of entities.
Trigger:	
Pre-conditions	
Normal Flow:	User selects sorting options to be applied on the list of entities from the options menu of the application. Application returns a sorted list of items.
Alternative Flow:	
Exceptions:	
Post-Conditions:	List is sorted and can be worked with.
Includes	
Frequency of Use:	
Special Require-ments:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**005**
Use Case Name:	**Search Entities**
Actors:	User
Description:	User performs searching on the list of entities.
Trigger:	
Pre-conditions	
Normal Flow:	User provides searching options to be applied on the list of entities by input-ting a keyword or a tag that an entry might include. Application returns list of entities corresponding to search requests.
Alternative Flow:	No results regarding user search options are found and user is notified.
Exceptions:	
Post-Conditions:	List of entities corresponding to search options is provided and can be worked with.
Includes	
Frequency of Use:	

Use Case ID:	**006**
Use Case Name:	**View Statistics**
Actors:	User
Description:	Application provides statistics on different aspects of system usage (Defined in Non-Functional Requirements).
Trigger:	
Pre-conditions	
Normal Flow:	User is provided with statistics calculated based on previous input to the application. Statistics are presented in different visual forms as statements, graphs and other.
Alternative Flow:	
Exceptions:	
Post-Conditions:	
Includes	
Frequency of Use:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

4.2.3 Class selection process

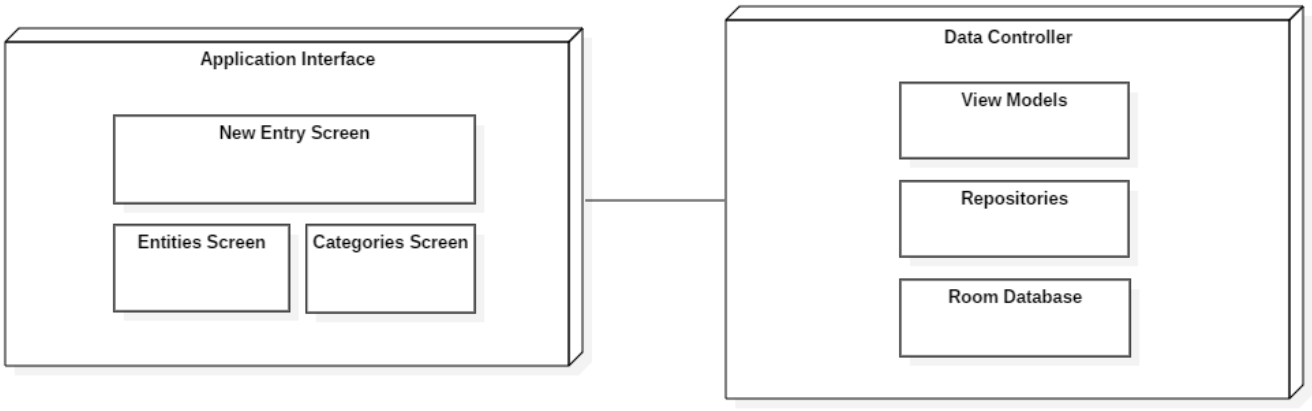
Class diagrams required to be implemented in application can partly be derived from the lexical analyses of the description of each use case from the previous chapter. The methods apply by highlighting the main parts of the sentence where nouns represent candidate classes and verbs suggest the way classes relate to each other or define the functionality they have.

TABLE 1. Lexical analysis of requirements

Use case number	Use case name	Use case description
001	Create Category	User creates new category with a provided form to input new category's characteristics: a name and a colour. Application processes it to create new category.
002	Create Entity	User is provided with an interface to create a new entity with two mandatory fields required: title and content. By inputting required fields, and possibly supplying it with a photo, a tag or a mark "special" user creates new Entity. It is also possible to select a category, otherwise it is put into category "General".
003	View Entity	User selects an entity from the provided list to view or modify its content. A view includes different controllers to operate on entity's content as for example image viewer or content text box.
004	Sort Entities	User performs sorting on the list of entities by selecting different options from the provided menu.
005	Search Entities	User performs searching on the list of entities by inputting a keyword or a tag that an entry might include.
006	View Statistics	Application provides statistics on different aspects of system usage as number of entities, last day of entry and other (Defined in Non-Functional Requirements).

The following nouns were selected through the analyses of table 1: category, entity, photo, tag, mark special, sorting, searching, statistics. The selected items might be applied as application's classes, functions or attributes.

According to the results of the analysis the two general classes would be category and entity: entity as main entry data type and category which encloses those entries. The statistics class would process user data and provide different analysis results. Searching and sorting would be realized as internal operations and would belong to the Main Activity class. Main application architecture is summarized and displayed in graph 2.



GRAPH 2. Architecture diagram

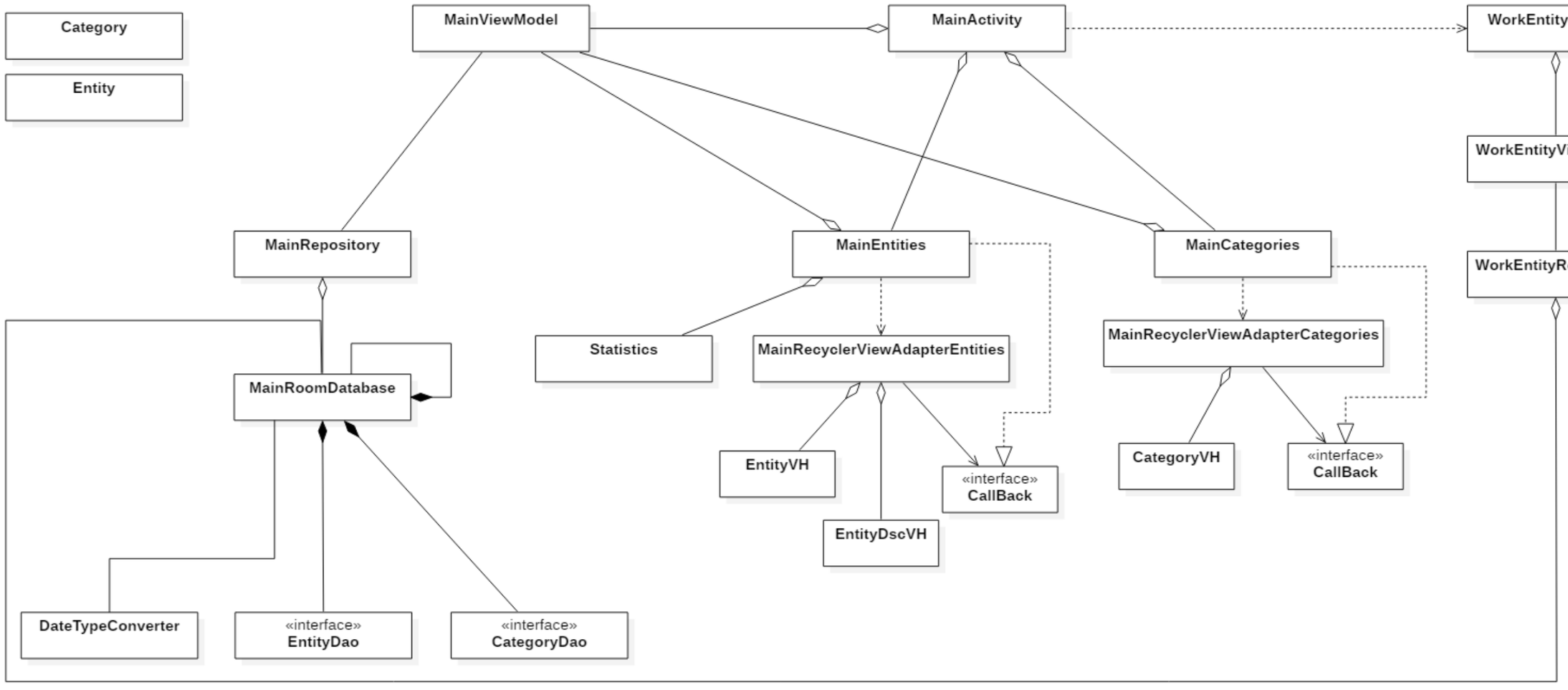
The application would consist of two activities. First, the Main Activity includes two fragments, one to display a list of entities and provide a control on searching, sorting and another one to display a list of categories. Both fragments feature adding and modifying of entries which in Entities Fragments is realized with another activity and in Category Fragments with an Alert Dialog. Second, the Work Activity to view and modify Entity class objects. An activity includes text inputs for title, content, tags and actions to add photo or put a mark “Special“.

Additionally, the whole application would be driven by a Room database explained in chapter five which would require a single class for itself and a DAO class for each table in the database. Moreover, each separate Android Activity View would require a View Model and a Repository to maintain a clean architecture.

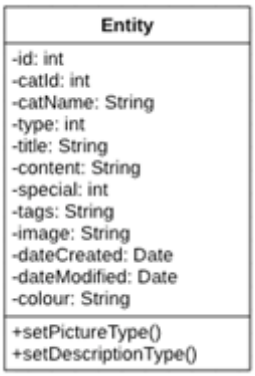
The Room database does not allow operations on the UI thread therefore each database operation should be executed asynchronously. This requirement is achieved in different ways, several methods have a LiveData as a return type so therefore no additional functionality needs to be added and other functions are executed in the accordingly designed class that extends AsyncTask and performs operations in the background.

4.3 Class Diagram

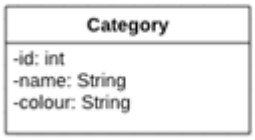
Below a complete diagram of classes used in the application can be observed (GRAPH 3). Class diagram describes a static presentation of the product which is later processed into the source code of the application. Later in this chapter each class is separately brought for parsing into attributes, operations and short description.



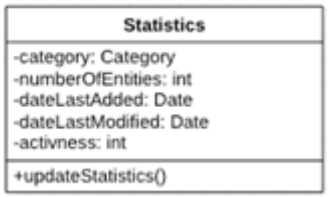
GRAPH 3. Class diagram



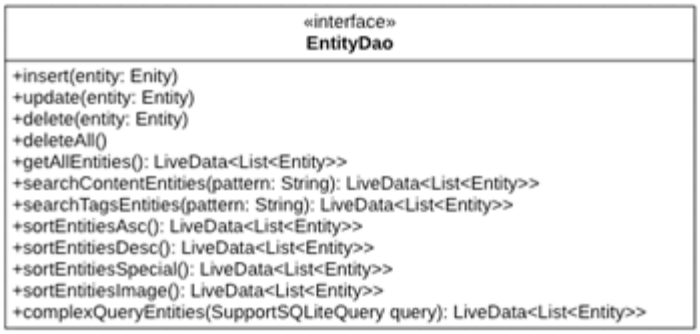
Entitiy class is a main entry, it encloses user input details as title, content, image, tags, special mark and category that is belongs to. Additionally, it automatically records date of creation and modification.



Category class groups all entries into categories, it includes user defined name and color.

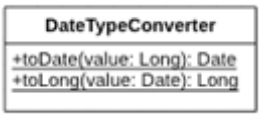


Statistics class contains statistics on certain category object.

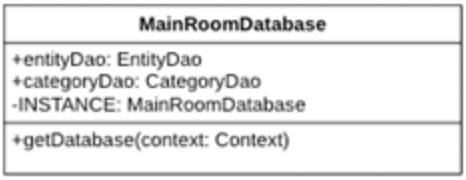


EntityDao interface is a Database Access Object defined to manipulate on Entity table in database. It performs different operations on DB to insert, update, delete, retrieve, sort or search.

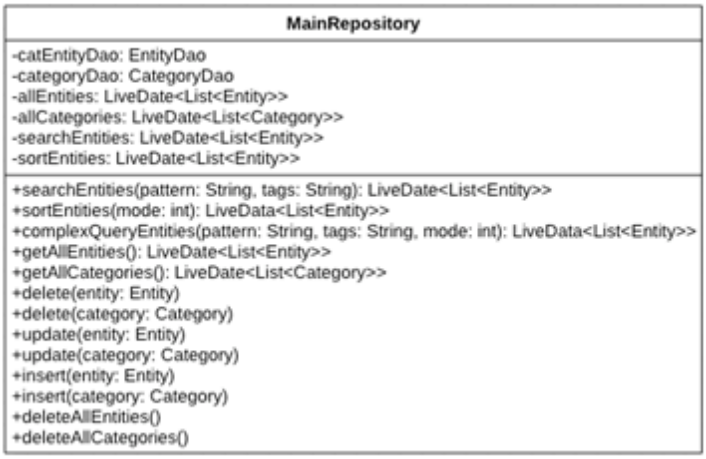
CategoryDao interface is a Database Access Object defined to manipulate on Category table in database. It performs different operations on DB to insert, update, delete or retrieve.



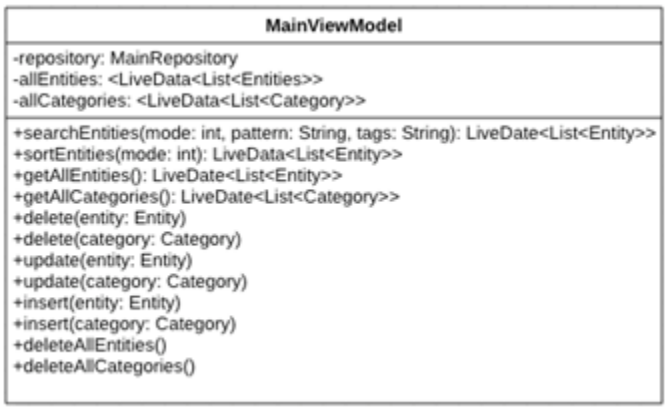
DateTypeConverter class performs conversion from Date type to Long type as date is stored in format Long in database and used in format Date in the application.



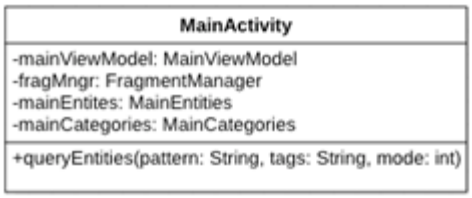
MainRoomDatabase class is a singleton class that provides access to the database. It also includes two interface getters to DAO objects.



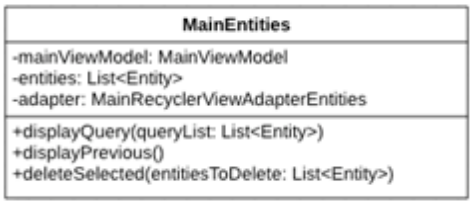
MainRepository class performs all required operations on data retrieved from the database or requests database to perform certain operations. All operations on DB are carried out asynchronously to prevent loading of the main thread.



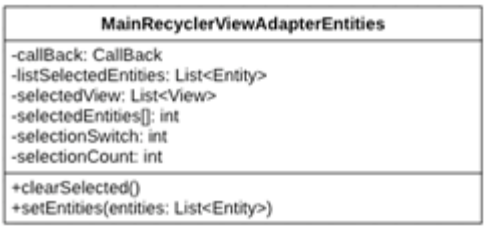
MainViewModel class links provided data from the Repository to the corresponding UI elements requesting it. There are no complex operations on data carried out in this class.



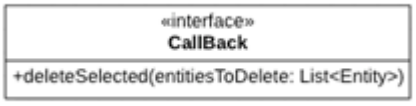
MainActivity class is the main application class, it contains two fragments and handles switching between them. It also carries the main load of user interfacing with the application.



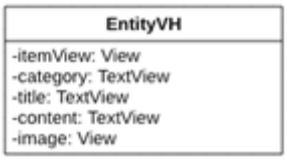
MainEntities class represents a fragment that displays a list of entities and all changes applied to it as searching, sorting and selection.



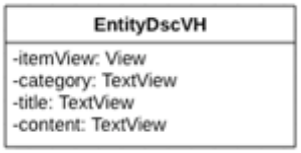
MainRecyclerViewAdapterEntities class represents an Adapter class for Recycler View in MainEntities class. It embeds functionality for multiple selection and manipulates two different View Holders for object models with and without supplementary image.



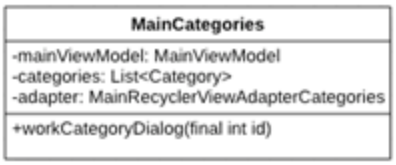
CallBack interface allows an Adapter to callback class MainEntities to perform operation on selected Entities.



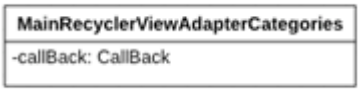
EntityVH class represents a custom View Holder for Entity object that includes image.



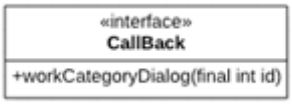
EntityDscVH class represents a custom View Holder for Entity object that does not include image.



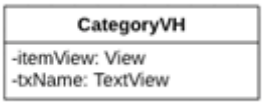
MainCategories class represents a fragment that displays a list of categories and provides a possibility to delete or modify it. It also provides a functionality to sort list of entities from MainEntities by the category they belong to.



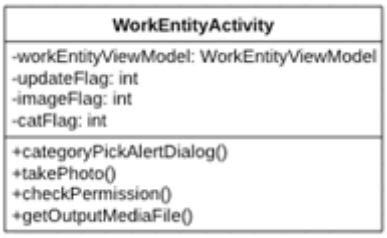
MainRecyclerViewAdapterCategories class represents an Adapter class for Recycler View in MainCategories class.



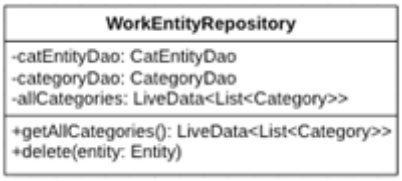
CallBack interface allows an Adapter to callback class MainCategories to perform operation on selected Category.



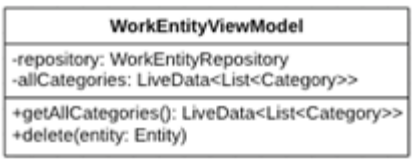
CategoryVH class represents a custom View Holder for Category.



WorkEntityActivity class displays and provides a possibility to modify single Entity. It includes functionality to choose and select a category, to create tags, to take a photo or to mark Entity as Special.



WorkEntityRepository class performs all required operations on data retrieved from the database or requests database to perform certain operations. All operations on DB are carried out asynchronously.

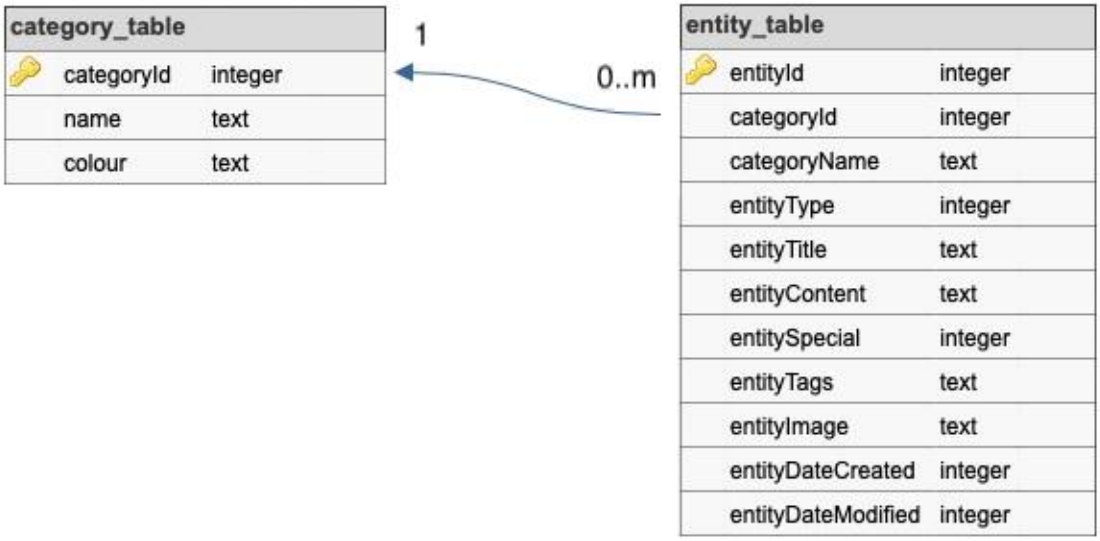


This class links provided data from the Repository to the corresponding UI elements requesting it. There are no complex operations on data carried out in this class.

4.4 Database

Studia, an application designed and developed in this work, is driven with an SQLite database, that is controlled by the Room Persistence Library which is designed to simplify the DB operations and provide more accurate interaction. All the data that is retrieved, processed and put back into the database require properly defined SQL Queries. Below you can find a DB schema and a list of SQL commands used in the project.

4.4.1 Database Schema



GRAPH 4. Database schema

The database consists of two tables containing categories and entities (GRAPH 4). Both tables have Primary keys functioning as entry identification: “categoryId” and “entityId” respectively. They are unique, meaning that all entries in the table have different IDs and are set to autoincrement by the database itself. The “categoryId” field in the entity_table uniquely identifies a “categoryId” in the category_table, where one category can have zero to many entities.

4.4.2 Database Queries

Room Persistency Library provides ‘@’ Java Annotations to perform simple operations on insert, delete and update. Other operations were executed with custom SQL requests listed below.

TABLE 2. SQL query descriptions

Query	Description
`SELECT * FROM entity_table`	Provides all entities from the enitity_table table
`SELECT * FROM entity_table` `ORDER BY entityDateCreated`	Provides all entities sorted by the time of creation.
`SELECT * FROM entity_table` `ORDER BY entityDateCreated`	Provides all entities sorted by the time of creation.
`SELECT * FROM entity_table` `ORDER BY entityDateModified DESC`	Provides all entities sorted by the last time they were modified.

`SELECT * FROM entity_table`
`WHERE`
`entityContent LIKE :pattern`
`OR`
`entityTitle LIKE :pattern`
	Searches for results where pattern matches a title or a content of entity.

`SELECT * FROM entity_table` `WHERE entityType = 1`	Provides all entities that include image
`SELECT * FROM entity_table` `WHERE entitySpecial = 1`	Provides all entities that are marked special
`SELECT * FROM entity` `WHERE entityTags LIKE :tag`	Provides all entities that include specific tags in tags string

`SELECT * FROM entity`
`WHERE`
`entityTags LIKE :tag[0]`
`AND entityTags LIKE :tag[1]`
	Provides all entities that include multiple tags provided

Simple Query operations presented in the table 2 are recorded in the application and executed when needed. For more complex operations on database Query command is assembled at the run time by following the next algorithm displayed in figure 1.

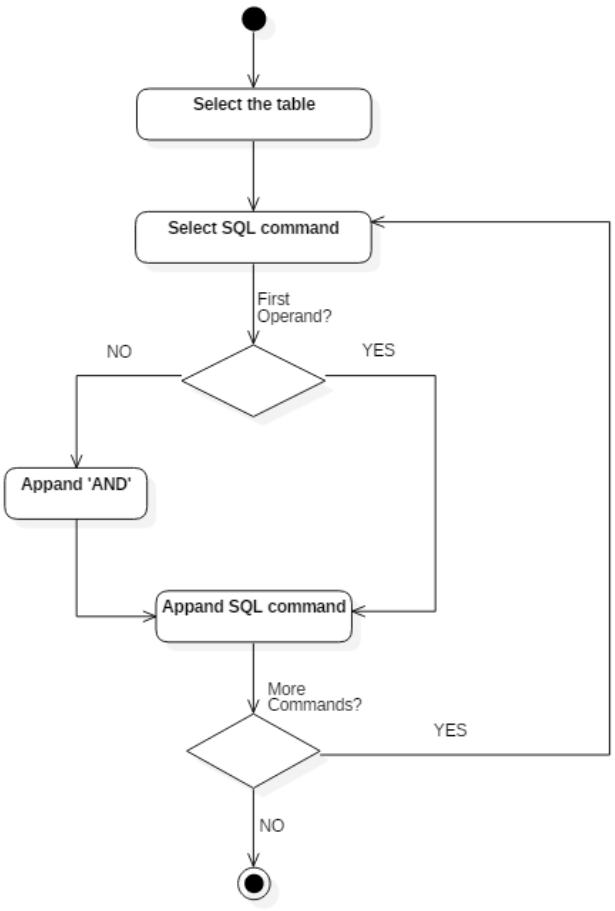


FIGURE 1. Activity diagram on query composition.

All commands regarding searching and sorting are processed first, SQL query is appended with every required parameter and finished with an either selected ordering types. The only operation to highlight is a search on tags as user can define how many tags to be searched on as needed therefore operation increase in length by appending every tag search to operation with an ‘AND’ conjunction.

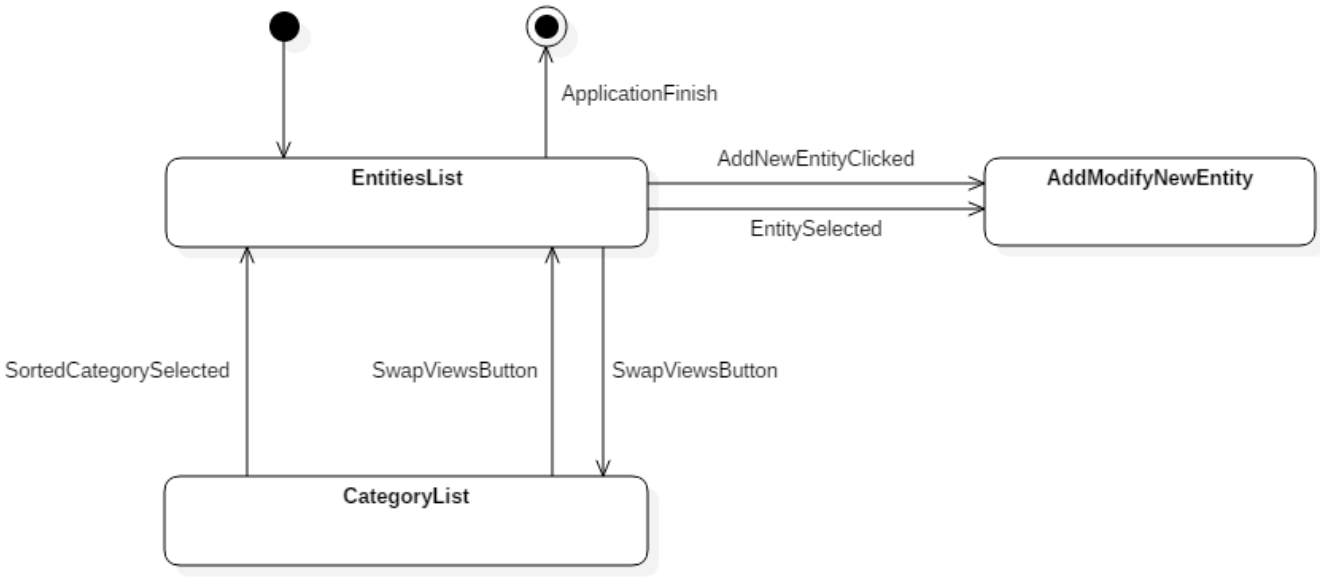
Consider an example where the user requests search by pattern, includes two tags, sort for entities marked special which have images and all of it sorted by last date of modification. The Final SQL request is presented below (SOURCE CODE 1).

```
SELECT * FROM entity_table
WHERE (entityTitle LIKE ‘%random% OR entityContent LIKE ‘%random%’)
            AND (entityTags LKE ‘%tag1%’ AND entityTags LIKE ‘%tag2%’)
            AND entityType = 1
            AND entitySpecial = 1
ORDER BY entityDateModified DESC
```

SOURCE CODE 1. The SQL clause created according to the described above algorithm.

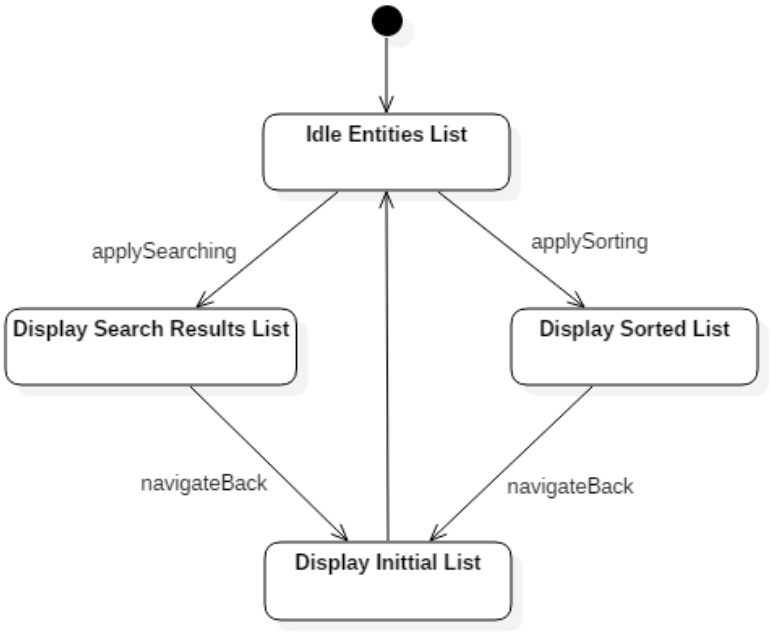
4.5 User Interface Design

The user interface was designed to be responsive, minimalistic and most importantly efficient in use. State machine diagrams were applied to determine a number of screens required to carry out the main functionality and relationship between the elements presented in them. Main views have to be adaptive to provide flexibility and exclude misuse of the application which is also taken into account and presented below. Accompanied by the UI mock-ups the picture of the User Interface is formed and can be considered for the further development. (Material Design Foundation 2019)



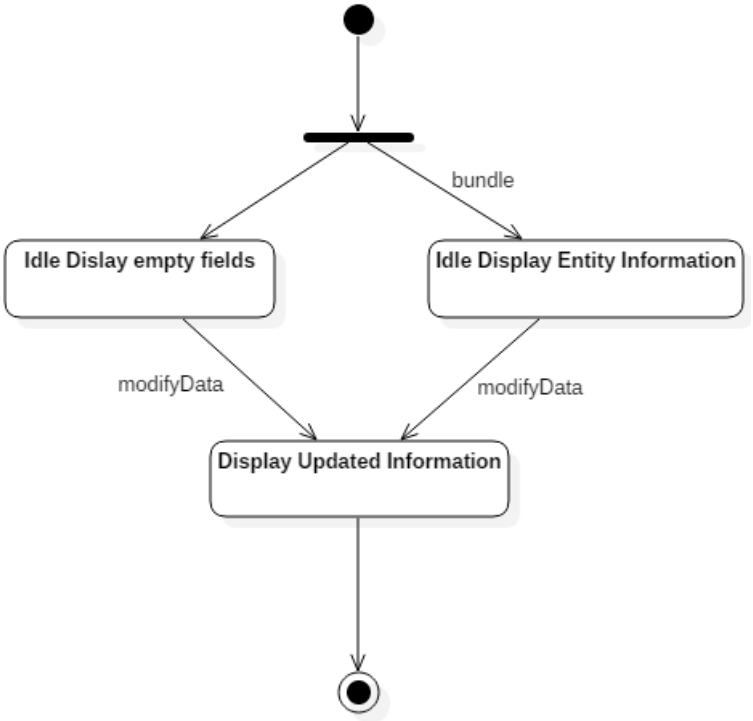
GRAPH 5. Application state machine diagram

As can be observed on the graph 5 presented above, a single activity presented as “AddModifyNewEntity” is used in two occasions whether a new entity is created or an existing one if modified or displayed. Switching between two main activities “EntitiesList” and “CategoryList” is accomplished with a corresponding button or an item selection from the category list in which scenario “EntitiesList” is reused to display a related sorting.



GRAPH 6. Entities state machine diagram

As can be observed in graph 6, MainEntities activity initially displays a list of items sorted by date of creation. When searching or sorting operation is applied the list is updated to display results. After an operation is performed the list can be worked with or returned to initial condition.



GRAPH 7. Create / Modify Entity state machine diagram

Graph 7 represents the activity to create or modify entities display the current information of an entity if it was selected, otherwise it provides empty fields to create one. It returns a new or an updated entity if information was provided otherwise the action is discarded.

With accordance to the main system dynamics acquired from the uses cases diagram, a state machine diagrams were created for each screen, required to accouple application which is sketched below in figures 2, 3 and 4 in form of a mock-ups. It was designed with a special application “Sketch” that is specialized on vector graphics editing.

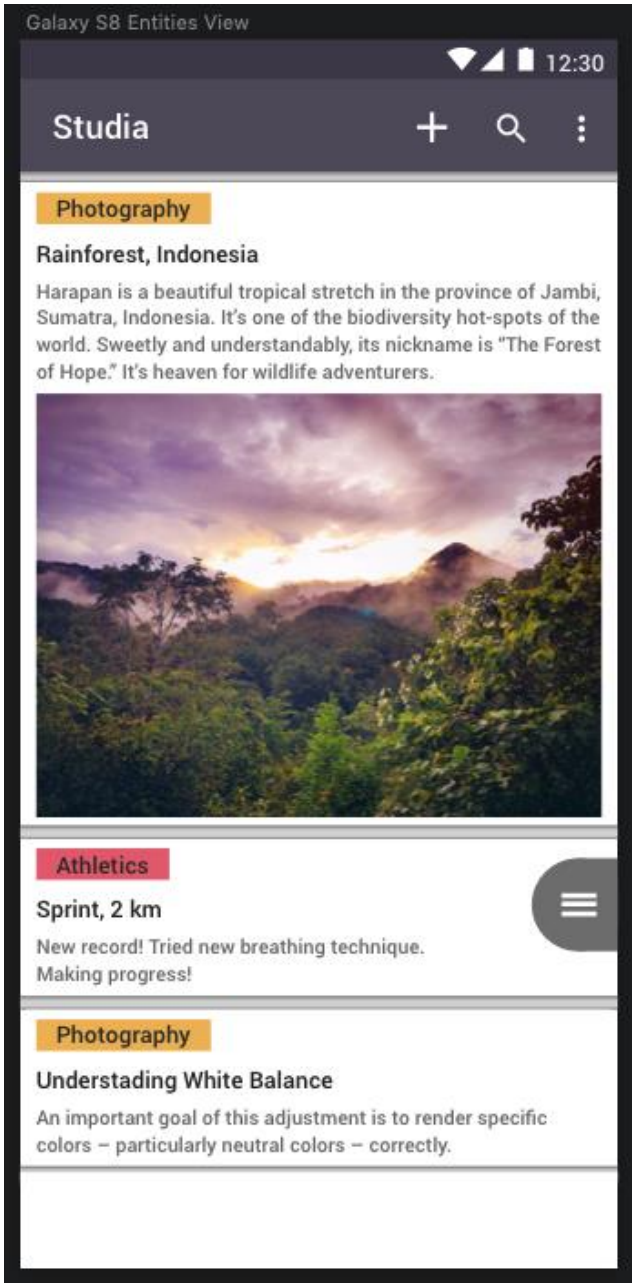


FIGURE 2. Mock-up entity screen

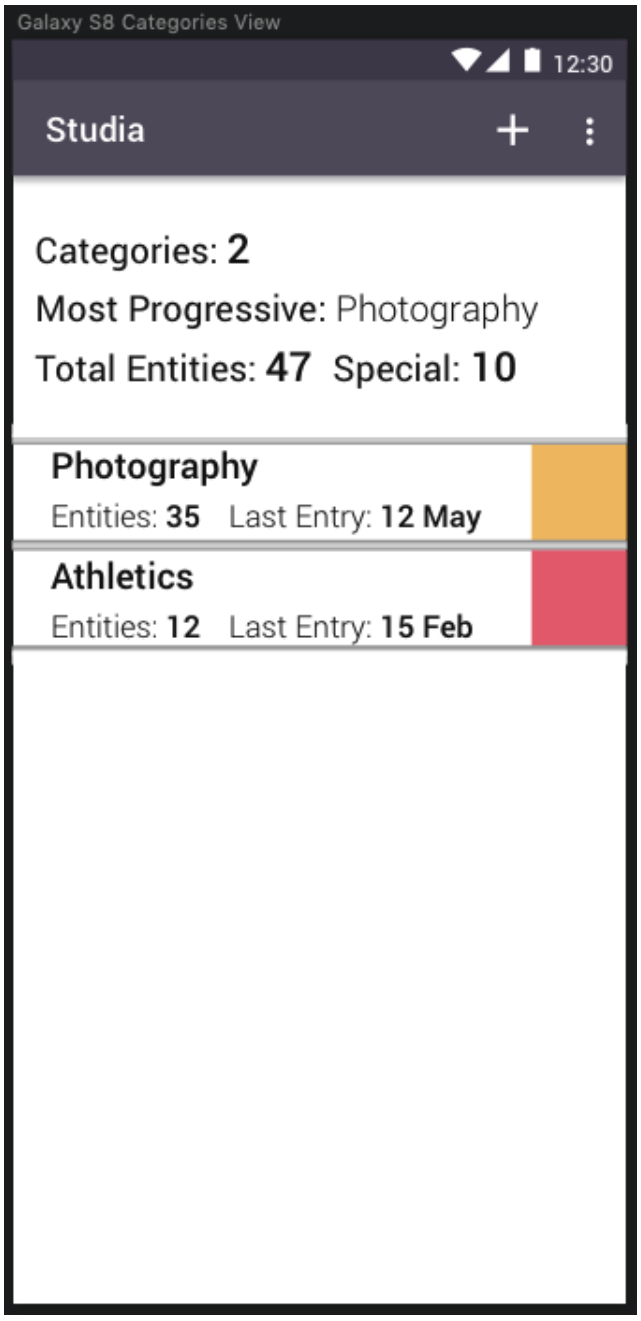


FIGURE 3. Mock-up category screen

FIGURE 4. Mock-up entities view / editing

5 TECHNOLOGIES USED

Android as an Operating System has gone a long way. Google is frequently releasing updates trying to improve the system for a better experience to the end users. Nevertheless, development tools are also being upgraded; newer technologies are provided annually to improve, modernize and ease the process of Software Engineering (Haase 2019).

Flutter, a tool that has recently received a considerable update is a cross-platform development technology that allows production for such platforms as Android and iOS and promised support for desktop operating systems in the future. This framework states to be fast in development time while allowing the creation of expressive and flexible user interfaces keeping performance at native application speed. It is mentioned to be the principal technology of Google's future operating system Fuschia. Flutter is also beneficial for developers in many ways as it is open-sourced, development tools are cross-platform, and the prefabricated widgets feature Google's Material Design and Apple's Cupertino for rapid application prototyping (Flutter 2019).

Material Design is a set of ready-made themes and guides on proper application user interface designing. Each theme presents a complete asset of patterns made in one stylistic manner including fonts, colors, shapes and specific UI elements as bars, buttons and others. The official documentation provides tutorials on proper design techniques for better user impact and more efficient use of application (Mew 2015).

Architecture Components are the most recently introduced development libraries aimed to propagate a proper application design. It introduces lifecycle-aware components; as said on the official website this framework would help a developer to manipulate on application components and their lifecycles which would prevent memory leaks, UI freezes and would overall resolve complications on configuration changes. The main case of its concern is interaction between application's User Interface and Data Processing layers (Android Architecture Components 2019).

If the functionality of these two aforementioned layers is combined in a non-properly designed solution, it could result in many issues such as freezing UI, application slowdowns, crashes and others. The solution designed to resolve this problem is represented in a proper separation of functionality and can be observed in the figure 5 below.

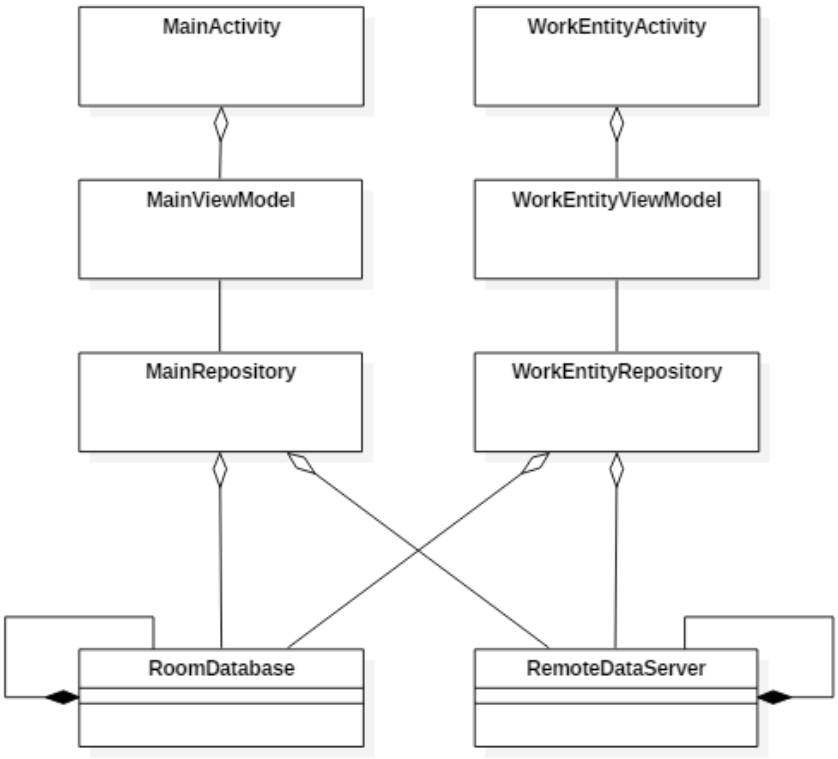


FIGURE 5. Architecture components

Repositories are responsible for data manipulations such as storing, receiving, sending and processing functions. They can manage different sources as the application can operate with local storage or a remote server.

ViewModels are designed as a bridge that bonds User Interface and Data Management. It is an explicitly designed class that receives and stores data from data drivers. In the program, it is linked through a special command to the activity and whenever one survives configuration change, all the data that it presents is retrieved from the ViewModel. It is also possible to link a single ViewModel to numerous activities or fragments in order to provide a communication bridge taking the place of more structurally complex mechanisms as interfaces and callbacks (Fujiwara 2017).

LiveData is a special data holder class that allows linkage between UI elements and a data. Each time the data that the holder is related to changes, a notification is sent to all the linked observers. The holder is aware of the life cycle of the linked UI elements and can therefore adapt. If a UI element is destroyed, LiveData would still continue sending updated information to the element so when it is recreated it will receive the most recent update. This allows flawless changes in application to take place without a concern of data loss or crashes (Android Developers Documentation 2019).

Room Persistency is a library for database manipulation based on SQLite, it includes several helpful features such as applying SQL commands to the database schema at a compile which allows to fix all the issues without facing them at a runtime. Moreover, all the operations are executed asynchronously which allows for the main thread to run independently and not crash application in case of issues related to accessing the database or the execution on the UI thread. It is designed to be operated in a simplified way by embedding the database scheme construction into class definitions for further operations on DB with class defined functions, making this approach more straight forward in an object-based sense. Additionally, this technology also makes it simpler to provide support for ViewModels and LiveData technologies as it can notify the holder if the data inside the database has changed (Android Developers Documentation 2019).

6 Application Development

This chapter covers explanations on different aspects of development strategies and describes certain complications faced during engineering process. Moreover, a brief listing of the applications and frameworks that were used in the application is provided.

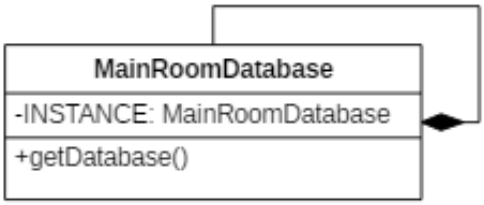
6.1 Tools

The project was built with an Android Studio version 3.2.2, an official integrated development environment, designed by Google and JetBrains, for the purpose of building solutions for Android operating system. The solution features backwards compatibility allowing to run on devices with minimum SDK version of 21. The dependencies included are Room Persistency Library version 2.0.0-beta01, Lifecycle components version 2.0.0-beta01, Material Design version 1.0.0-beta01 and AppCompat Library version 1.0.0-beta01. All aforementioned tools are related to the AndroidX Extension library.

6.2 Architectural Design Solutions

Architecture components were used as a main solution for data manipulations in the application. All the operations on database were designed to run asynchronously, so that when User Interface requests data to display, UI thread would not have to wait for operations to be performed and would rather then be notified when data is available to display it. This allows for User Interface and data manipulations to work in parallel which not only results in better performance, but also provides the user with a better experience due to lack of UI freezes.

Main Room database class is designed as a singleton to ensure that only one connection to the active database is established at a time (GRAPH 8). The class contains a single instance of the itself and when a function to get database is executed it checks whether the connection is established and otherwise creates one.



GRAPH 8. Class diagram – Singleton database access class

The main recycler view which displays the user list of entered Entities is configured with two separate view holders that differ between each other in one single detail, that one provides a storage for image. When an entity is processed it is validated whether it contains an image and therefore assigned with a proper View Holder. That solution decreases the amount of memory used during the runtime of the application, as image holders are resource-heavy UI elements.

The main application activity contains two fragments, at each point only one fragment is displayed. On the start of the application two fragments are initiated and until further configuration changes, both are kept in the memory of the device. This allows for faster UI interaction comparing to the solution with multiple activities.

An approach featuring interface callbacks was used in both Recycler View Adapters to request fragments, applying them, for further actions on data provided. In case of Entities the RV adapter provides a possibility to select multiple entities and provides a list of them to the fragment. Categories RV callback returns a single selected category entry.

Operations on searching and sorting were categorized by the simplicity factor. If an operation would require only a single parameter searching or sorting then a specially designed Query would be executed. If an operation would require both searching and sorting by multiple parameters then a Query would be created and carried out on the runtime of the application. The explained operations could be managed internally by modification of Adapter classes but this approach would require a greater amount of application rework if additional changes would be required to be applied in the future.

Images were stored in the internal memory of the application to provide the user with a possibility to access them with other applications on the device for any sharing, viewing and other purposes. Concerning the case of application uninstallation, all the data would remain on the device.

7 SOFTWARE TESTING

In order to ensure proper application behaviour, functionality and integrity, it is run thorough certain tests. The results of the tests are evaluated by overall successfulness of execution and by accordance to the expected output. Experiments can be carried out on the normal flow of the application or can be forced to stress out the limitations of the solution. For the later product development and upgrade specific test are usually performed on every feature that is integrated (LaTonya 2015).

7.1 Test Results

Test results are presented below in table 3, each case contains a name and states if the execution was carried out according to the expected scenario. If the test progressed with some malfunctions or inconsistency a remark would be left at column “Comment”.

TABLE 3. Test results

Action	Expected Behaviour	Test Result	Comment
Creation of Category	Category is created, added to the database and displayed.	Successful	
Creation of Entity	Entity is created, added to the database and displayed.	Successful	
Creation of Entity with Image	Entity is created, if image is included, it is added to device storage and displayed. A proper adapter View Holder and card layout is displayed.	Successful	

Action	Expected Behaviour	Test Result	Comment
Statistics on categories	Statistic on every category, including number of entities, last day of Modification.	Successful	
Searching by keyword	Entities are searched for matching keyword in either Title or Content of Entity.	Successful	
Sorting by Date of modification	Entities are sorted by latest modified first.	Successful	
Sorting by Date of creation	Entities are sorted by date of creation, latest first.	Successful	
Sorting by Entities that include Image	Only Entities containing images are displayed.	Successful	
Sorting by type Special	Only Entities of type special are displayed.	Successful	
Sorting by category	Only Entities that belong to selected category are displayed.	Successful	
Creation of 50 entities with Images	All entities are created and displayed to user.	Successful	Application does not include image processing and optimization therefore all correlated operations on entities with images have a delay of approximately one second independently on number of Entities with images after count of 20 and more of a kind.

7.2 Release notes

Studia provides a simple and elegant solution for note keeping. The user can perform any operations on searching and sorting or simply review previously input entities. Flexible statistics will provide a current status on development in any category of choice. No issues or complications were found to be noticed in the current version of the product.

8 FURTHER DEVELOPEMENT

Several features are considered for development or improvement in later versions of the application. Statistics shall be developed to be able to construct and display graphs to present a more vivid analysis on application usage. A better image processing algorithm should be designed to optimize the response time of the application. It should be possible for a user to include more than one image to Entity and therefore a better interface solution, as an image viewer that provides a scrolling behaviour if multiple images are included, should be implemented.

REFERENCES

Android Architecture Components. Android Developers Documentation. 2019. Available: https://developer.android.com/topic/libraries/architecture. Accessed: 20 March 2019.

Flutter Introduction. Flutter. 2019. Available: https://flutter.dev/. Accessed: 01 June 2019.

Fujiwara, L. 2017. View Models: A Simple Example. Available: https://medium.com/androiddevelopers/viewmodels-a-simple-example-ed5ac416317e. Accessed: 23 May 2019.

Functional and Nonfunctional Requirements: Specification and Types. Altexsoft. 2018. Available: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/. Accessed: 31 May 2019.

Haase, C. 2019. Google I/O 2019: Empowering developers to build the best experiences on Android + Play. Available: https://android-developers.googleblog.com/2019/05/google-io-2019-empowering-developers-to-build-experiences-on-Android-Play.html. Accessed: 31 May 2019.

LaTonya, P. 2015. The Four Levels of Software Testing. Available: https://www.seguetech.com/the-four-levels-of-software-testing/ Accessed: 19 May 2019.

LiveData Overview. Android Developers Documentation. 2019. Available: https://developer.android.com/topic/libraries/architecture/livedata. Accessed: 23 May 2019.

Lotz, M. 2018. Waterfall vs. Agile: Which is the Right Development Methodology for Your Project?. Available: https://www.seguetech.com/waterfall-vs-agile-methodology/ Accessed: 20 March 2019.

Material Design Foundation. Material Design adaptive system of guidelines. 2019. Avaliable: https://material.io/design/foundation-overview. Accessed: 23 May 2019.

Mew, K. 2015. Learning Material Design: master Material Design and create beautiful, animated interfaces for mobile and web applications. Birmingham: Packt Publishing.

Room Persistency Library. Android Developers Documentation. 2019. Available: https://developer.android.com/topic/libraries/architecture/room.html. Accessed: 23 May 2019.

Sommerville, I. 2016. Software engineering. Harlow Singapore: Pearson.