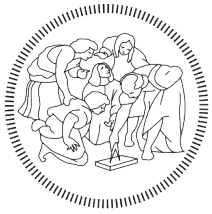


POLITECNICO
MILANO 1863

Notes Bites

Software Design Document

Filippo Antonielli, William Bonvini

April 8, 2021

NotesBites

Notes Bites Design Document
Version v1.0, September 2020

Contents

1 Introduction

This chapter gives an overview of what this document is meant for and provides the reasons of the two most important choices we have made (the platform and the application's purpose).

1.1 Document's Goal

The goal of this document is to show, in the specified order, the following information

1. the platform that has been chosen to develop the application with and the motivations behind it
2. the purpose and features of the application Notes-Bites
3. the requirements that such application satisfies (both functional and non functional)
4. the design choices (both architectural and visual) we have made and the motivation behind them
5. the external services we have used and the rationale behind it
6. a series of UML diagram to show the interactions between classes (Class Diagram), interactions between the application and the users (Sequence Diagrams), and possible scenarios of usage (Use Case Diagrams)

1.2 Choice of the Platform

We have decided to develop the application with Android, since both of us have experience with Java. Moreover we would like to reach the highest number of users as possible and Android is the operating system with the highest percentage of users.

1.3 Choice of the Application

As we'll describe in the next chapter, Notes-Bites is an application that gathers university notes on Machine Learning and Internet of Things related courses. We have chosen to write an application for university students, since we have struggled in the past to find reliable and clear notes in the topics we are most interested in. We would like to create a community in which students can learn both the theoretical and practical aspects of specific courses. Moreover, we want to offer to the users the possibility to test their understanding of the subjects by doing several quizzes.

2 Overall Description

2.1 Goal of the Application

The goal of Notes-Bites is to gather university notes on Machine Learning and IoT related courses. Our goal is to create a platform that provides students with all they need in order to fully understand difficult concepts and to face the associated exams.

2.2 Core Features

Each course is divided into modules, and each module contains a theoretical explanation and a related quiz that the user can do to test its understanding. The results of the quizzes are stored within the internal database in such a way that the user can track their progress. Some module may contain also a YouTube link in order to deepen the topics of the module.

2.3 Product Functions - Functional Requirements

The software will offer the following main features:

- **Access to list of courses**
  users need to choose the subjects they are interested in studying. Once they tap on a subject, this one will be highlighted. By tapping on the info button next to each subjects an overview of the course will be presented.
- **Overview of each Subject**
  the user, while is deciding which subjects is interested in, should be able to see an overview of the modules that are covered by the app, and a small summary of the content.
- **Registration and Log in**
  users can decide to authenticate themselves by means of a Google account. Once they are authenticated they will be able to see the homepage for the logged users.
- **Access to list of modules for each course**
  users by tapping on any subject in the homepage can visualize the list of all the modules for that course. If a user has logged in, the most recent modules for each subject will be displayed in the homepage.
- **Access to modules content page**
  once a module is selected by the user, he can decide to visualize the markdown document with the notes of that module or to take the quiz. In some module a YouTube link can be present to deepen the topic of the module.
- **Access to module readable content**
  the user need to be able to see the markdown document related to the selected module.

- **Access to module video content**
  in some modules can be present a YouTube button that will redirect the user to the YouTube application.
- **Access to quizzes**
  users need to be able to access the quiz corresponding to the selected module
- **Completion of quizzes**
  users need to be able to interact with the app in order to answer questions and get feedback on the correctness of the answer they selected
- **Save results of quizzes**
  the results of a quiz have to be saved within a database in order for the user to retrieve such information and improve his score.
- **More than one quiz for a specific module**
  create a database structure that takes into consideration the possibility of adding more quizzes for a specific module

2.4 Non Functional Requirements

- **Mantainability**
  Notes-Bites must be easily maintainable. This translates in the decoupling of the components of the system and in clearly commented code.
- **Portability**
  Notes-Bites must work for all Android's updates from Android 6.0 on.
- **Extensibility**
  the application needs to be programmed taking into consideration possible future functionality extensions. This translates into making it simple to programmers to add modules of code without impacting previously developed functionalities.

2.5 User Characteristics and Actors

Guest: a person who is not logged in the application. He is able to use the main functions of the application such as visualize the notes and access to the quizzes, but cannot access the Homepage for Users and see the recent modules per each subject.
User: a person that has logged in the application. He is able to access all the components of the application.
user: with user we refer to a person which can be a User or simply a Guest.

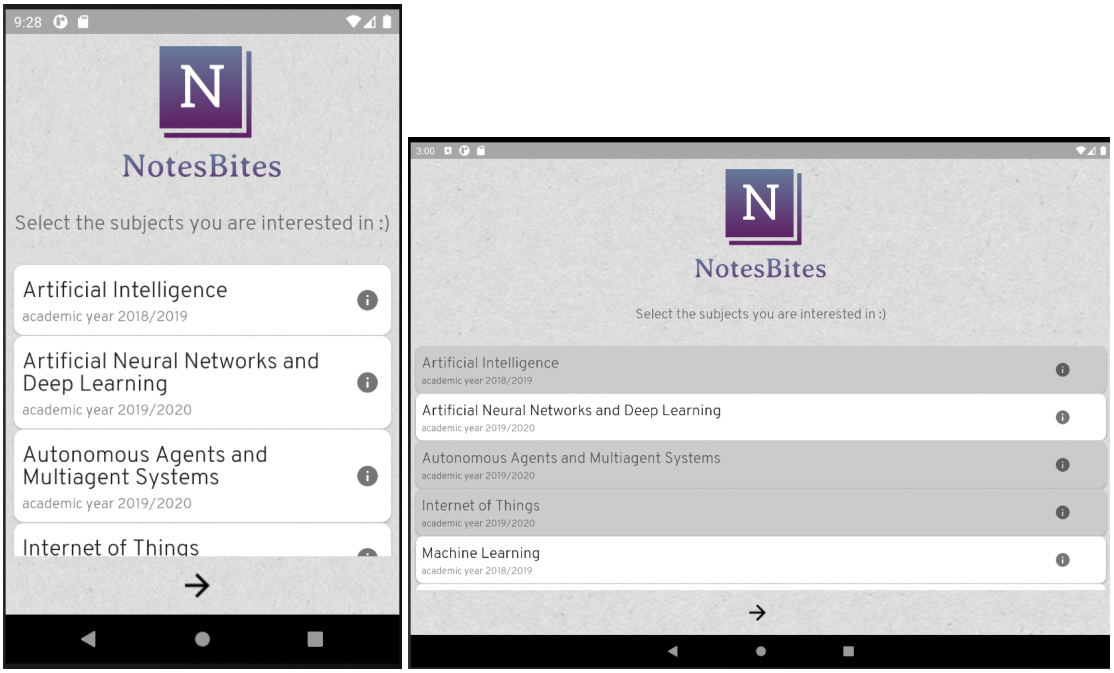
2.6 Constraints

The minimum SDK required by NotesBites is API 19, which corresponds to Android 4.4 KitKat.
This means that whoever uses an older version of Android won't be able to download the app from the Play Store. Just a very small percentage of Android users will be penalized, and it is highly unlikely that our target audience will incur in this problem.

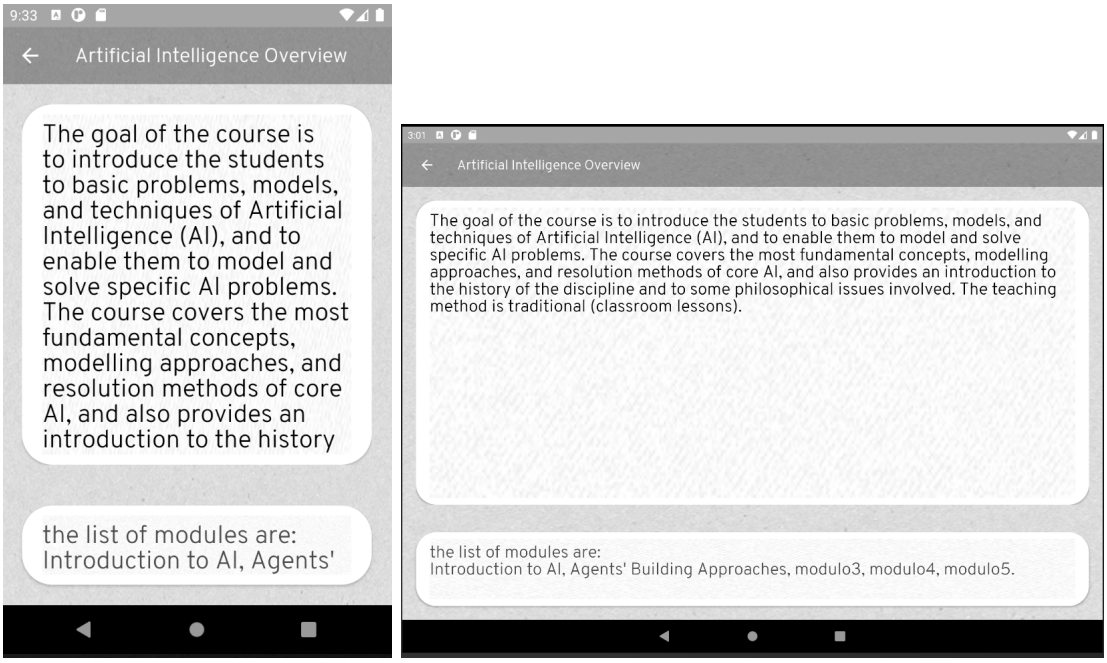
3 General Overview

3.1 User Interface

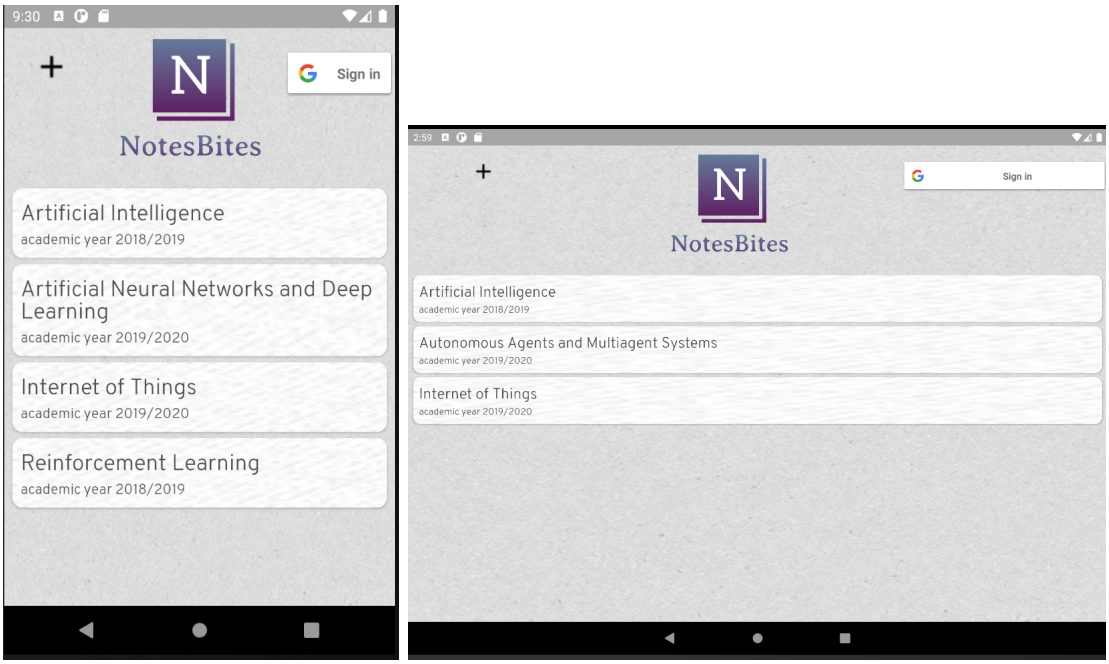
This section is meant to provide a general idea of how the mobile application should look like.
As a first step a Guest is prompted an interface and he is able to choose the subjects in which he is interested in studying.



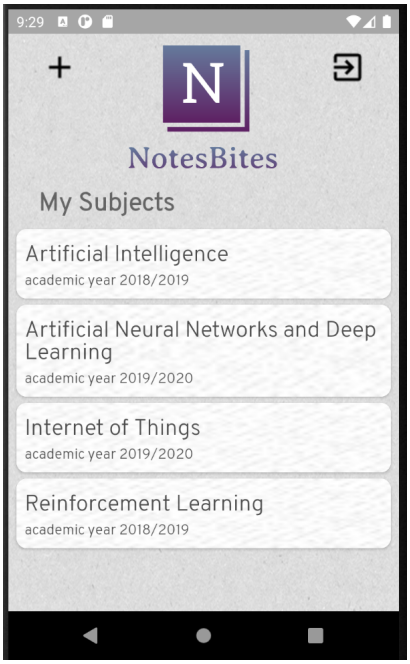
By tapping on the info button of the subjects a new page will be shown with a short and discursive overview on the material we are offering for that specific subject.

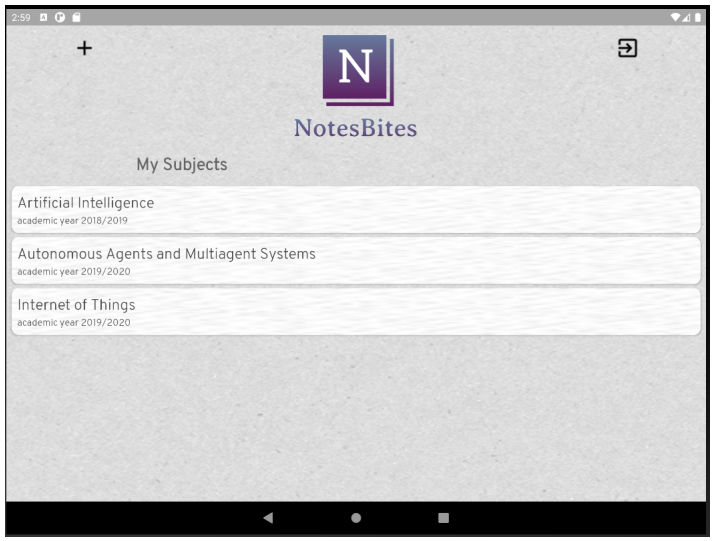


Once the Guest has selected as many subjects as he is interested into, he will see a personalized homepage with the subjects that have been selected in the previous page. This is the GuestHomepage.

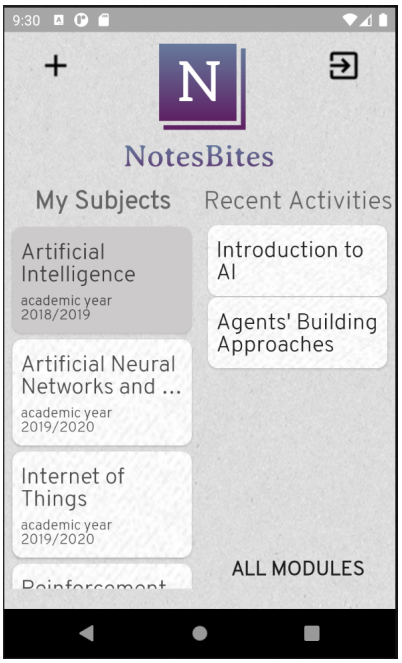


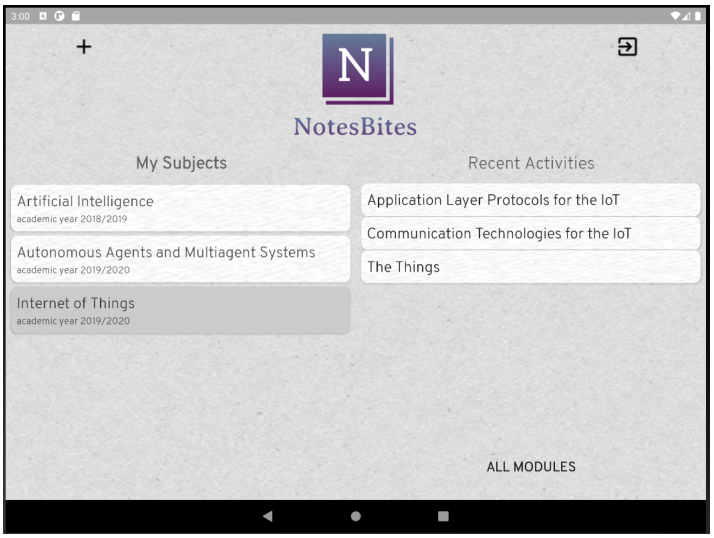
From the GuestHomepage the Guest can already start to visualize and study the material provided with each subject, but our suggestion is to perform the login in order to unlock more free functionalities. Once the login is performed the User will see another personalized homepage.



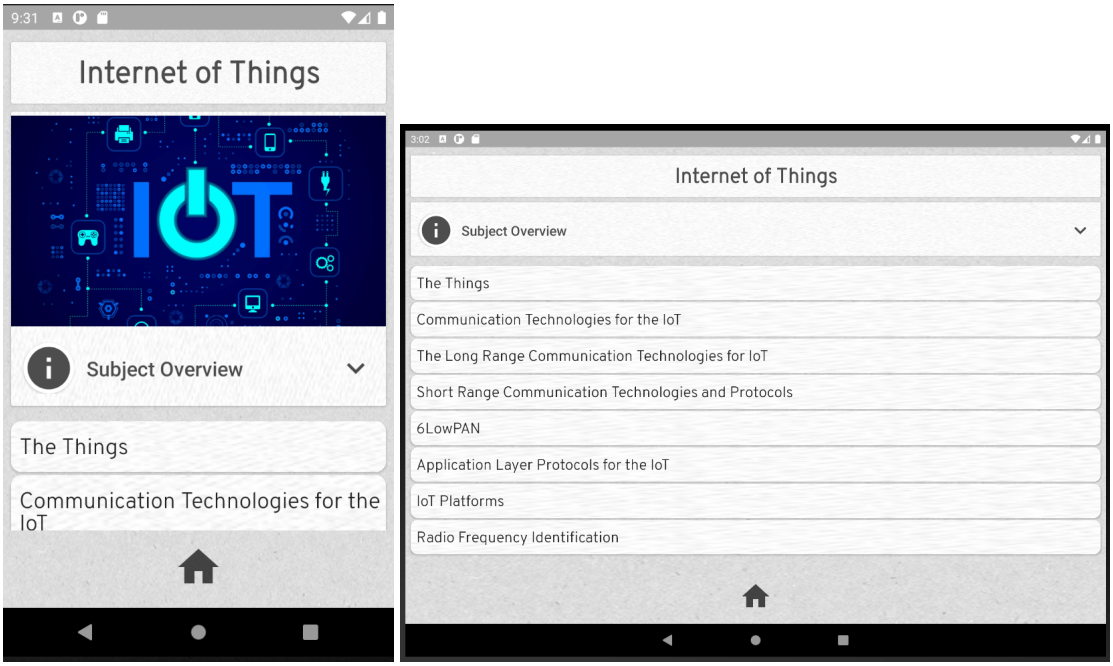


If the User clicks on a subject, given that he logged in, he can see the recent visited modules of that subject, in order to resume immediately from where he has interrupted.

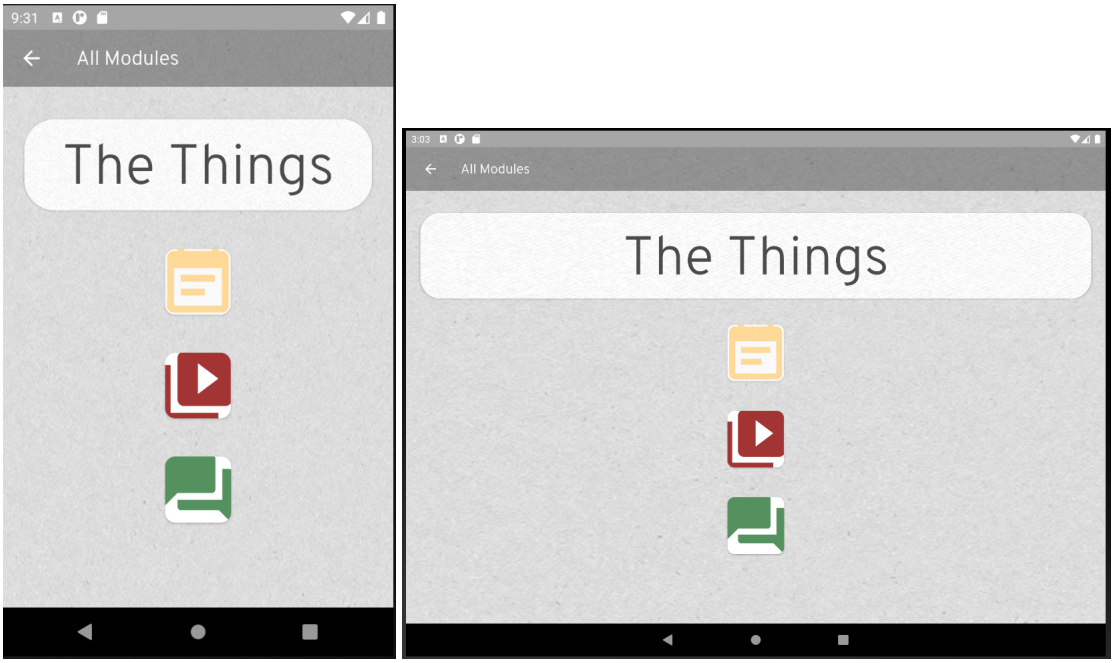




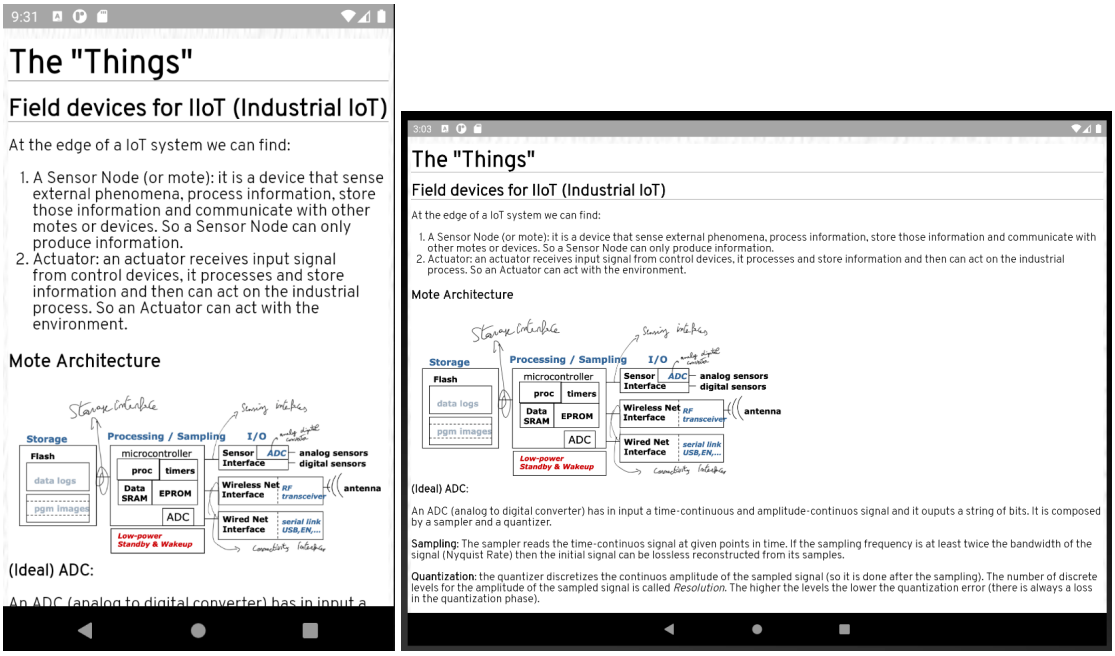
By tapping on a module the User will reach the page relative to the specific module. Instead by clicking on ALL MODULES he will reach the Subject Overview page.



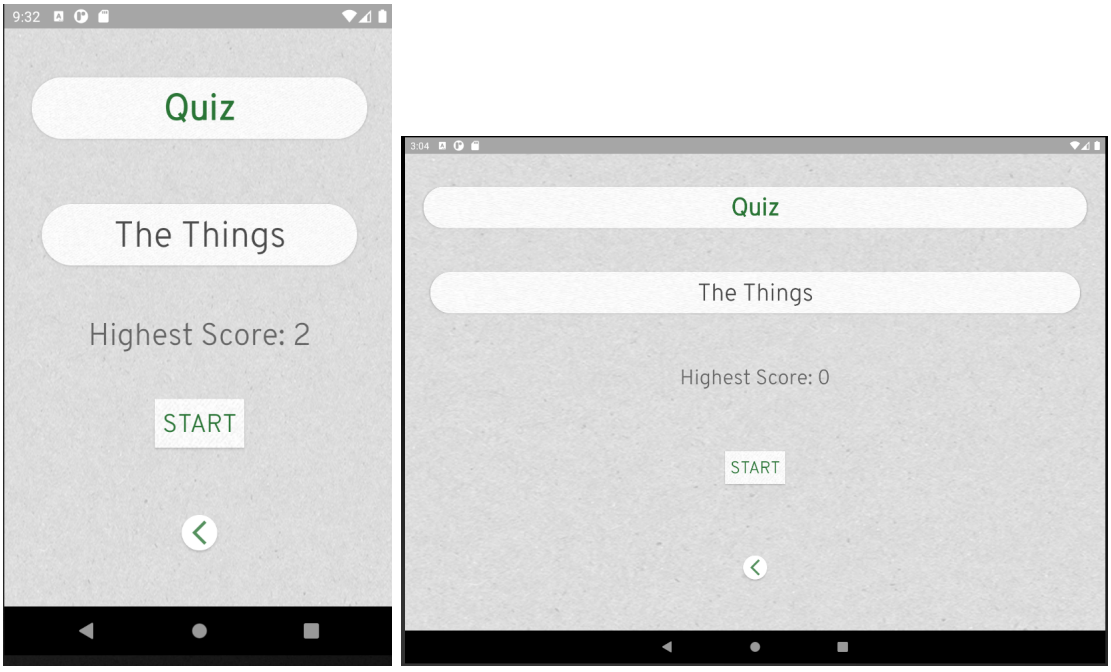
From this page it is possible to visualize the brief overview of the subject by tapping on the drop down menu, or by clicking on a module he will reach the Module page



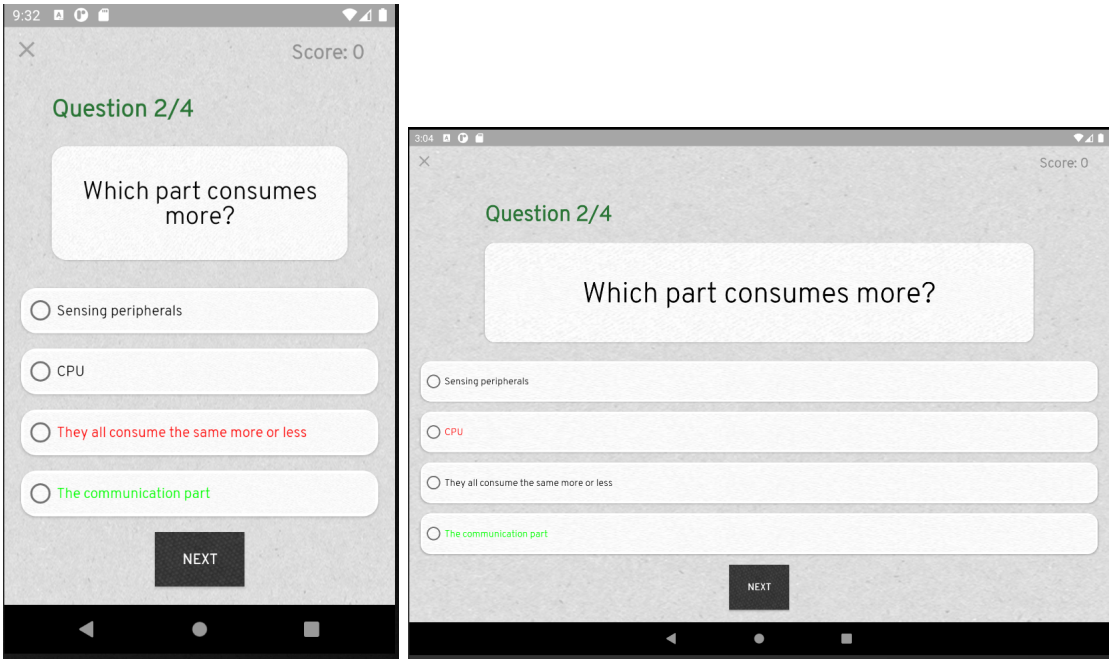
From this page it is possible to access to the markdown notes regarding the module,



click to the video button to be redirected to YouTube and watch a video to deepen the topic of the module, or click on the green button to access the quiz



From the Starting Quiz page, if the user taps on the START button the quiz is started. The number of questions are not fixed and depend on the module selected. The user can also visualize when he is right or wrong regarding a question, and see the total points at the end of the Quiz.

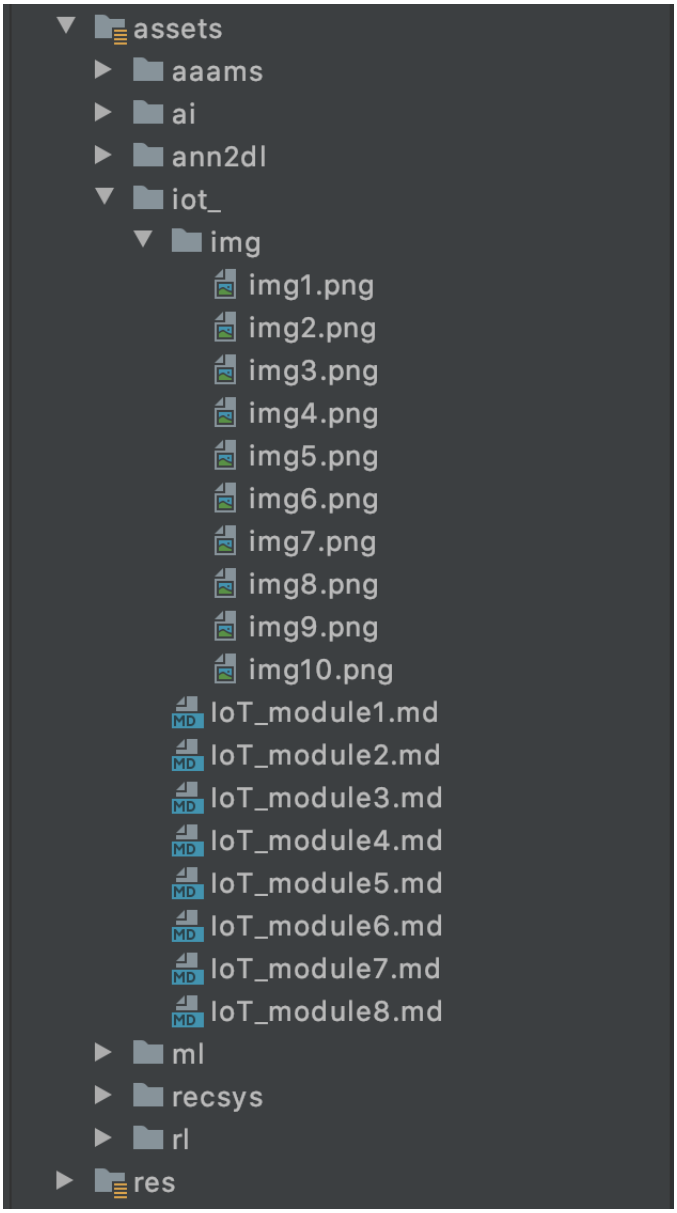


3.2 Markdown Parsing library

In order to display the markdown files we have written in these years in an Android app, we looked for a Markdown parsing library that could handle all the features we were interested in. We needed support for images, inline and full line LaTeX formulas, and html tags.
We found many libraries but most of them did not support the functionalities we requested. Moreover, most of them were not maintained anymore, which translated into the possibility of our app not working properly in future Android releases.
After some research we found Markwon, a maintained Markdown parsing library for Android with many features supported and more to come. Markwon parses markdown following commonmark spec with the help of commonmark-java library and renders result as Android-native Spannables (an interface for text to which markup objects can be attached and detached).
No HTML is involved as an intermediate step. No WebView is required. It's extremely fast, feature-rich and extensible.
It gives ability to display markdown in all TextView widgets (TextView, Button, Switch, CheckBox, etc), Toasts and all other places that accept Spanned content. Library provides reasonable defaults to display style of a markdown content but also gives all the means to tweak the appearance if desired. All markdown features listed in commonmark spec are supported (including support for inlined/block HTML code, markdown tables, images and syntax highlight).

3.3 Markdown Location

In NotesBites the markdown files are under the assets folder in Android Studio.



When a new module of a subject is added, in the mdContent it is writen the path from the assets folder where the markdown file is situated, for istance if I have to add the eight module of Internet of Things, in the mdContent I have to write "iot_/IoT_module8.md"

4 Architectures

4.1 High Level System Architecture

The software architecture of NotesBites has been developed following the best practices recommended by the Android Official Documentation. We relied heavily on the Model-View-ViewModel architecture.
The view model of MVVM is a value converter, meaning that the ViewModel is responsible for exposing (converting) the data objects from the model in such a way that objects are easily managed and presented. Given that in future versions of the app we will want the app to communicate with an external database in order to retrieve data, we used the repository pattern in order to abstract the View and the ViewModel from the data source (this way providing a single source of truth). We will explain in depth each component in the following sections.

4.1.1 Model-View-ViewModel Architecture

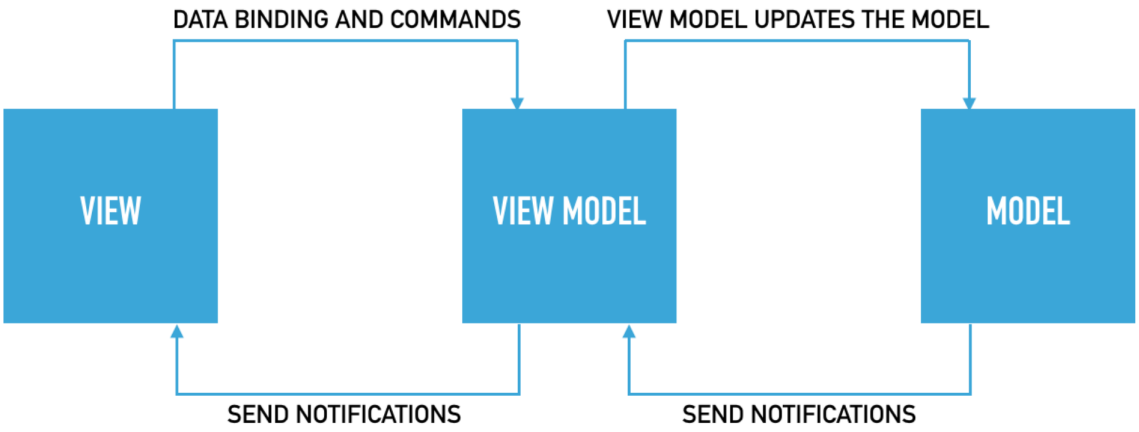
MVVM is one of the architectural patterns which enhances separation of concerns, it allows separating the user interface logic from the business (or the back-end) logic. Its target (with other MVC patterns goal) is to keep UI code simple and free of app logic in order to make it easier to manage.
As the name suggests it is made of the following three elements:

- **Model**
  The Model represents the data and business logic of the app. It is the component of the app that retrieves data from the different data sources and make them available to be displayed.
  As we will discuss later, the data of NotesBites are stored within an internal database and the app assets, and are passed to the view through a repository pattern in order to provide a single source of truth.

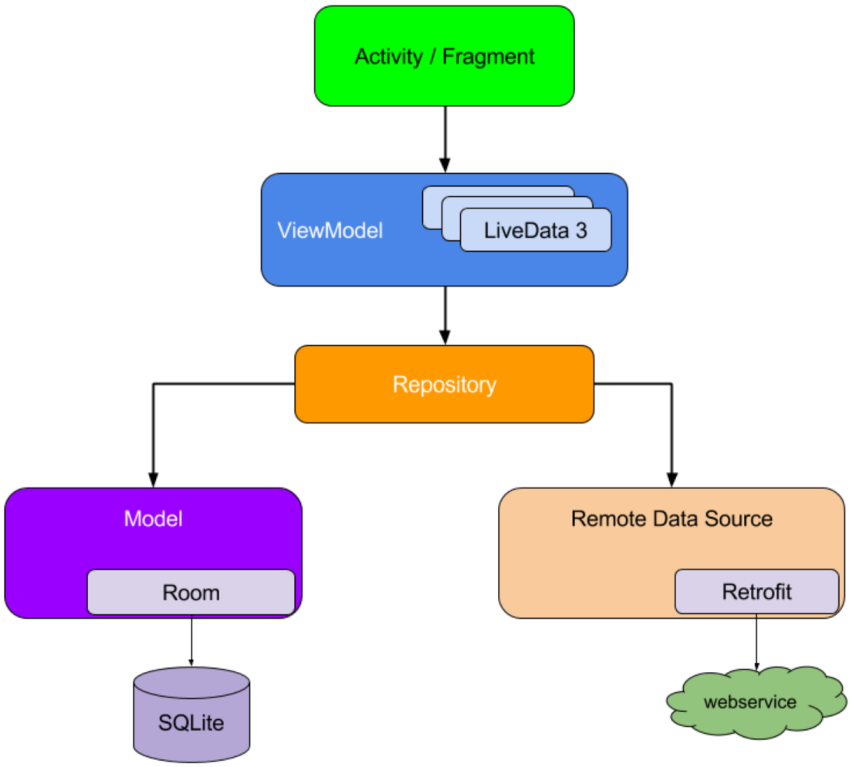
- **ViewModel**
  The ViewModel interacts with the Model and also prepares observable(s) that can be observed by a View. A ViewModel can provide hooks for the View to pass events to the model. In Android app development ViewModels are particularly useful because they store the state of a particular activity (in fact usually there is a ViewModel instance for each Activity) in such a way that if there is a configuration change and the activity gets destroyed, the information saved within it gets restored. One of the important implementation strategies of this layer is the decoupling from the View, i.e, ViewModel should not be aware about the View instance it is interacting with.

- **View**
  Finally, the View's role in this pattern is to observe (or subscribe to) a ViewModel observable to get data in order to update UI elements accordingly. The View is made of the activity classes and the correspondent xml layouts and fragments.

Here follow a schematic representation of the high level MVVM pattern:



The image is highlighting the fact that direct communication among classes goes only in one direction (from the View classes to the ViewModel classes and from the ViewModel classes to the model classes), while notification of changes in data goes in the opposite direction (whenever a tuple in the database gets updated, a notification is sent to whoever is listening/needs the updated data).
NotesBites exploits an expanded version of the pattern above, that is recommendend by the Official Android Documentation and is summed up in the following diagram:



The elements that have been introduced in the diagram above are the following:

1. **LiveData**
   The diagram above is telling us that the data that is observed by the activities/fragments

are instances of LiveData objects. LiveData is a class that ensures that the app will only trigger app component observers that are in an active lifecycle state.
LiveData is an observable data holder class. Unlike a regular observable, LiveData is lifecycle-aware, meaning it respects the lifecycle of other app components, such as activities, fragments, or services.
We will talk in depth about this class in the next chapter.

2. **Repository**
   As can be noted by the diagram, a ViewModel delegates the data-fetching process to a new module, a repository. Repository modules handle data operations. They provide a clean API so that the ViewModel can retrieve data easily, with no need of knowing who is the source of such data. the ViewModel needs only to know what data it wants to retrieve and the API calls to make whenever data is updated. You can consider repositories to be mediators between different data sources, such as persistent models, web services, and caches. We personally chose to implement the repository module because in future implementations we will give the possibility to the user to decide what subjects to download on his smartphone and what not.
   the Repository will handle
   a) the retrieval of data from the local sqlite database to be displayed in the view
   b) the loading of data from the external server to the local sqlite database
   c) the direct retrieval of data from the external database
3. **Retrofit**
   Retrofit is a type-safe HTTP client for Android and Java. In a future implementation we count on relying on it to retrieve subjects' and users' data in the form of a JSON (Retrofit makes it easy to parse JSONs into Plain Old Java Objects, namely POJOs).

4.1.2 Libraries & Best Practices

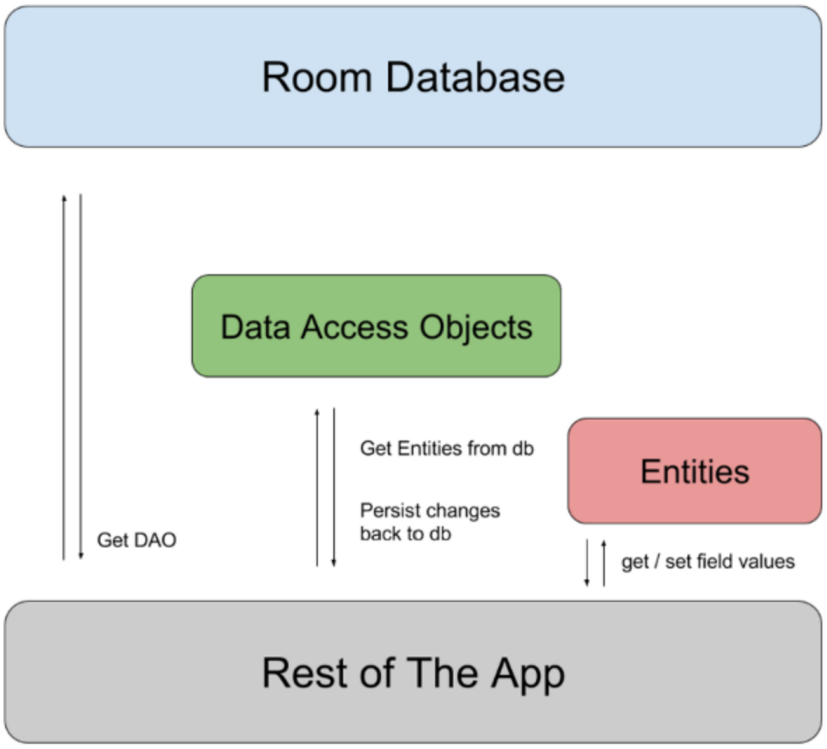
Room

For the data layer we relied on the the Room persistence library as suggested by the Android Development Community.
Room provides an abstraction layer over SQLite to allow for more robust database access while harnessing the full power of SQLite.
The library helps handling the data stored within the SQLite database that drives the app. This library allows users to view a consistent copy of key information within the app.

There are three major components in Room

1. **Database**
   The component that handles the population of the SQLite database.
   In NotesBites this component is named *NBDatabase*
2. **DAO**
   DAOs are the main component of Room and are responsible for defining the methods that access the database. All queries are written in DAO only.
   In NotesBites, we created a DAO for each group of entities that refer to the same context. We will talk about them in depth in the low level architecture section.
3. **Entity**
   This component represents a database table. Each field of the entity is persisted in the database. In NotesBites we created 4 entities: Subject, Module, Quiz, and QuizQuestion.

Here follows a diagram that shows the interaction among the various Room modules.



Advantages of Room:

1. compile-time verification of raw SQL queries: the developer gets noticed whenever they write a wrong column/table name in a query.
2. there is no need to write a lot of boilerplate code to convert between SQL queries and Java data objects
3. Room helps avoiding issuing database queries on the main thread (as it can cause delays on the main thread), through the usages of DAOs and LiveData.
4. It makes your code more modular and readable as you divide your database code in three main components: Entity, DAOs, and Database.

4.1.3 LiveData with Room

LiveData is a lifecycle-aware observable data holder class.
Unlike a regular observable, LiveData is lifecycle-aware. This awareness ensures LiveData only updates app component observers that are in an active lifecycle state.
it's common practice to use together Room and Livedata because Room is able to generate all the necessary code to update the LiveData object when a database is updated.
The generated code runs the query asynchronously on a background thread when needed.
The advantages of using LiveData

1. **No manual lifecycle handling**
2. **Proper configuration changes**
   If an activity or fragment is recreated due to a configuration change, like device rotation, it immediately receives the latest available data.

3. **Ensures your UI matches your data state**
   Instead of updating the UI every time the app data changes, your observer can update the UI every time there's a change.

4. **No memory leaks**
   Observers are bound to Lifecycle objects and clean up after themselves when their associated lifecycle is destroyed.

5. **No crashes due to stopped activities**
   If the observer's lifecycle is inactive, such as in the case of an activity in the back stack, then it doesn't receive any LiveData events.

4.2 Low Level System Architecture

In this section we will describe how we applied the principles described in the section above (high level architecture) to the NotesBites app. In particular we will give a summary of the type of Java classes we defined and their role.
for a visual representation look at the UML class diagram reported below or in the UML Diagrams chapter.
We divided the project structure in two layers

1. Presentation Layer
   layer that contains the classes related to display data (the logic layer is mostly incorporated here)

2. Data Layer
   layer that contains the data and the classes to retrieve and update them

4.2.1 Presentation Layer

folder structure for the presentation layer
One package for each screen.
List of application screens:

- GuestHomepage
  the homepage the user is directed to if he has not logged in.

- Homepage
  the homepage the logged user is directed to.

- Module
  the screen that refers to the content of a particular lesson (contains link to markdown, YouTube video, and quiz)

- Notes
  the screen that displays the markdown of a particular module

- SelectSubjects
  the screen from which the user chooses the subjects to be displayed in the homepage

- SimpleSubjectOverview
  the screen that contains the description of a particular subject's content and the list of the names of the modules for such subject.

- StartingQuiz
  the page where the user can start the quiz or return to the Subject Overview page

- Quiz
  the screen where are displayed the questions and the answers
- SubjectOverview
  the page in which the user can see the list of modules for the selected app and access to any of those

for each of these screens we have at least an Activity class (to render the layout) and a ViewModel class (to hold the data to be rendered).

4.2.2 Data Layer

folder structure in the data layer:

- daos
- local
- model
- repository
- utils

Repository
We have one repository for each app context.
We defined three app contexts:

1. the list of subjects the user has selected to get displayed
2. the list of modules for each subject selected by the user the quiz context, since the data to be retrieved to show it to the user are contained in two tables that are almost independent to the rest of the database ("quiz" table and "quiz_question" table).

The actual repositories are called:

- SubjectsRepository
- ModuleRepository
- QuizRepository

Model
contains the classes that are used by Room to create the structure of the tables in the SQLite database (each class corresponds to the tuple structure of a specific table). the classes in the model are:

- Module
- Subject
- Quiz
- QuizQuestion

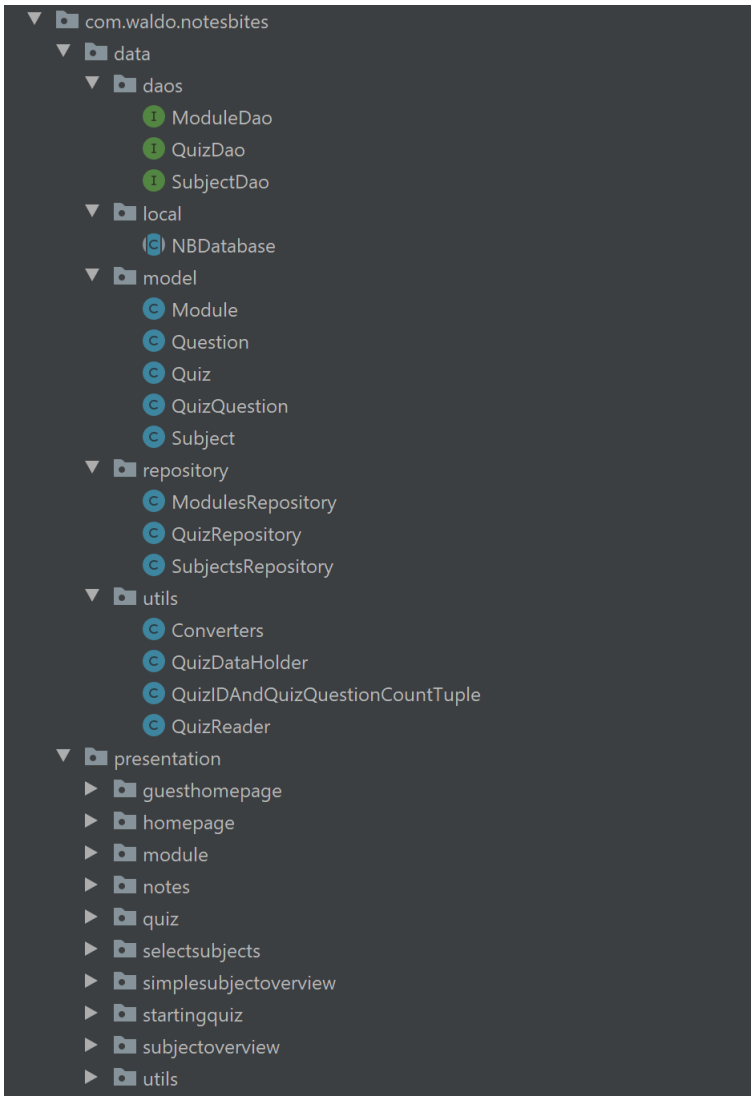
Local
this folder holds the reference to the local database. So far the app runs exclusively on the local database, but in future releases it will retrieve the data from an external database and cache the needed data in the local one.
DAOs This package contains the Data Access Objects, one for each context, as in the repository case. This means we have 3 DAOs, namely:

- SubjectDao
- ModuleDao
- QuizDao

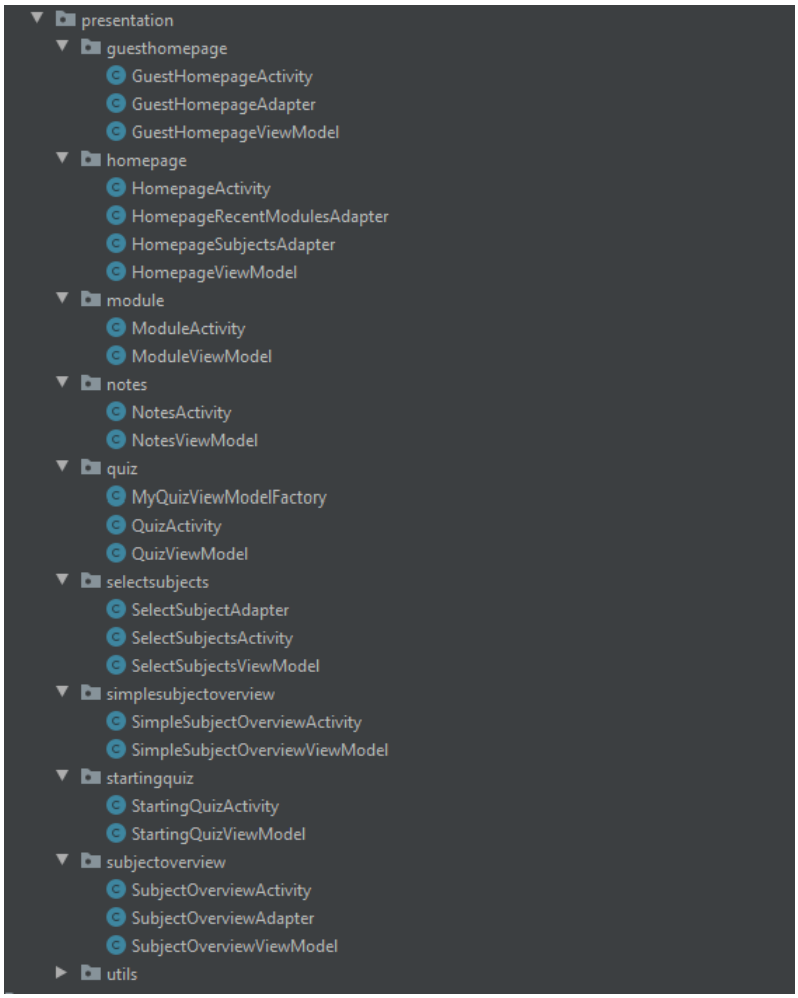
4.2.3 Java Package Organization

Here follows the actual package organization as described in the previous section.
As it can be noted, the Dao elements are not Java classes but Interfaces because it is required by the Room Library (we will discuss in depth about it in the next chapter).
This the package organization we came up with, and think it helps clarifying the role of each class.

Data Layer Package Organization



Presentation Layer Package Organization

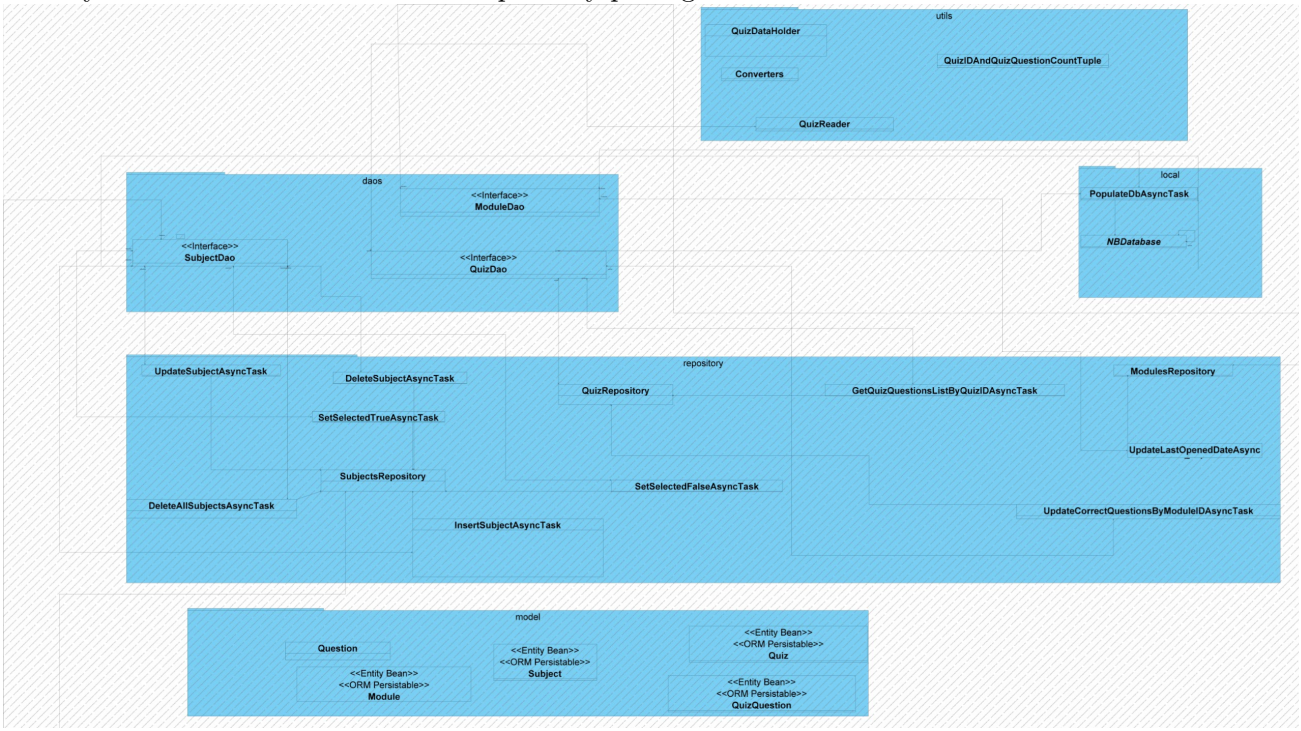


4.2.4 UML Class Diagram

For a better understanding of the division of the classes within our application, we will show the class diagram of the two main packages: the data layer and the presentation layer.
The interaction between the two packages happens whenever a ViewModel class interacts with a Repository class.

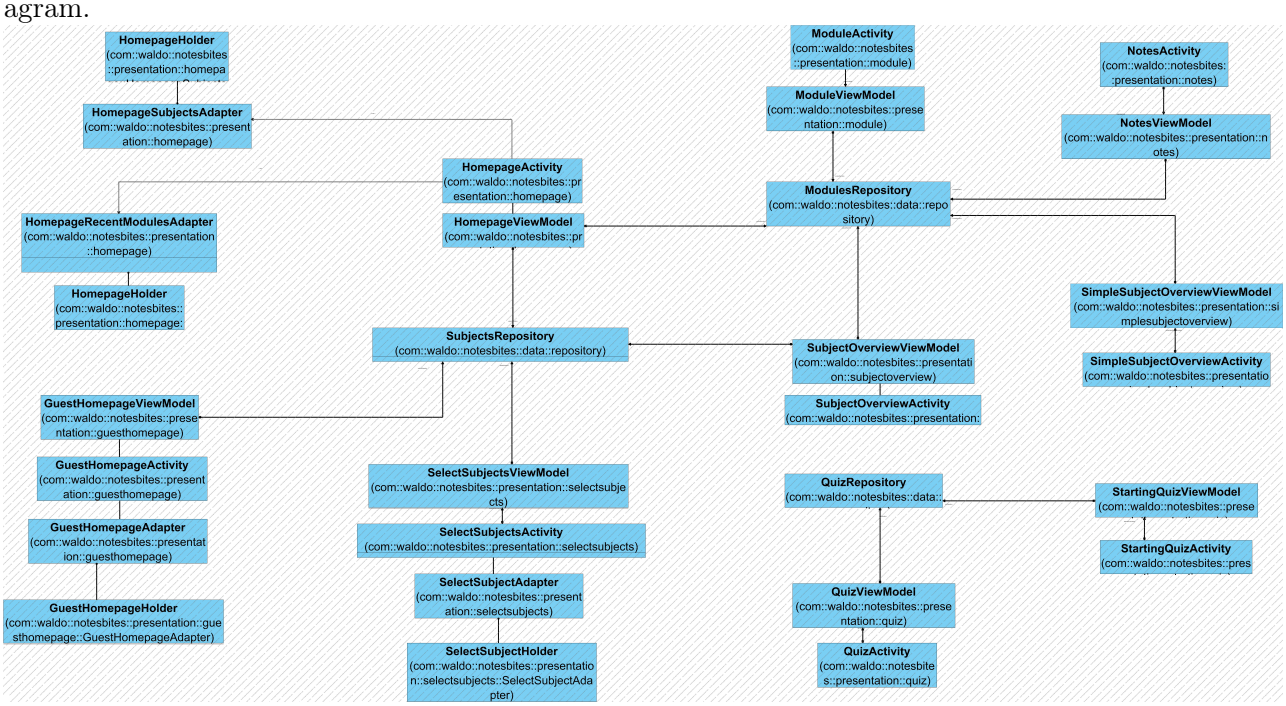
Data Layer Class Diagram

Here follows the classes that are needed for a proper representation of the model and for correctly accessing the database to retrieve data and populate it.
These classes have been divided into subpackages. We grouped together classes with the same role. It's to point out the fact that we created a class for each operation in the database that is to be performed asynchronously. instances of these classes are instantiated by the three classes within the Repository package.



Presentation Layer Class Diagram

To favour the understanding of the interaction of the presentation layer to the class Diagram we have inserted in the diagram below the interaction with the Repository classes, even though they don't belong to the Presentation layer and, consequently, to the class diagram associated with it.
Simple classes like OnSwipeTouchListeners have been omitted for simplification of the diagram.



4.3 External Services

4.3.1 Google Login

In the GuestHomepage we offer the users to Sign In with a Google account. For doing this we followed the official documentation of Google where we configured the Google API Console project. At the current state of the application the Sign In allows the users to access the Homepage instead of the GuestHomepage and when the YouTube button is clicked in the Module activity, the YouTube app will be logged with the same account as the one used in Notesbites.

4.3.2 YouTube API

NotesBites relies on YouTube to display the video content related to subjects' modules. Once the user clicks on the video content button in the screen relative to any subject's module, he is redirected to the YouTube application.

5 Data Design

5.1 Database Design

5.1.1 Data Description

Here follows a description of the strucutre of the local database of the application.
Bold attributes are primary keys of the table, foreign keys are displayed in the Entiry-Relationship diagram.

- **Subject**
  1. **SubjectID**:
     the ID of the specific subject, it is an incremental number (the first subject has ID = 1, the second has ID = 2, and so on)
  2. name:
     the name of the specific subject
  3. description:
     a brief description of the specific subject
  4. imageResourceID:
     the ID of the image corresponding to the subject
  5. overview:
     a small textual overview on the topics treated in the subject
  6. selected:
     a boolean set to true if the subject has been selected by the user

- **Module**
  1. **ModuleID**:
     the ID of the specific module, it is an incremental number (the first module has ID = 1, the second has ID = 2, and so on)
  2. name:
     the name of the module (the name of the topic of the lecture)
  3. description:
     a brief description of the module
  4. priority:
     an integer that imposes the order of appearance of the modules within a subject. if it is a low number it means it is to be displayed before a module with a high number.
  5. mdContent:
     is the path of the markdown file that contains the note of the specific module
  6. videoURL:
     a URL that redirect to the YouTube video regarding the lesson.
  7. belongingSubjectID:
     contains the ID of the subject the module belongs to

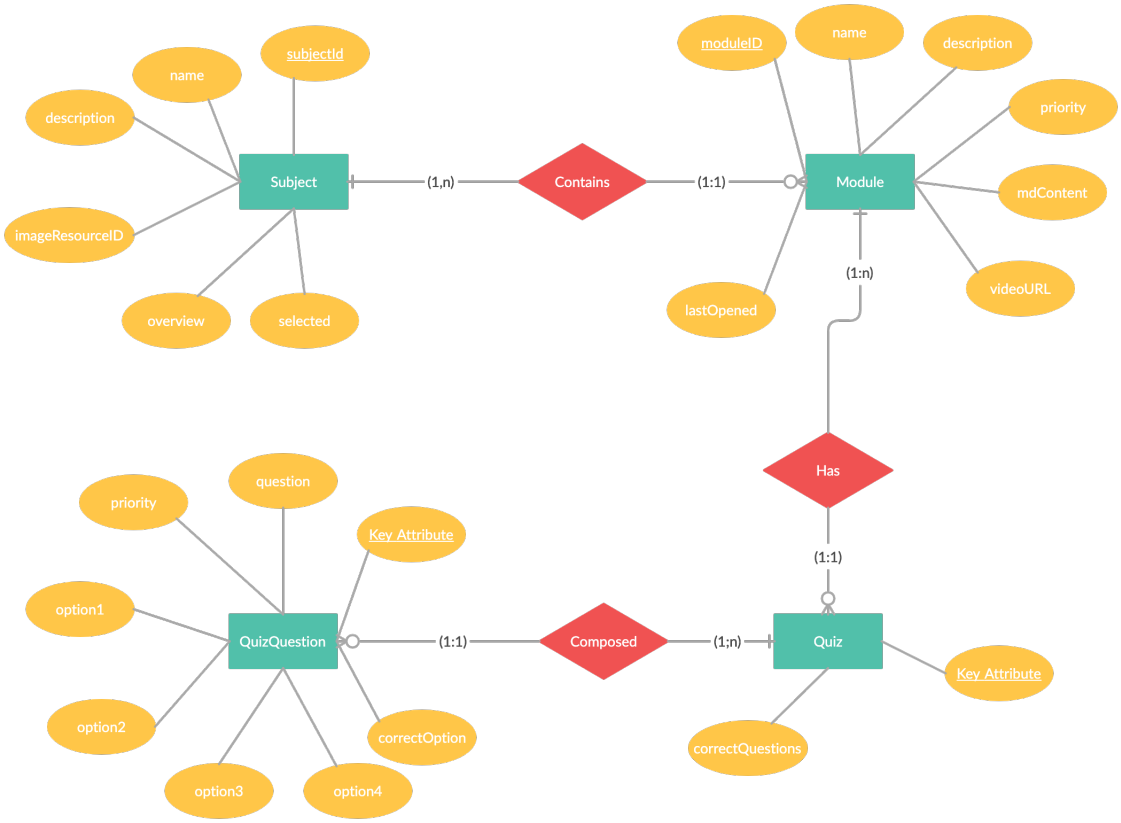
8. lastOpened:
   is of Date type, keeps track of when a module has been accessed last (it is needed to display the recent modules in the Homepage)

- **Quiz**
  1. **quizID**:
     the ID of the quiz for the specific module of the specific subject at hand
  2. belongingModuleID:
     contains the ID of the module the quiz belongs to
  3. correctQuestions:
     keeps track of how many questions are correct when a user takes such quiz

- **QuizQuestion**
  1. **quizQuestionID**:
     the ID of a question regarding the specific quiz it refers to
  2. question:
     the text of the question
  3. priority:
     an integer that tells us what is the order of appearance of the questions within the quiz. a quiz question with low priority value is to be displayed before a quiz question with high priority value
  4. belongingQuizID:
     the id of the specific quiz at hand
  5. option1:
     the text of the first option
  6. option2:
     the text of the second option
  7. option3:
     the text of the third option
  8. option4:
     the text of the forth option
  9. correctOption:
     the text of the correct answer, it is equal to one of the 4 options

5.1.2 Entity Relationship model

Here follows the ER diagram that shows the relationship among the entities we defined in the previous section.



The cardinality relationships among entities are described both by the diagram and in the following lines:
A subject needs to contain at least one module, but usually it contains many more;
A Module can only be associated to one and only one Subject.
A Module can have one or more quizzes: at the current state of the app we have only one Quiz per module, but with future release of the app we can add more quizzes to some modules, given the design of the database.
A Quiz belongs always to one and only one Module.
A Quiz is composed of many Quiz Questions: the number of questions per Quiz is not fixed and it varies depending on the Module selected;
a Quiz Question is associated to one and only one Quiz.

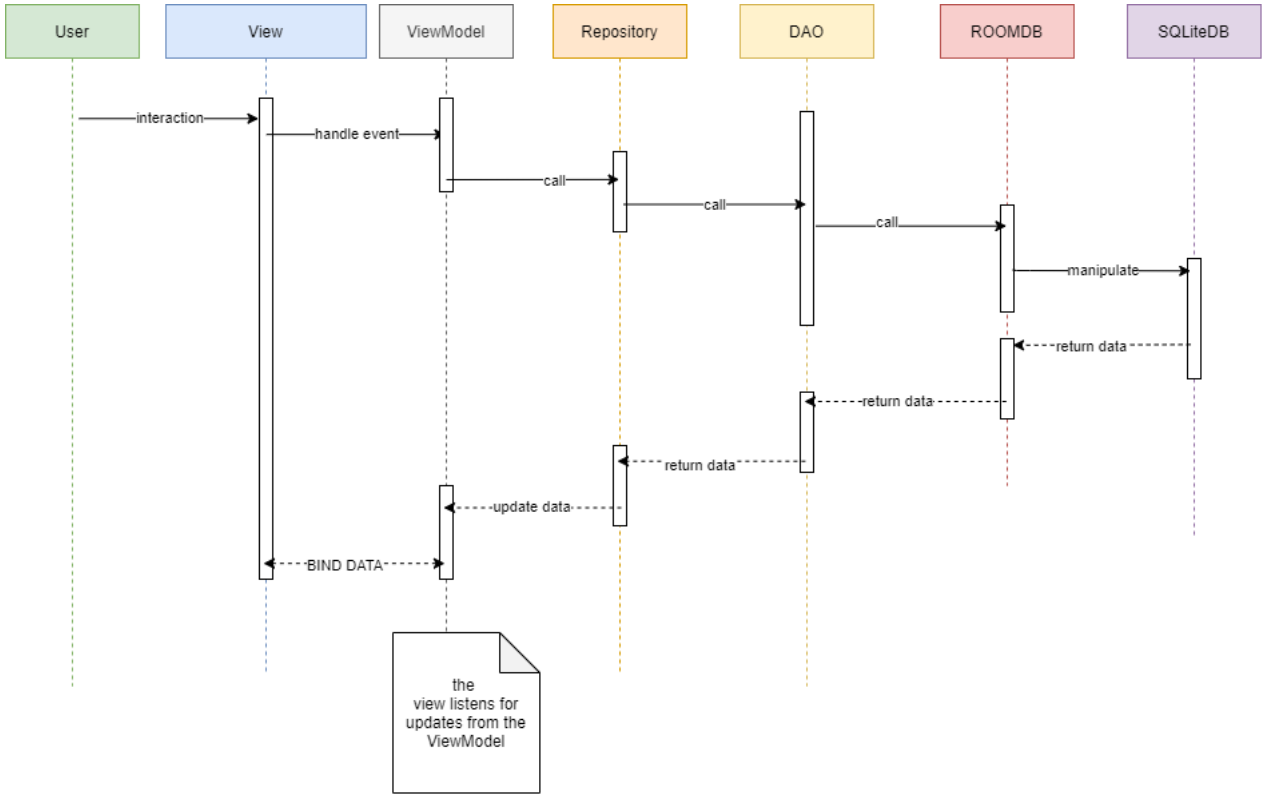
5.2 Implementation

The database has been implemented using the Room persistence Library. We have given a detailed overview of its advantages with respect to other libraries and how we used it in the Architectures chapter.

6 UML Diagrams

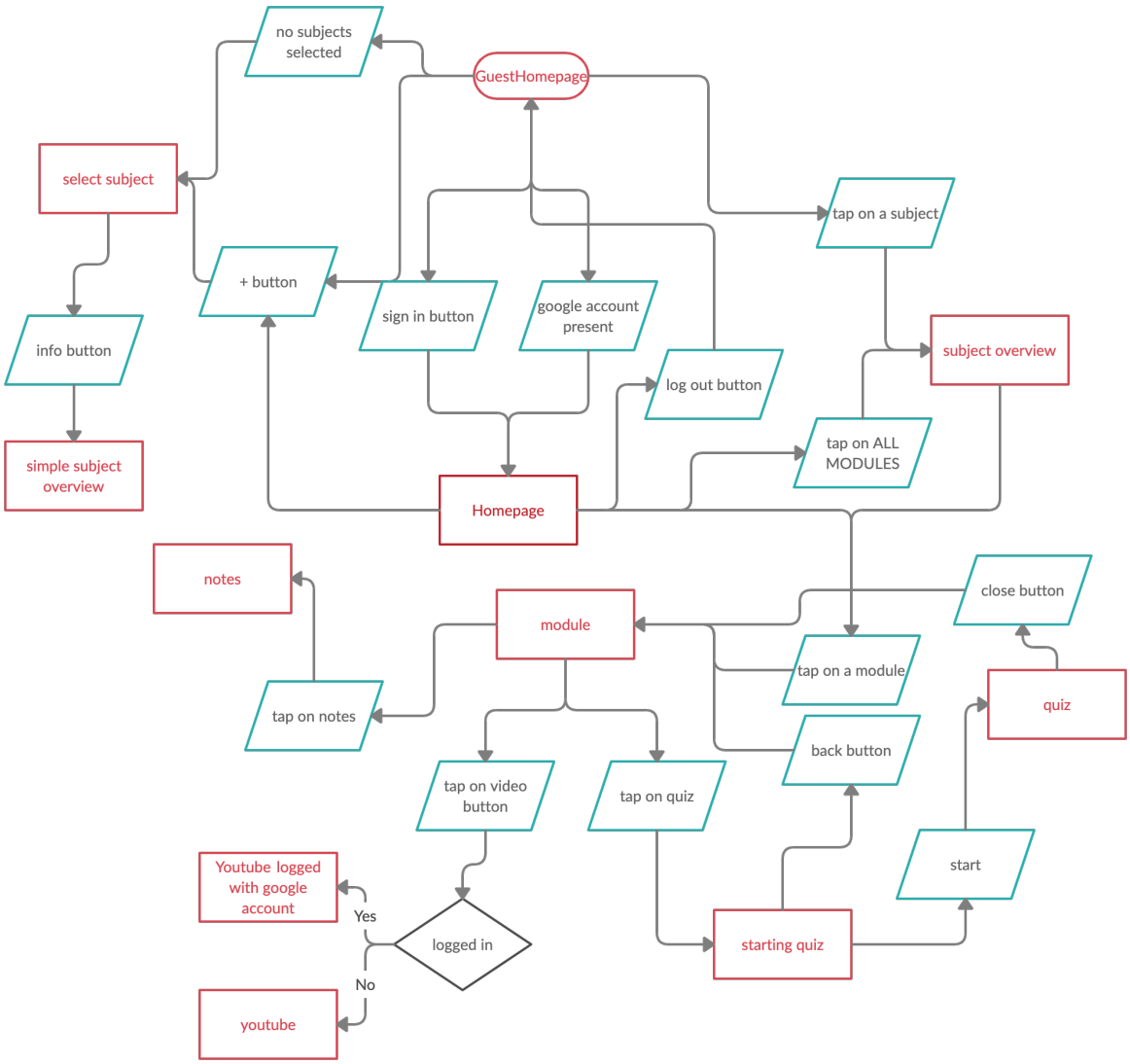
6.1 Sequence Diagrams

The followng sequence diagram shows what happens whenever an user app interats with the other to perform an action that consists in an update in the database.



As we have previously described in the Architectures chapter, in the MVVM pattern the direct communication happens only in the direction from the View to the Model, the communication in the other direction happens by means of returning a value or, in the case of the interaction between the ViewModel and the Model it happens because the View subscribes to an element stored in the ViewModel and gets notified whenever it changes its value.
An example of scenario in which the sequence of events we have described in the diagram above happen is whenever a user interacts with the user inteface to answer a question of the quiz: the view displays whether the given answer is correct because it retrieves from the ViewModel the correct answer and a call to the Model is made to retrieve the next question. Once the question is retrieved by the ViewModel, the View notices the change of value of the current question and updates its content with the new question and the new options.

6.2 Flow chart



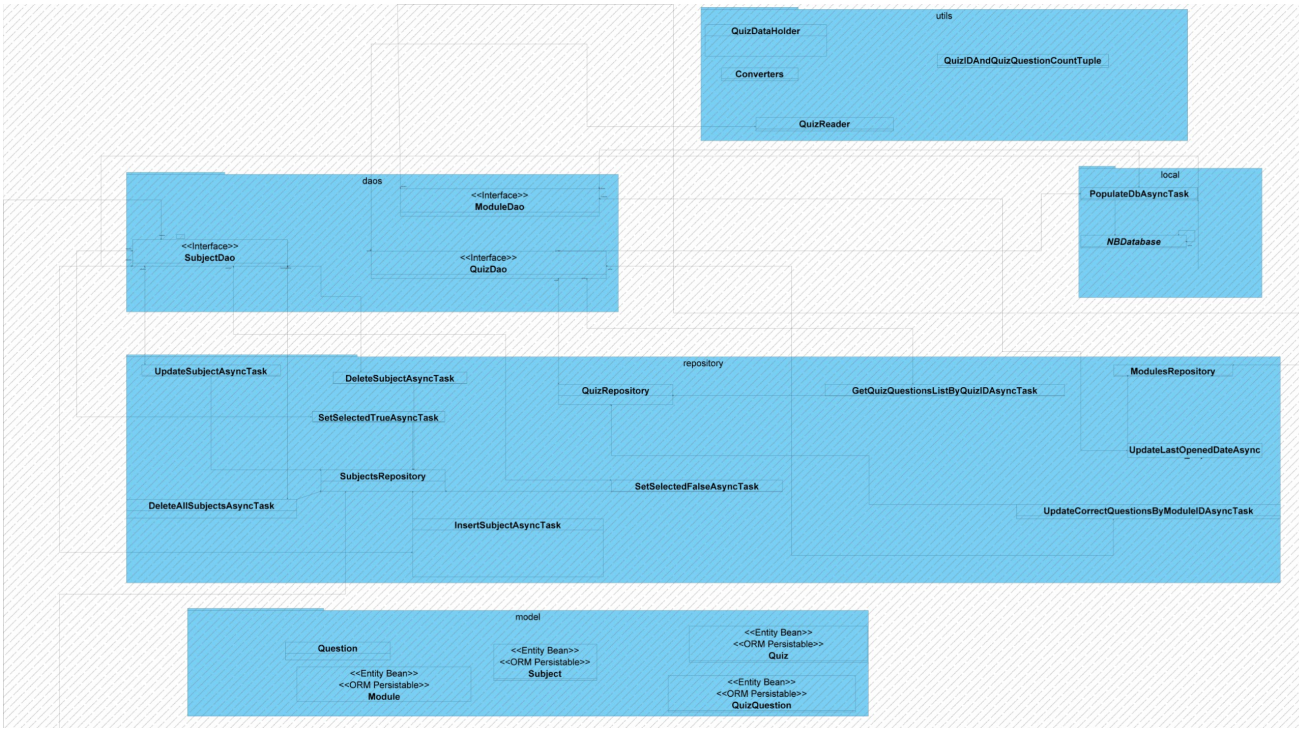
In this image is represented the flow diagram of the application. The main activity, which is the one that is accessed every time a user closes and reopens the application, is the GuestHomepage.
At the first start of the app however, the first activity the user sees is the Select subject: this is because, when the GuestHomepage is called, there is a check in the onCreate method; if there are not any subject selected (so basically when the app if started for the first time), the intent to reach the Select Subject activity is started.
The same intent is triggered also by pressing the + button to add or remove the subjects.
By tapping on the info button in the Select Subject page is possible to reach the Simple Subject Overview.
From the GuestHomepage there is a button to do the log in with a Google account, in this way the user reaches the Homepage activity, however this is not mandatory;
thanks to the log in a user can access the Homepage where he can see the most recent modules he has accessed order to quickly resume where he interrupted.
We are also planning future updates with features reserved to the Users only (so to the user logged in).
However everyone can access the main features of the application, which are the view of the notes and the quizzes.

Both from the GuestHomepage and the Homepage is possible to reach the Subject Overview activity.
In here the user can open any module he wants, and once he is in the Module activity, he can decide to have a look to the notes, to watch the videos that are linked if available or to take part to the quiz.
By clicking on the quiz button he will arrive in the starting quiz activity.
From here he can return to the Module activity or start the quiz. The number of questions of each quiz are variable, and it depends on the selected modules.
More questions or quiz can be added with future updates, so it is possible in the future that some modules can have more than one quiz.

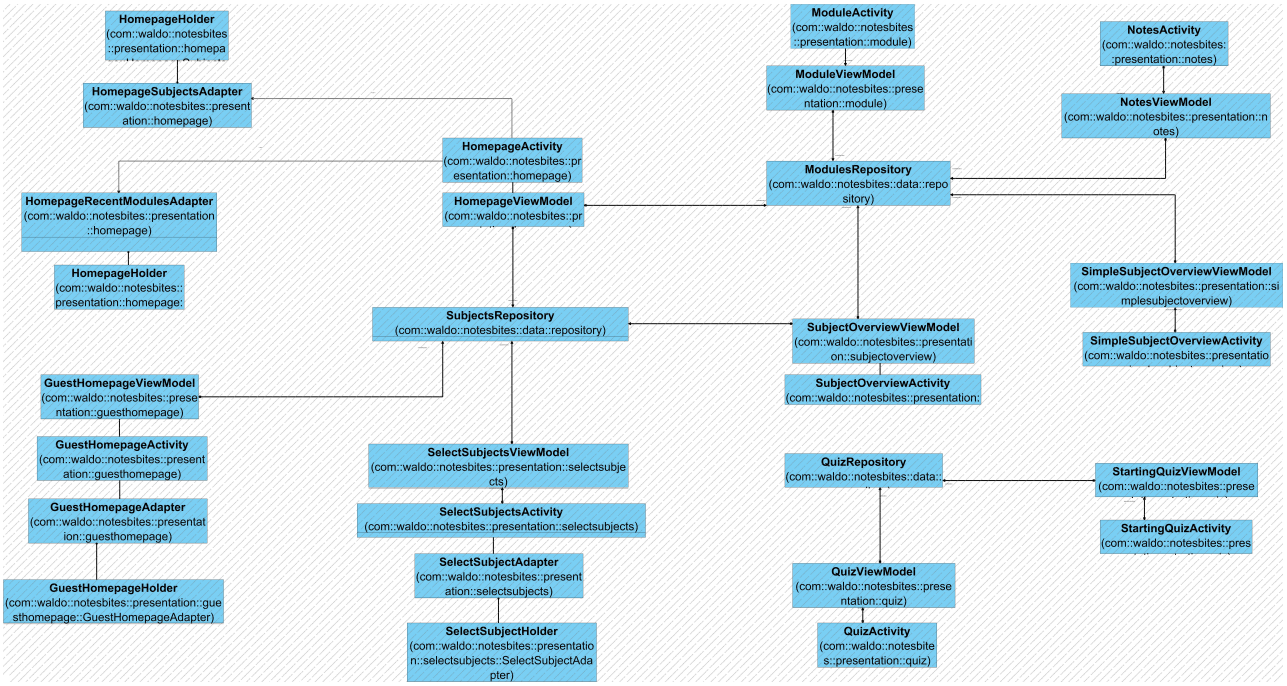
6.3 Class Diagrams

The Class diagram has already been presented in the Architectures chapter (given it is helpful to understand the project architecture), but we will show it in this chapter as well to gather in a unique place all the UML diagrams.

Data Layer Class Diagram



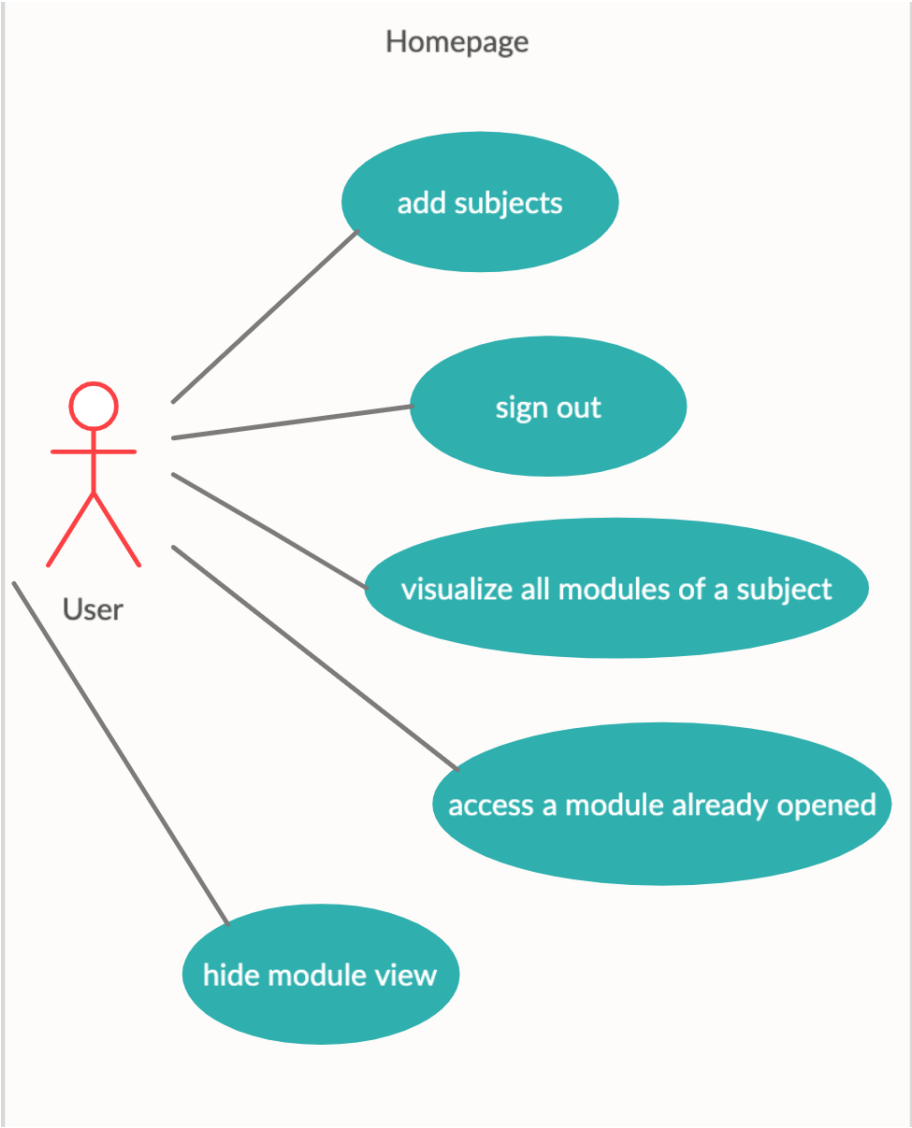
Presentation Layer Class Diagram



6.4 Use Case Diagrams

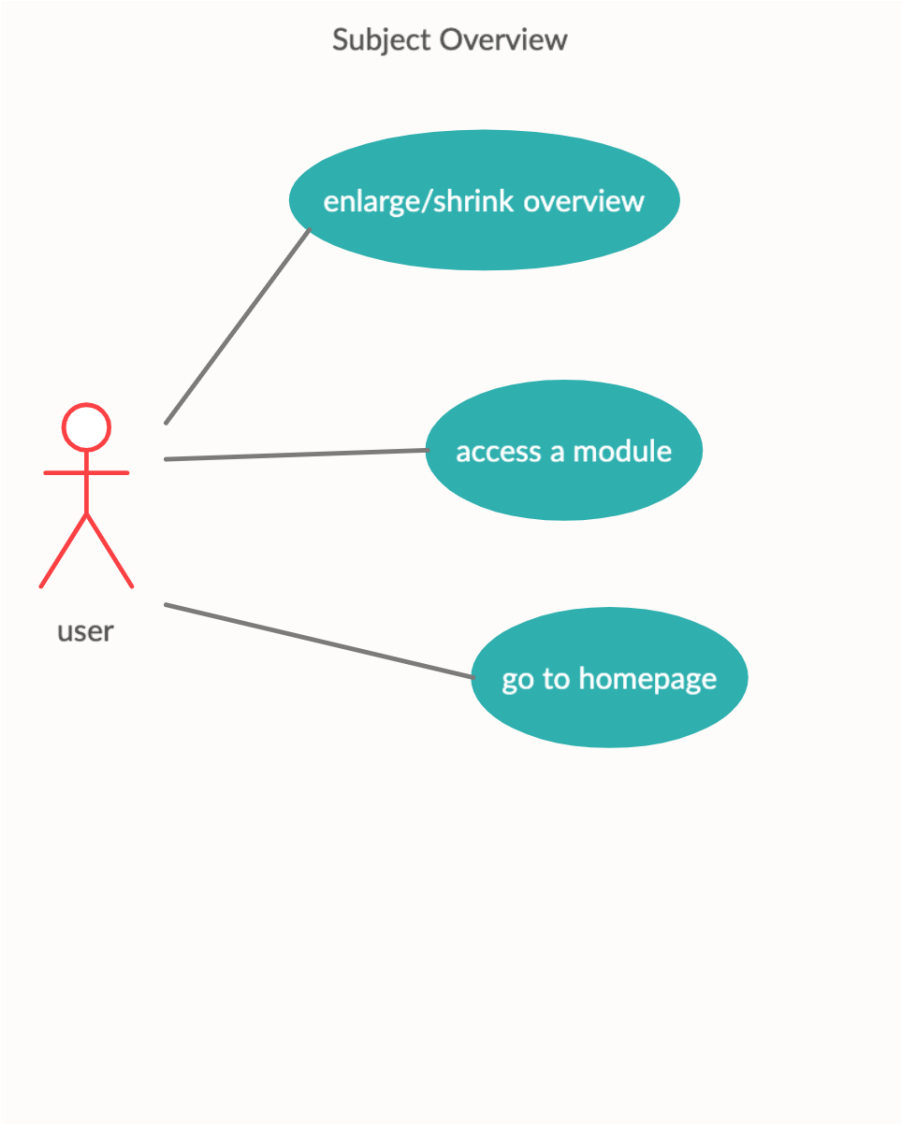
In this section we will show the actions a person can perform in specific activities in the app

6.4.1 Homepage



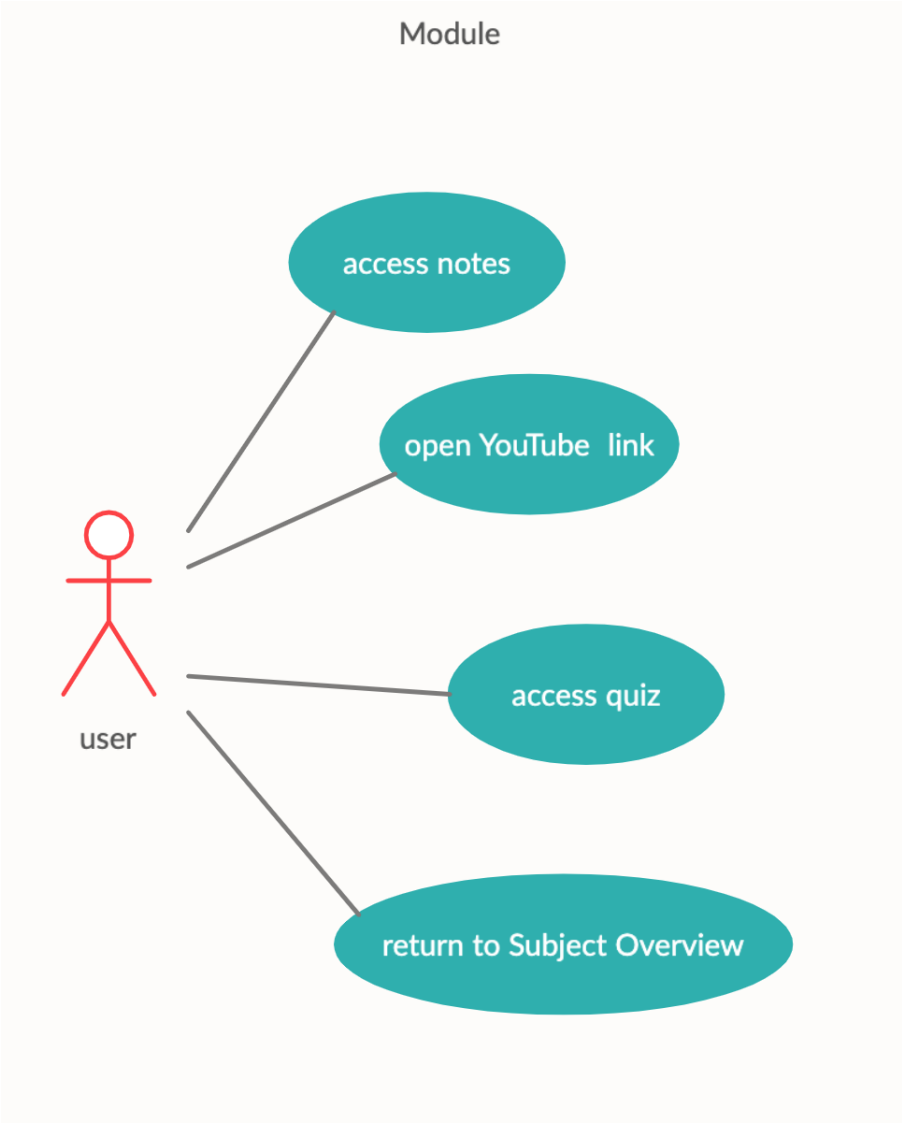
Here is possible to see what a User (with the capital U because we are referring to a person who has performed the sign in operation with his Google account since we are in the Homepage and not GuestHomepage) can do in the page.
He can add a subject (or delete one he has already added) by clicking on the add button where he will be redirected to the Select Subject activity. He can log out from his Google account and by doing so he will arrive in the GuestHomepage.
He can tap on the button to visualize all the modules of the selected subject and he will be redirected in the Subject Overview page.
Since we are in the Homepage not accessible from the Guests (users that have not performed the access with a Google account), the User can see the modules he has already opened of a subject and immediately arrive in the Module page. Lastly, by doing a swipe to the right on the "My Subject" text he can hide the section where the modules are shown.

6.4.2 Subject Overview



In the image above are shown the actions a user (this time with user we refer to both a User and a Guest) can perform in the Subject Overview page. As already shown in previous schemas the user can see the list containing all the modules of a subject and access to one of them or go to the respective homepage (GuestHomepage for the Guests, Homepage for Users).
He can also enlarge and shrink the overview of a subject by simply tapping on the drop down menu.

6.4.3 Module



In the module page there are four buttons.
A user can tap on the first one to access the markdown file which contains the notes of the module.
By tapping on the second he will be redirected on the YouTube app (if the link is available for the selected module) where he can watch a video that aims to elaborate on the topic of the module;
if the user has logged with his Google account he will be redirected to the YouTube app with his account, so he can easily add the video to any playlist he wishes.
If the click on the third button he will arrive to the Starting Quiz page and by tapping to the back button he will return to the Subject Overview page.

7 Future Implementations

In this section we will talk about the next features we are planning to integrate into the app.

7.1 Download only certain subjects

We'd like to give the user the possibility to download only a subset of all the available subjects.
The SelectSubjects page would be the activity in which the user chooses what to download locally on his database.
This obviously translates into hosting the markdown files in an external database an exploit the Repository pattern to handle the fetching of data from the external database to the local one and to communicate information (like the results of quizzes) from the smartphone to the external database.
This feature would obviously make the app less heavy, given that there is no need to host locally files that are of no interest to the user.

7.2 Open Collaboration Project

The university notes we have taken during these years have surely their limitations. They probably don't cover perfectly every aspect of the correspondent topic and students' notes are always prone to mistakes.
It would then be useful to have a community of users that can easily edit the content of any module. The edits would need to be approved by an administrator or by a community of users of the app in order to be reported in the official markdown file.
Most of the students are prone to take notes of subjects they are interested in, and we think that this feature can be beneficial for both the community of users and whomever decides to share his work.
The former would have available better material, and the latter would be recognized for his contribution.
Finally, we think it is great to promote an open collaboration mindset, given that it has proved to be successful in other environments, like Wikipedia.