

Insert here your thesis' task.





**FACULTY  
OF INFORMATION  
TECHNOLOGY  
CTU IN PRAGUE**

Bachelor's thesis

## **StudyPad - Android Client**

***Roman Levinzon***

Department of Software Engineering  
Supervisor: Ing. Miroslav Balík, Ph.D

May 5, 2019



---

## Acknowledgements

THANKS (remove entirely in case you do not wish to thank anyone)



---

## Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 5, 2019

.....

Czech Technical University in Prague

Faculty of Information Technology

© 2019 Roman Levinzon. All rights reserved.

*This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).*

### **Citation of this thesis**

Levinzon, Roman. *StudyPad - Android Client*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.



---

# Abstrakt

StudyPad je kombinace služby pro porířování poznámek a sociální sítě s cílem pomoci studentům zapamatovat si různé informace. Cílem práce je vyvinout aplikaci pro OS Android, která bude sloužit jako klient. Tento text uznává stávající řešení, obsahuje analýzu domén a požadavků, popis a výběr architektury aplikace a její implementace

**Klíčová slova** Mobilní aplikace, Android, Kotlin, MVVM, Clean architecture, vzdělávání

---

# Abstract

StudyPad is a combination of a note-taking service and a social network, aimed at helping students to memorise different pieces of information. The goal of this thesis is to develop an application for Android OS that will serve as the client. This text acknowledges existing solutions, contains domain and requirements analysis, description and the choice of application's architecture and its implementation.

**Keywords** Mobile application, Android, Kotlin, MVVM, Clean architecture, education



---

# Contents

<b>Introduction</b>	<b>1</b>
Goal . . . . .	1
Motivation . . . . .	2
<b>1 State-of-the-art</b>	<b>3</b>
1.1 StudyPad Service . . . . .	3
1.1.1 Sharing & Collaboration . . . . .	3
1.2 Android Platform . . . . .	4
1.2.1 Android SDK Versions . . . . .	5
1.2.2 Android High-level Framework Elements . . . . .	6
1.2.3 Resources . . . . .	6
1.2.4 Views . . . . .	7
1.3 Android Studio . . . . .	7
1.4 Gradle . . . . .	7
1.5 Kotlin . . . . .	7
1.5.1 Kotlin Extensions . . . . .	8
1.6 Application Architecture . . . . .	8
1.6.1 Clean Architecture . . . . .	8
1.6.2 MVP . . . . .	8
1.6.3 MVVM . . . . .	9
1.6.4 Android Architecture Components . . . . .	9
1.6.5 Recommended Architecture . . . . .	9
1.6.6 Chosen Architecture . . . . .	10
1.7 Firebase . . . . .	10
1.8 System description . . . . .	11
1.8.1 User Roles . . . . .	11
1.8.2 StudyPad API . . . . .	12
1.8.3 Android client . . . . .	12
1.9 Domain Description . . . . .	15

1.9.1	User . . . . .	15
1.9.2	Note . . . . .	16
1.9.3	Notebook . . . . .	16
1.9.4	University . . . . .	16
1.9.5	Published Notebook . . . . .	16
1.9.6	Topic . . . . .	16
1.9.7	Tag . . . . .	16
1.9.8	Comment . . . . .	16
1.10	Requirements . . . . .	17
1.10.1	Functional Requirements . . . . .	17
1.10.2	Non-functional requirements . . . . .	19
1.11	Existing solutions . . . . .	20
1.11.1	StudyBlue . . . . .	20
1.11.2	Quizlet . . . . .	21
1.11.3	Cram . . . . .	22
1.11.4	TinyCards . . . . .	23
<b>2</b>	<b>Design</b>	<b>25</b>
2.1	Material Design . . . . .	25
2.1.1	Used Components . . . . .	25
2.2	UI & UX . . . . .	26
2.2.1	Library Managent . . . . .	26
2.2.2	Notebook Publishing . . . . .	28
2.2.3	Sharing Hub . . . . .	31
2.2.4	Searching Flow . . . . .	32
2.2.5	Published Notebook Details . . . . .	33
2.2.6	Challenges . . . . .	35
2.2.7	Profile . . . . .	38
2.3	Application architecture . . . . .	38
2.3.1	MVVM . . . . .	39
2.3.2	Repository . . . . .	39
2.3.3	Interactors . . . . .	39
2.3.4	Dependency Injection . . . . .	39
2.3.5	StudyPad Architecture . . . . .	39
<b>3</b>	<b>Implementation</b>	<b>41</b>
3.1	Used Libraries and Technologies . . . . .	41
3.1.1	DI Framework . . . . .	41
3.1.2	HTTP Client . . . . .	42
3.1.3	Threading . . . . .	43
3.1.4	Android Navigation Component . . . . .	43
3.1.5	ViewModel . . . . .	43
3.1.6	LiveData . . . . .	44
3.2	Architecture Implementation . . . . .	45

3.2.1	Repository . . . . .	45
3.2.2	Interactors . . . . .	46
3.2.3	Applying ViewModel and LiveData . . . . .	48
3.2.4	Present the data in Fragments . . . . .	49
3.3	Component diagram . . . . .	50
3.4	Installation . . . . .	50
<b>4</b>	<b>Testing</b>	<b>51</b>
	<b>Conclusion</b>	<b>53</b>
	Future development . . . . .	53
	<b>Bibliography</b>	<b>55</b>
<b>A</b>	<b>Acronyms</b>	<b>59</b>
<b>B</b>	<b>Contents of enclosed CD</b>	<b>61</b>



---

## List of Figures

1.1	StudyPad sevice . . . . .	4
1.2	Android Versions distribution . . . . .	5
1.3	Architecture Example Diagram . . . . .	10
1.4	Android component Diagram . . . . .	13
1.5	Component Diagram . . . . .	14
1.6	Domain Diagram . . . . .	15
1.7	Functionality Symbols . . . . .	20
2.1	My Library Section Wireframes . . . . .	27
2.2	Notebook Publishing Flow - Sketch Phase . . . . .	29
2.3	Notebook Publishing Wireframes . . . . .	30
2.4	Sharing Hub Wireframes . . . . .	31
2.5	Search Flow Wireframes . . . . .	32
2.6	Published Notebook Details Wireframes . . . . .	34
3.1	Navigation Editor . . . . .	44
3.2	ViewModel lifecycle . . . . .	45





---

## List of Tables

1.1	StudyPad & StudyBlue requirements comparison . . . . .	21
1.2	StudyPad & Quizlet requirements comparison . . . . .	22
1.3	StudyPad & Cram requirements comparison . . . . .	23
1.4	StudyPad & TinyPads requirements comparison . . . . .	23



---

# Introduction

Each year, our smartphones get smarter and mobile applications become more advanced. The arrival of smartphones and mobile applications has completely changed our way of living, made it easier, and it is hard to come up with a single aspect of life, that has not been improved by one or the other application. They are everywhere: helping us to navigate in our neighbourhood, keeping us up to date with latest news, helping us to stay in touch with our loved ones, stay fit and healthy and more. In addition, some of the most popular applications tend to teach us something new.

Educational applications are very popular on both most popular mobile platforms (iOS and Android). Many of them use a flash card system in the study process – displaying small pieces of information one after another, so that the user can memorise it more efficiently. However, most of these services are fairly limited in terms of what they are trying to teach and some of the greatest features are scattered across different applications and services. StudyPad is a new solution to this problem. Using Android OS, today's most popular mobile platform, StudyPad wants to give its users freedom in what they can learn and give them proper tools to share, exchange and collaborate on study materials to make the education process even more easier.

## Goal

The goal of this thesis is to deliver StudyPad client application for Android OS with the emphasis on content creation, sharing and collaboration. This includes following steps:

- analysis of the existing solutions,
- analysis of the functional and non-functional requirements,
- requirements design and its implementation,
- testing of the resulted solution.

### **Motivation**

The primary motivation is to practice software development processes on the big and scalable project while applying advanced practices and technics of Android Development.

---

# State-of-the-art

## 1.1 StudyPad Service

StudyPad is a combination of a note taking service and a social network. It is intended for students and everyone who wishes to learn something new. The primary focus of StudyPad is on content creation, its sharing and discovery.

The StudyPad project originated from several subjects on Faculty of Information Technology (FIT) in Czech Technical University (CTU). The first iteration of backend was introduced as a semestral project for subject Enterprise Java (BI-EJA), and subject Introduction to Android Development (BI-AND) had great influence on the first iteration of the Android Client. All of these components will be described more thoroughly in Chapter 2.

StudyPad core concepts are *Notebooks* and *Notes*. Notebook is simply a collection of notes united by one theme. This may be a subject in school, a language that you would like to learn or a set of questions that you could hear at a job interview. Note, in turn, is a part of the notebook and represents a single piece information that has a name and a content. It can also be interpreted as a question and answer or a term and its definition.

Each user has his/her own space where they can create, store and edit notebooks and notes (hereafter *Library*). All notebooks stored in the library can be used in various tests and exercises to help users to memorise its content.

### 1.1.1 Sharing & Collaboration

StudyPad also allows users to easily share their notebooks with each other. Each notebook can be shared by creating a published version of it (hereafter Published Notebook), thereby making it available to everyone else for viewing and importing. The publication process involves providing additional information about the notebook, including its name, an optional description, topic and optional tags that serve to narrow the topic. Everything the user has provided (hereafter Author) along with their school and language will then be

used in the process of searching and filtering – which will facilitate the search for the necessary materials. StudyPad also makes it possible to quickly share a notebook by sending a link, which will act as a deep-link, navigating the user directly to a Published Notebook.

The Author of the notebook reserves the right to make edits/corrections to the Published Notebook. Other users (hereinafter subscribers) in turn can save a Published Notebook to their library or suggest some changes or corrections to improve the content. By saving Published notebook to his/her library, the subscriber will be able to make any local changes as he/she sees fit and use it as normal. The Subscriber will be notified about any updates made to the Published Notebook so that he/she can apply the changes to his/her local version, however, any local changes will be deleted. The Author of the notebook, in turn, will be notified about latest suggestions and comments left by the subscribers.

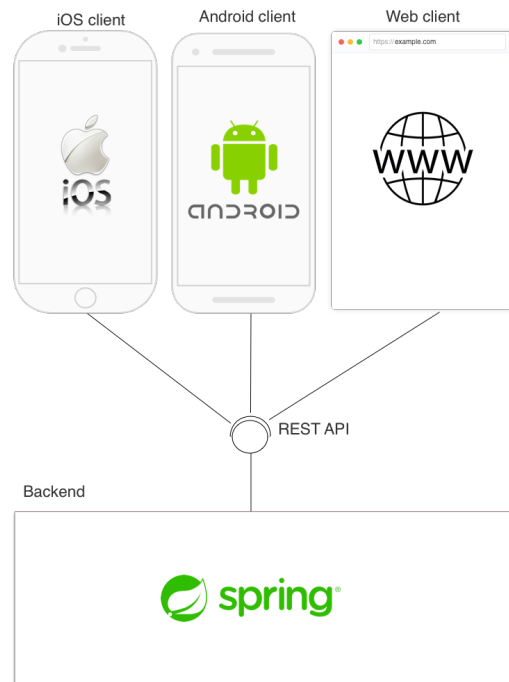


Figure 1.1: StudyPad service

## 1.2 Android Platform

Android is a mobile operating system developed by Google. It is based on a modified version of the Linux kernel and other open source software, and is designed primarily for touchscreen mobile devices such as smartphones and tablets.[1]

### 1.2.1 Android SDK Versions

Since its inception, several versions have been released, with the latest one being 9.0, codenamed “Pie”. Every new version provides new and improved APIs to work with and backward compatibility for applications running on older versions.[2] Originally, the Support Libraries were used to provide backward compatibility while targeting newer versions. This was the case until AndroidX was introduced.

AndroidX, in essence, is a refactored version of the Support Libraries. Just like Support Libraries it ensures backward-compatibility and is shipped separately from the Android OS, but it also provides a more clear and convenient package names and uses a strict semantic versioning system. [3]

When developing applications for Android, the developer has to consider which OS versions the application will support. Specifically, the `minSdkVersion` and `targetSdkVersion` attributes are used to identify the lowest API level with which your application is compatible and the highest API level against which you have designed and tested your application. StudyPad will target the latest Android version available, with the minimal version set to 5.0, codenamed Android Lollipop. This will ensure a high compatibility rate of 85% and Material Design Support, which is one of the non-functional requirements.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.2%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.1%
4.2.x		17	1.5%
4.3		18	0.4%
4.4	KitKat	19	7.6%
5.0	Lollipop	21	3.5%
5.1		22	14.4%
6.0	Marshmallow	23	21.3%
7.0	Nougat	24	18.1%
7.1		25	10.1%
8.0	Oreo	26	14.0%
8.1		27	7.5%

Figure 1.2: Android Versions distribution

### 1.2.2 Android High-level Framework Elements

Android SDK provides multiple elements and APIs for developers to help them build UI and communicate with the Android OS.

- **Activity:** **Activity** represents a single window where the UI elements can be placed. Android OS managing the creation and destruction of the **Activities**, that is why **Intents** are used to start/launch it. Activity's lifecycle is [4].
- **Fragment:** **Fragments** were introduced in Android 3.0, primarily to support more flexible UI designs for larger screens (like Tablets). In such designs, **Activity** serves as container for one or more **Fragments**, and because each **Fragment** defines its own layout with its own lifecycle, it can be used in multiple **Activities** [5].
- **Broadcast Receiver:** **Broadcast Receiver** allows the application to receive and send local and system-wide broadcasts similar to the publish-subscribe design pattern. By using **Broadcast Receiver** application can act upon receiving various kind of events, like the change of internet connectivity or charging status.[6].
- **Service:** **Service** is the application component that does not require UI and is used to perform long-running operations, even if the user stops interacting with the application [7].
- **Intent:** *"An intent is an abstract description of an operation to be performed"*. **Intent** can be used to launch an **Activity**, start a **Service** or a **BroadcastReceiver**. **Intent** can either start a specific component (explicit Intent) or try to find a component that can complete certain actions (implicit Intent) [8]. For example, there could be several applications available on the device with the image viewing functionality, implicit intent, in this case, will allow the user to choose which application will handle this section. Explicit intent, on the other hand, is perfect when launching your own **Activities** and other components.

### 1.2.3 Resources

Resources are the additional files and static content that your code uses, such as bitmaps, layout definitions and user interface strings. Resources are stored in the separated package called **/res** and can be referenced in the application using ids. Also, depending on current configuration, Android can choose appropriate resource version (if multiple are provided) at run-time. For example, it can be used for localisation support by providing different strings resources for different languages.[9]



### 1.2.4 Views

**View** is fundamental building block for all the UI components. It serves as base class for **Widgets**, which are the interactive elements and **ViewGroups**, that acts as a containers for other Views. [10]

**ViewGroup**, as mentioned before, is an invisible container that holds other views. Android provides different ViewGroup subclasses, each with its own purpose and a way to structure its children [11]. **LinearLayout**, for example, allows to align widgets horizontally or vertically.[12]. Another important ViewGroup that has received a lot of attention and support during the last couple of years is **ConstraintLayout**. Using constraints, it allows developers to create more complex layouts without using nested ViewGroups (i.e., **LinearLayout** inside another **Linear Layout**).[13]

Android uses XML for declaring the UI for Activities and Fragments. XML is then converted to the View object using **LayoutInflater**. Views can also be added programmatically via Java/Kotlin code.

## 1.3 Android Studio

Android Studio is an official IDE for Android development. Out of the box it provides a lot of essential tools, like visual Layout editor, emulator and more.

## 1.4 Gradle

Gradle is a build toolkit, that is used to automate and manage the build process, while allowing you to define flexible custom build configurations using Groovy-based DSL (Domain Specific Language). Android plugin, shipped alongside the Android Studio, allows to modify and control various build parameters such as **buildVariant**, dependencies, artifacts signings, and manifest entries.[14]

## 1.5 Kotlin

*“Kotlin is a pragmatic programming language for JVM and Android that combines OO and functional features and is focused on interoperability, safety, clarity and tooling support.”* Kotlin is a general purpose language developed by JetBrains and it works everywhere Java works and is fully interoperable with it, meaning every Java code can be converted to Kotlin and vice-versa.[15] Since May 17th 2018, Kotlin became the official language for Android development with all the tools and libraries required shipped with Android Studio out of the box.[16].

### 1.5.1 Kotlin Extensions

Similar to some other languages, Kotlin provides a way to extend any class with new functionality without inheritance. There are two types of extensions: Functions and Properties. All the extensions are resolved statically and do not modify the class they extend.[17]

When it comes to Android, a plugin developed by JetBrains provides a lot of useful extensions that enhance the experience of Android development. For example, it is possible to reference Views from Activity/Fragments directly using its ids. Previously it was required to call `findViewById<View>()` (Listing 1.1) to obtain a view reference, which resulted in a lot of boiler-plate code. [18]

Listing 1.1: Kotlin View Binding example

```
// Instead of findViewById<TextView>(R.id.textView)
textView.setText("Hello, world!")
```

Due to its official support and features, Kotlin was chosen as a primary language for StudyPad development as opposed to Java.

## 1.6 Application Architecture

Application Architecture defines the way different application elements interact with each other and how the application is designed and organised.

### 1.6.1 Clean Architecture

Clean architecture is a set of ideas derived from different kinds of other approaches with the main principle being a separation of concerns. Separation of concerns ensures that all the application's components are easily replaceable and as less independent as possible. When it comes to Android and every software that contains UI, it is essential that the UI does not contain any business logic and serves only as a way to show information.

### 1.6.2 MVP

MVP or Model-View-Presenter is an architecture pattern that is very common in Android development. Model is the data layer View is the UI layer, which in the Android worlds are Fragments and Activities. Presenter - is the layer between the View and the Model. It receives events from the View and updates the Model layer and applies the result to the UI layer. The View and Presenter both are abstracted to an Interface, it makes all the layers easily testable. [19] However, one of the main disadvantages is that Presenter must hold a reference to a View, which can lead to memory leaks and other problems around Android lifecycle.

### 1.6.3 MVVM

MVVM or Model-View-ViewModel is another pattern to apply Clean Architecture principles. It is very similar to MVP when it comes to View and Model layers, with the only difference being ViewModel. ViewModel, as oppose to Presenter, is not aware of the View – it does not hold the reference to it. Instead, View subscribes to ViewModel events and its state accordingly. ViewModel handles UI interactions, updates the Model layer and then post changes so that the View can get updated.[20]

### 1.6.4 Android Architecture Components

Previously, Google, as the leading Android contributor, never cared about the way developers build their applications. It has changed drastically with the introduction of Architecture Components at Google I/O in 2017.[21] Architecture Components are a collection of libraries that can be used to design a robust and maintainable architecture. LiveData and ViewModel components are the most important. LiveData is a lifecycle-aware component is used to store data and notify its subscribers if the data changes. ViewModel can store UI-related data and survive Activity Destruction which is especially useful in case of device rotation for example.

### 1.6.5 Recommended Architecture

Introduction of Architecture Components made it clear what approach Google recommends when designing application architecture - MVVM. Figure 1.3 demonstrates this approach:

- Fragments and Activities represent the View
- View is connected to a ViewModel. ViewModel there is represented by a ViewModel Architecture component that stores model data using LiveData, that Fragment or Activity can subscribe to. ViewModel requests data from the repository, without knowing where the data comes from
- A Repository pattern represents the Model layer. Repository's responsibility is to retrieve and modify the data using one or multiple sources. Typically there is remote data source (API) and the local data source (on-device database).

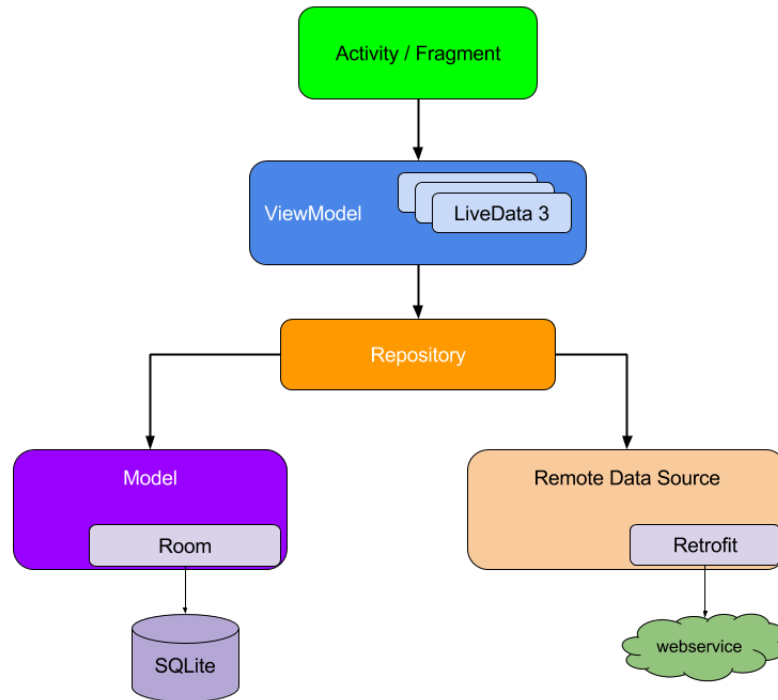


Figure 1.3: Architecture Example Diagram

### 1.6.6 Chosen Architecture

The fact that the MVVM pattern is recommended by the developers of the Android OS and the existence of Architecture Components makes MVVM a solid choice and good starting point for the application architecture. Its detailed design and implementation is discussed in the following chapters.

## 1.7 Firebase

*“Firebase lets you build more powerful, secure and scalable apps, using world-class infrastructure.”* [22] Firebase provides a high number of services, some of which are a necessity in today’s world of software development.

StudyPad will be using the following services:

- **Firestore Analytics** Firestore Analytics provides free application measurement solution, that allows you analyse how the users interact with your application based on up to 500 distinct events [23]
- **Firestore Cloud Messaging** *“Firestore Cloud Messaging (FCM) provides a reliable and battery-efficient connection between your server and*

*devices that allows you to deliver and receive messages and notifications on iOS, Android, and the web at no cost.”[24]*

- **Firebase Auth** “*Firebase Auth provides an end-to-end identity solution, supporting email and password accounts, phone auth, and Google, Twitter, Facebook, and GitHub login, and more.”[25]*
- **Crashlytics** Crash Reporting allows you track detailed reports of the errors and crashes in the application. All the issues experienced by the users are then grouped into clusters of similar stack traces and triaged by the severity of impact on application users.[26]

While Firebase Analytics and Crash Reporting services are a great help while developing mobile applications, Firebase Auth and FCM are essential for some of the functional requirements.

Storing user credentials, and most importantly, storing them safely is not an easy task. With a little effort, Firebase Auth will handle everything related to user management, including access token generation and refreshment, as well as credentials storing and encryption.

Firebase Cloud Messages will enable push-notifications, which is a great way to interact with users and introduce some real-time functionality to the application.

## 1.8 System description

The StudyPad system follows classic client-server software architecture. The server part is represented by the Spring Boot application, which communicates with its clients using REST API. The client part consists of client applications for several platforms: Android, iOS and Web. The Web client is being developed alongside the Android client to serve as an Admin Panel.[27] The iOS client was developed by the author of this thesis within the subject Introduction to iOS Development (BI-IOS). [28] The main task of this thesis is to deliver an Android OS client.

### 1.8.1 User Roles

Some of the use-cases requires moderation. The simplest example is a university creation use-case. Though it is possible to submit new school/university in case user haven't found it, the name, that user has submitted has to be verified first and other details, like school location and translations should be updated. This is a main purpose of the Admin Panel and it will require an introduction of different user roles: User and Admin.

The User role is a default one and will be assigned automatically to everyone who completes the registration process. It will only be used in the client

applications. Admin role is the only one that have the access to the admin panel and an ability to view and modify certain data.

### 1.8.2 StudyPad API

StudyPad has its own REST API that is represented by Spring Boot application using JSON as a data-interchange response format. [29] The Spring Framework is a Java-based framework that enhances developing of Java EE applications. It has modular architecture and supports Dependency Injection and using Inversion of Control principles.[30]. In Spring 5.0 official support for Kotlin has been introduced.[31]

The author of this thesis developed the first iteration of the API as a semestral project for subject BI-EJA. The API is still being updated to support requirements for the client applications.

### 1.8.3 Android client

The detailed structure of the Android client and how it is connected with other components is presented on the component diagram on Figure 1.5.

Android component will be divided into three layers: the Data Layer, Business Layer and Presentation layer. The Data layer is only responsible for downloading and storing data. The Business Layer responsibility is to handle user interactions and converting downloaded data to something that presentation layer can present. Presentation layer is only responsible for displaying data. Android component will also connected be to Facebook SDK, Google Auth SDK and Firebase SDK to provide authentication options. Firebase SDK will also enable an analytics service, crash reporting and push notifications.

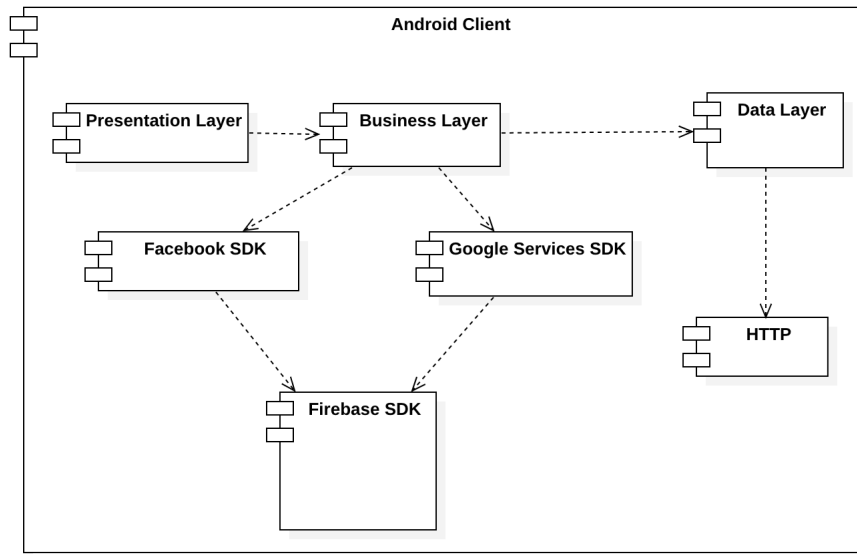


Figure 1.4: Android component Diagram

## 1. STATE-OF-THE-ART

---

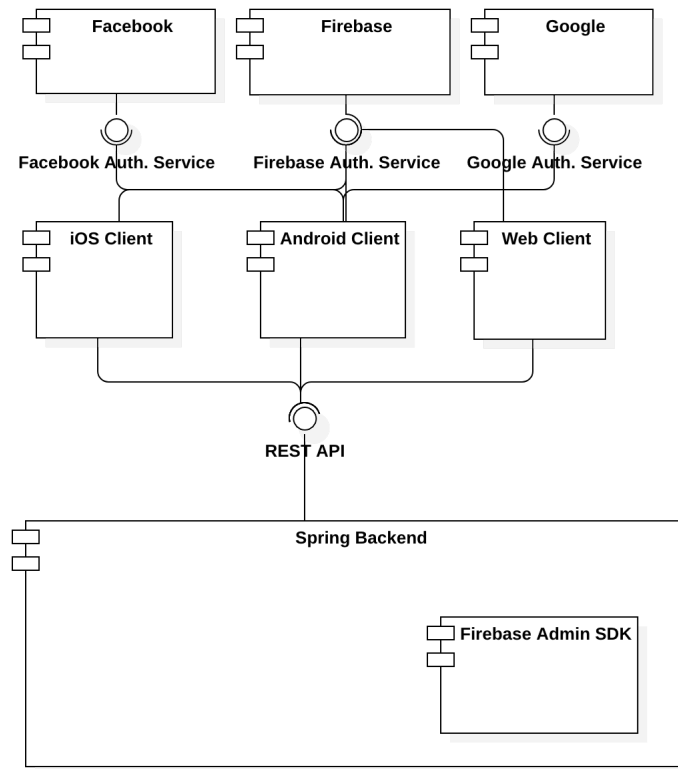


Figure 1.5: Component Diagram



## 1.9 Domain Description

The Class Diagram on Figure 1.6 represents the Domain Model of the application, it provides visual representation of Entities and relations between them. The Design is based on the entities used on server-side.

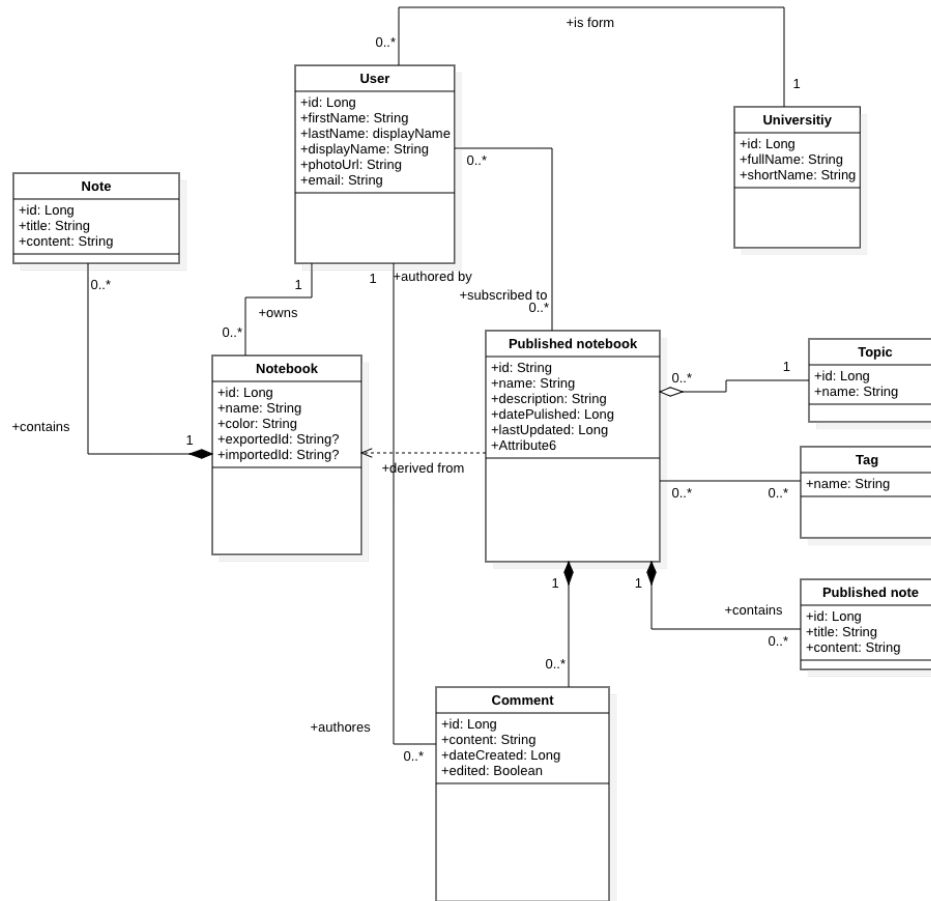


Figure 1.6: Domain Diagram

### 1.9.1 User

The User entity represents someone who has completed registration flow using one of the client applications or created manually (in case of Admin role). This entity contains such properties as: role, first name, last name, email address, password, university. Due to the fact, that StudyPad provides several ways for the user to authorise, some of the properties will either come from the user's input or from the 3rd party API (such as Firebase).

### 1.9.2 Note

Note represents a single piece of information. It consists of two properties: title and content. These can be described as term and definition or question and answer. Each note must be assigned to one of the notebooks, hence there is a 1:N relation.

### 1.9.3 Notebook

Notebook is one of the main entities used in the application flow, and can be created by an authenticated user. The soul purpose of the Notebook is to store Notes and serve as a source for a Published Notebook. Properties name and colour are used to help users distinguish between different Notebooks.

### 1.9.4 University

University represents a school, where the User can assign himself as a student during registration flow. It is used to unite users from the same school to make content searching easier.

### 1.9.5 Published Notebook

Published Notebook represents a shareable content. It can be created by user, based on one of his/her notebooks by providing some additional details: name, optional description, topic and optional list of tags. All these properties are later used in a searching flow to optimise results.

### 1.9.6 Topic

Topic represents the main topic or subject of the Published Notebook. Topic contains only one property – its name

### 1.9.7 Tag

Tag is a short label attached to the Published Notebook. It is mainly used to narrow the topic or school. Tag has only one property – its actual value stored as the name.

### 1.9.8 Comment

Users can comment on published notebooks. Most of the properties are assigned automatically, with the only exception being content: it represents the body of the comment and assigned by the user.

## 1.10 Requirements

It is important to establish all functional and non-functional requirements for StudyPad. The section bellow contains all requirements designated before the start of the development.

### 1.10.1 Functional Requirements

#### User Authentication

- **F1: Registration/Login using email** Access to StudyPad is possible by creating an account using an email address/password combination.
- **F2: Registration/Login using Facebook** The user will be able to use his/her Facebook account to access StudyPad.
- **F3: Registration/Login using Google** The user will be able to use his/her Google account to access StudyPad.
- **F4: Store OAuth token** API Authentication Token will be stored in a device memory.
- **F5: Token refreshment** API Token will be refreshed when needed, so the user will not have to login again.
- **F6: University selection** As part of user registration flow, the user will be able to select his/her university.
- **F7: University Addition** The user will be able to add his/her school, in case of not finding it in a StudyPad database.

#### Library Management (Notes & Notebooks)

- **F7: Notebook creation** The user will be able to create new notebooks with the name he/she choose.
- **F8: Notebook deletion** The user will be able to delete existing notebooks.
- **F9: Notebook name edition** The user will be able to edit notebooks names.
- **F10: Note creation** The user will be able to create a note with a specific title and content.
- **F11: Note edition** The user will be able to edit an existing note, or completely delete it.

- **F12: Show Notebooks:** The user will be able to view all the notebooks he/she created.
- **F13: Show Notes:** By clicking on notebook item, the user will be able to view the list of notes that are assigned to this notebook.
- **F14: MathJax/ASCII Math support:** User will be able to use complex mathematical expressions as notes content

### Sharing Hub

- **F15: View published notebooks** The user will be able to view notebooks published by other users.
- **F16: Search through published books** The user will be able to search through the published notebooks by applying different filters (such as author, university, and subject/topic).
- **F17: Browse through published notebook** User will be able to see notes inside the notebook that has been published.
- **F18: View comments** User will be able to view others users comments discussing a notebook that have been published.
- **F19: Leave a comment** The user can comment on an other user published notebook.
- **F20: Delete a comment** The application will allow the user to delete his/her comment.
- **F21: Edit a Comment** The application will allow the user to edit his/her comment
- **F21: Save published notebook** User will be able to save published notebook to his/her library.
- **F22: Publish notebook** User will be able to publish his/her notebook.
- **F23: Update published notebook** The Author of the published notebook will be able to update its information.
- **F24: Delete published notebook** The Author of the published notebook will be able to delete the his/her notebook from shared space.
- **F25: Share notebook** The user will be able to share his/her notebook by generating a deep-link.
- **F26: Notification on update** The user will be notified on updates to the Published notebook they have subscribed to.

- **F27: Suggestions** Subscribers will be able to suggest a change or correction to the Published Notebook. The author will be able to approve it, which will result in the update of the Published Notebook, or decline it.
- **F28: Copy Published Notebook** The user will be able to copy a published notebook. This will create a standalone copy of this notebook in the user's library.

### Study Hub

- **F29: Start a Flashcards walkthrough** The user will be able to use an interactive method of looking through his/her notes.
- **F30: Start a written test** The user will be able to participate in a written test based on one of the notebooks to test his/her knowledge.
- **F31: Start a self-check** The user will be able to participate in a quiz challenge that will be based on one of his/her notebooks

### Profile

- **F32: View Profile Information** The user will be able to view his/her profile information such as first name, last name, and university.
- **F33: Edit Profile Information** The user will be able to edit his/her profile information.
- **F34: Logout** The user will be able to logout from the application.

#### 1.10.2 Non-functional requirements

- **N1: Native Android application** The application will be written using native Android SDK.
- **N2: Android Version** The application minimal SDK version must be low enough to support as many devices as possible and high enough to use most applicable Android APIs considering other functional and non-functional requirements.
- **N3: Material Design** The application user interface will follow the latest Material design guidelines and best practises.
- **N4: Scalable app architecture** The application's architecture must be scalable and easy testable.
- **N5: App Localisation** The application will be able to adapt to different languages based on user locale.

## 1.11 Existing solutions

There are several services out there whose goal is similar to StudyPad. However, most of those solutions are specialised in language-learning and have limited sharing and/or searching options. Each application/service will be reviewed separately and will include a requirements and main processes comparison.

Symbols are used in Tables to indicate requirement availability in the applications. Symbol on Figure 1.7a represents the functionality that is fully supported by the application. Symbol on Figure 1.7b demonstrates that the given functionality is either limited/incomplete or not working as intended. Lastly, symbol on Figure 1.7c indicates that the given application does not support the requirement.



Figure 1.7: Functionality Symbols

### 1.11.1 StudyBlue

**StudyBlue** not only partially shares a name with StudyPad but also a goal and a wide range of features. It is clearly one of the biggest rivals on the market and it is very important to determine what StudyBlue does right and what it does not. How most important StudyPad requirements are compared to the StudyBlue functionality can be seen on Table 1.1

Notebooks here are called *decks* and notes are represented by *cards*. Moreover, all decks in StudyBlue are stored in specialised folders named Classes and Interests. Interest, as an entity, not only stores decks, but also unites them by a subject or a topic (e.g. Spanish, Literature). Being a student of the university, the user can also create an Interest that will be connected to his/her school – Class. Both Interests and Classes are split into two parts: shared space, where the user can find all the decks created by other users, and personal space, where the user can keep the decks he/she has created, saved or copied.

Table 1.1: StudyPad &amp; StudyBlue requirements comparison

Requirements / Application name	StudyPad	StudyBlue
F6: University Selection	★	★
F13: Show Notes	★	☆
F14: MathJax /ASCII Math support	★	☆
F16: Browse Published Notebooks	★	★
F17: Searching Fo Published Notebooks	★	★
F18-F21: User Commentaries	★	☆
F27: User Suggestions	★	☆

- **Library management:** The main difference in library management comes with the fact that each deck must be associated with an Interest or Class, moreover, it is not possible just to create a deck, the user has to create at least two cards first. Because of that, library management not only includes managing your decks and card, but also interests and classes. Everything the user has saved or created, whether class, interest or a deck can be accessed via *Backpack*.
- **Publishing:** The Publishing process is quite different compared to what StudyPad is trying to achieve. When creating a deck, the user can choose whether they want to make his/her deck visible for other users in this Class/Interest or to make it private. It is not possible to collaborate on a deck or suggest a change.
- **Importing:** As mentioned before, user can save a deck from an existing Interest/Class to their personal space, but they will not be able to make any edits without making a copy.
- **Discovering:** The fact that all decks are either assigned to an Interest or a Class makes searching quite simple. The user can search for deck, classes and users by its name. All these searching options are separated in the UI, so there is no way of combining them. Searching based on the Interest is also available, but the user has to add this Interest to his/her “Backpack”.
- **UI & UX:** StudyBlue has been in development roughly since 2009, and as for now, UI looks outdated and UX can be greatly improved. The most common UI issue here – Inconsistent usage of UI elements styles.

### 1.11.2 Quizlet

**Quizlet** is primarily used for learning languages, from where most of the limitations come from. The closest analogy to a Notebook here is a Study set with Terms inside. Being an application for language learners makes the

## 1. STATE-OF-THE-ART

---

process of generating different tests and quizzes quite easy, and this is where this application shines the most. Table 1.2 shows requirements comparison between StudyPad and Quizlet.

Table 1.2: StudyPad & Quizlet requirements comparison

Requirements / Application name	StudyPad	Quizlet
F6: University Selection	★	☆
F13: Show Notes	★	☆
F14: MathJax /ASCII Math support	★	☆
F16: Browse Through Published Notebooks	★	★
F17: Search Through Published Notebooks	★	☆
F18-F21: User Commentaries	★	☆
F27: User Suggestions	★	☆

- **Publishing:** Similar to StudyBlue, all Study Sets are visible to everyone else by default. However, Quizlet provides a wider variety of privacy options. Study Set can either be completely public, private or semi-private using a password protection. Password protection can also be used to allow certain users to modify a study set.
- **Importing:** Importing flow allows user to either copy or save the study set to a specific folder. This flow may confuse some users, because only copy allows the user to actually add a study set to their library and modify it. Saving study set to the specific folder only saves the link to it and splits library management in two parts.
- **Discovering:** This limitation comes from the fact that Quizlet is an app for studying foreign languages. As a consequence, the only distinctions between Study Sets are its name and its language. These are the only two options available when searching through study sets.

### 1.11.3 Cram

**Cram** is very similar to Quizlet but seems highly outdated in terms of UX/UI and brings some sharing limitations to the table. Table 1.3 shows a requirements comparison between StudyPad and Cram.



Table 1.3: StudyPad &amp; Cram requirements comparison

Requirements / Application name	StudyPad	Cram
F6: University Selection	★	☆
F13: Show Notes	★	★
F14: MathJax /ASCII Math support	★	☆
F16: Browse Through Published Notebooks	★	★
F17: Search Through Published Notebooks	★	☆
F18-F21: User Commentaries	★	☆
F27: User Suggestions	★	☆

- **Publishing:** Content publishing is similar to Quizlet – All sets are either visible by other users or not. Sharing a deep-link to a single study set was not functional at the time of writing this section.
- **Discovering:** Searching for content in Cram is even more limited compared to Quizlet, only the name of the study set is used
- **Importing:** Library management here is split in three parts: User personal sets, Favourite sets and Recently studied. When searching, there is no way to save a published study set to a personal library, however, it will be automatically saved to Recent section, or the user can add it manually to Favourites. It is not possible to make any local edits.

#### 1.11.4 TinyCards

**TinyCards** is a flash-card application made the same developer that created Duolingo – one of the biggest language-learning applications in the market. TinyCard is meant to be more generic as it allows users to create custom study sets, often not limited to languages. Table 1.4 shows how the StudyPad requirements compare to the TinyCards functionalities.

Table 1.4: StudyPad &amp; TinyCards requirements comparison

Requirements / Application name	StudyPad	TinyCards
F6: University Selection	★	☆
F13: Show Notes	★	★
F14: MathJax /ASCII Math support	★	☆
F16: Browse Through Published Notebooks	★	★
F17: Search Through Published Notebooks	★	☆
F18-F21: User Commentaries	★	☆
F27: User Suggestions	★	☆

## 1. STATE-OF-THE-ART

---

- **Importing:** Similar to Cram, it is not possible to edit the study set the user has downloaded and saved to their library
- **Challenges:** Tests are generated automatically and there is no way to choose the test type.

---

# Design

## 2.1 Material Design

Material Design is a design language that Google developed in 2014. Later in 2018, it was revamped with a goal to provide more flexibility for designers to create more custom themes. Material Design was chosen as a main source of inspiration when creating StudyPad UI due following reasons:

- Material Design has very wide support on Android Platform. Every component that is described in Material Design Guidelines is implemented by Google and can be used by developers.
- Material Design is cross-platform, meaning that its guidelines can easily be applied when creating design for other platforms like iOS or Web.

### 2.1.1 Used Components

This section contains a list of Material Design specific Components (not including widely used elements such as Buttons and Input Fields) used when creating design for StudyPad.

- **Floating Action Button (FAB):** According to Material Design Guidelines: *“A floating action button (FAB) performs the primary, or most common, action on a screen. It appears in front of all screen content, typically as a circular shape with an icon in its center.”* [32] FAB is widely used in creating content, hence it found its place in the note-taking part of the application.
- **Bottom App Bar:** *“Bottom app bars provide access to a bottom navigation drawer and up to four actions, including the floating action button.”* [33]. In StudyPad it is primarily used in the details screens with one primary action (FAB) and two or more secondary actions.

- **Chips:** “*Chips are compact elements that represent an input, attribute, or action.*”[34]. In StudyPad it is used mainly for displaying Tags and in Searching flow. In most of the cases it will be placed in the ChipGroup – a special container to store dynamic amount of Chips
- **Bottom Sheets:** “*Bottom sheets are surfaces containing supplementary content that are anchored to the bottom of the screen.*”. The most common Bottom Sheet type used in StudyPad is Modal Bottom Sheet, that allows user to provide certain kind of input, like Topic, Univerisity selection e.t.c

## 2.2 UI & UX

This section contains most important screens description from UI & UX standpoint, wireframes and it’s iterations

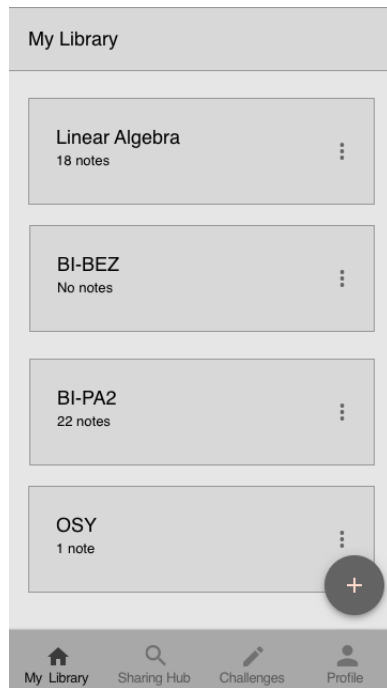
### 2.2.1 Library Managent

Library management requirements (F7-F14) are covered by the first top-level destination: **My Library**. This part of the application serves as the main entry point for the returning users. Wireframes of this sections can be viewed on Figure 2.1.

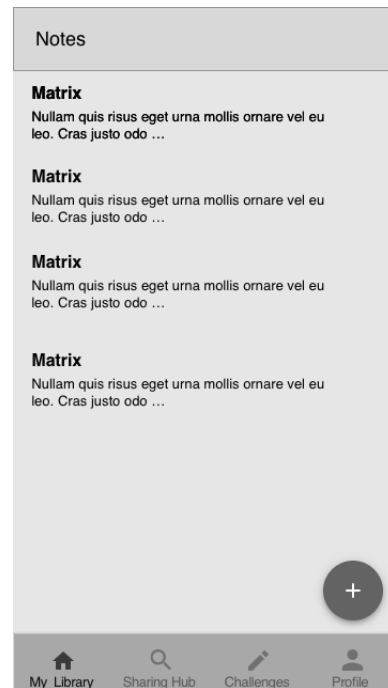
Its first screen, as shown on Figure 2.1a, contains a scrollable list of Notebooks and provides access to the various notebook-management related actions such as: Notebook creation(triggered by FAB), edition, deletion and sharing

By clicking on one of the notebooks in the list, user will be transferred to the Notes List Screen (Figure 2.1b). Similar to Notebooks List Screen, this screen contains a scrollable list of notes from this notebook and covers all the note management related actions and interactions such as Note creation, edition and deletion. Once again, FAB is used for to trigger create action.

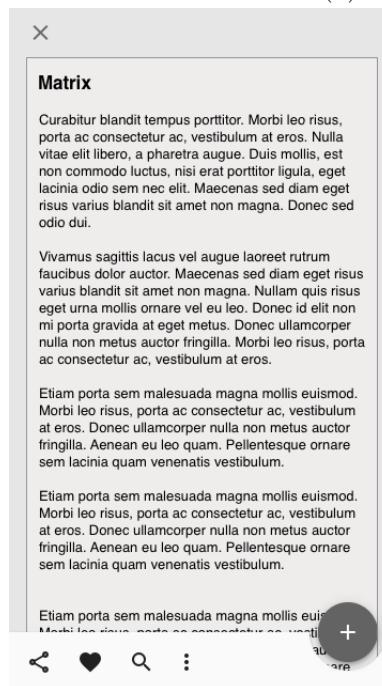
Finally, by choosing a note, the user will be able to view its details. As shown of Figure 2.1c, this screen utilises Bottom app bar to provide access for key actions and FAB.



(a) Notebooks List Screen



(b) Notes List Screen



(c) Note Detail Screen

Figure 2.1: My Library Section Wireframes

### 2.2.2 Notebook Publishing

The fact that all the content in StudyPad is made by its community makes Notebook Publishing Flow one of the core functionalities of the application. The flow itself acts as a complex form with different types of inputs:

- Notebook name – Single-line input field, will be pre-filled with the name of local Notebook that is being published.
- Language of the Notebook – Selector, will be pre-filled with the current user locale.
- University/School associated with this Notebook – Selector, will be pre-filled with the User's university, this field is optional.
- Topic/Category of the Notebook – Selector, this field is mandatory
- Tags – ChipGroup, chosen tags will be represented by Chips. The field is optional
- Description/Message for other users – Multi-line input field, this field is optional.

Considering an amount of input needed, its variety, the decision was made to split this form into several steps. It will allow making the form more dynamic and put more emphasis on some inputs by grouping them.

First step will include all fields that can be pre-filled: Notebook's name, language and University. This will allow to skip this step in most cases, making it only about controlling pre-filled information. Second step will require users to provide search-relevant information, such as Tags and Topic/Category, topic being a required option. Third step will allow user to provide a message or a more detailed description of the Notebook. Fourth step will be all about confirmation, in which the user can double-check information he/she entered and submit it.

Though the choice of UI elements in different steps is well defined, the way we present these steps is not. Simplified wireframes with different approaches of how the steps can be presented are shown on Figure 2.2. In the end, decision to go with the Stepper solution was made as it gives a number of advantages:

- Stepper is a standardised element of Material Design
- In every step, Stepper gives an overview of all steps available and its completion state
- Stepper allows to quickly navigate between different steps.

More detailed wireframes for this flow using Stepper are shown in Figure 2.4

## 2.2. UI & UX

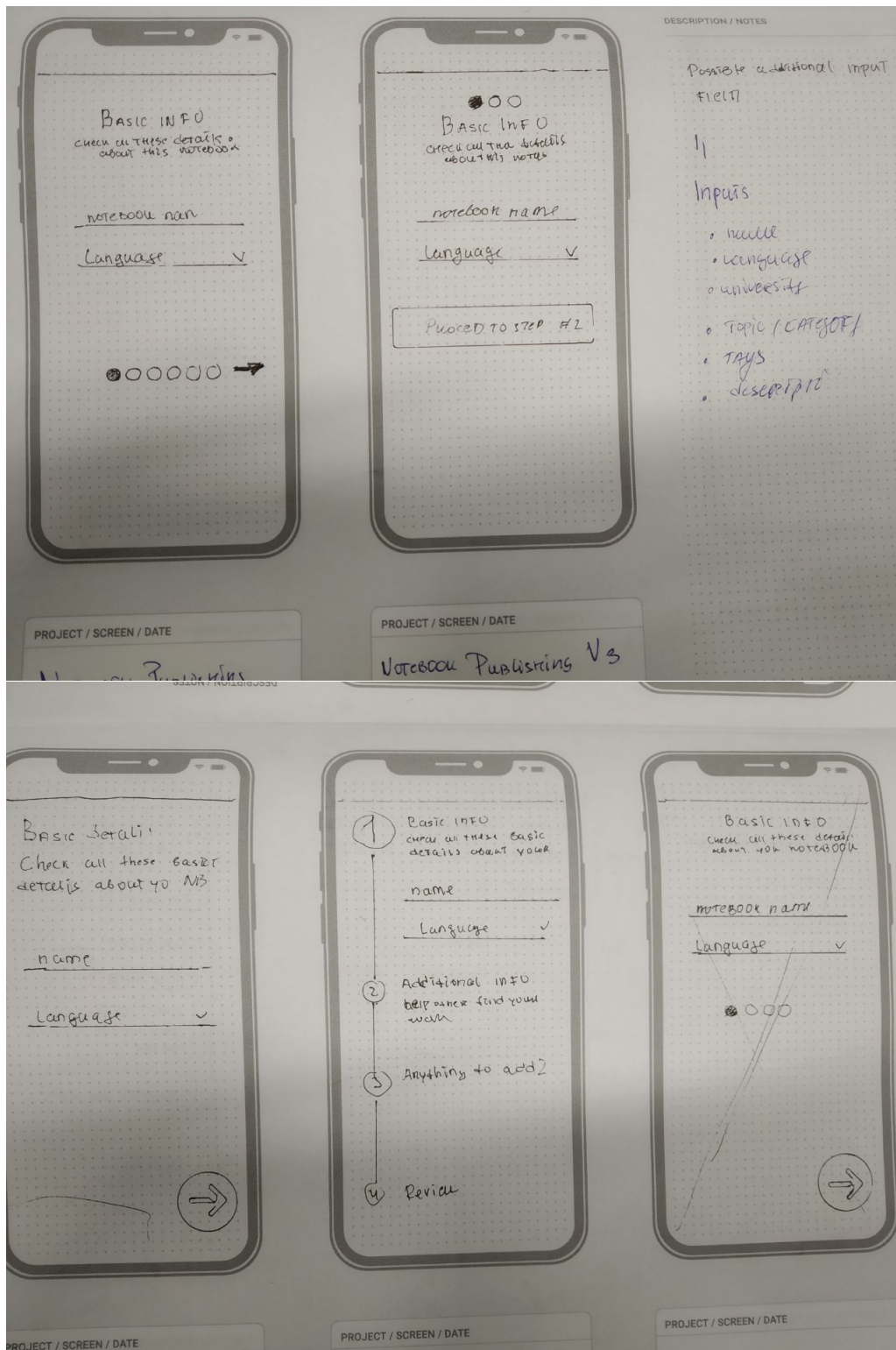


Figure 2.2: Notebook Publishing Flow - Sketch Phase

## 2. DESIGN

---

1 Basic details

Linear Algebra

Czech Technical University

English

CONTINUE

2 Additional details

3 Description

4 Confirmation

This wireframe shows the first step of the notebook publishing process. It features a vertical progress bar on the left with four steps: '1 Basic details' (active), '2 Additional details', '3 Description', and '4 Confirmation'. The main content area contains three input fields: a text field with 'Linear Algebra', a dropdown menu with 'Czech Technical University', and another dropdown menu with 'English'. A blue 'CONTINUE' button is positioned below the input fields.

✓ Basic details

2 Additional details

English

Tags

+ Enabled Action Tag 1

Tag 1 Tag 1 Tag 1 Tag 1

CONTINUE

3 Description

4 Confirmation

This wireframe shows the second step of the notebook publishing process. The progress bar now has '1 Basic details' completed (marked with a checkmark) and '2 Additional details' active. The main content area includes a dropdown menu with 'English', a 'Tags' section with a '+ Enabled Action' button and four 'Tag 1' tags, and a blue 'CONTINUE' button. The 'Description' and 'Confirmation' steps remain in the progress bar.

(a) First Step of Notebook Publishing (b) Second Step of Notebook Publishing

✓ Basic details

✓ Additional details

3 Description

English

CONTINUE

4 Confirmation

This wireframe shows the third step of the notebook publishing process. The progress bar now has '1 Basic details' and '2 Additional details' completed (marked with checkmarks), and '3 Description' active. The main content area features a dropdown menu with 'English' and a blue 'CONTINUE' button. The 'Confirmation' step remains in the progress bar.

(c) Third Step of Notebook Publishing

Figure 2.3: Notebook Publishing Wireframes



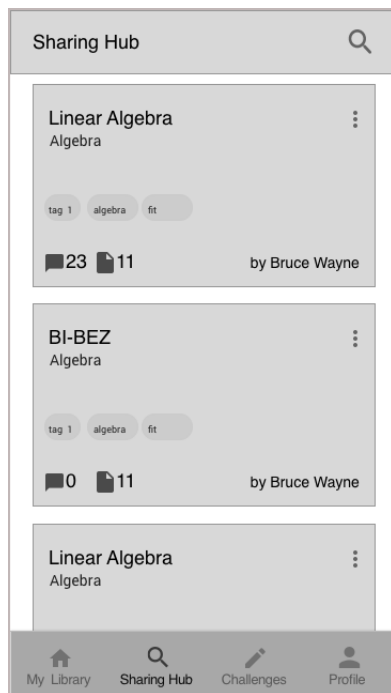
### 2.2.3 Sharing Hub

**Sharing Hub** is another top-level destination, that allows users to access various Notebooks published by other users.

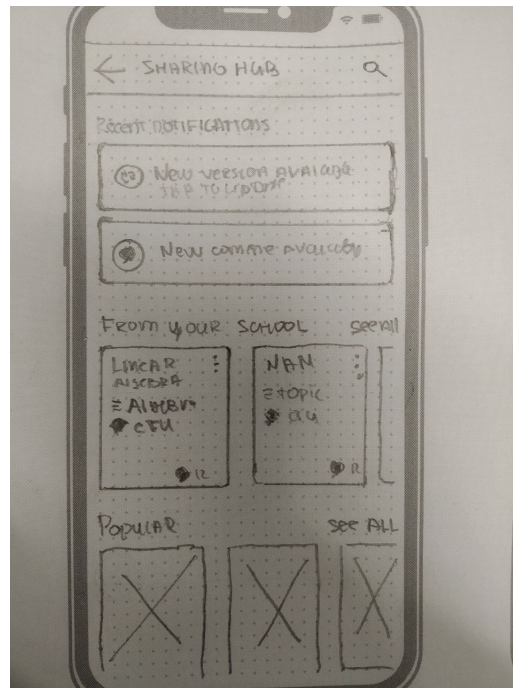
In the early stage of the StudyPad development, Sharing Hub root screen (Figure 2.4a) contained only one list with the Published Notebooks, with the idea of showing the most recently added Notebooks. Later on, the decision was made to make this screen more personalised by introducing Sections and Notifications preview (Figure 2.4b).

Section is a small list of Published Notebooks of fixed size, that can be scrolled horizontally. By introducing these small lists it will be possible to group Published Notebooks into personalised sets (Notebooks from the user's university for example). Each Section is result of an executed query, so it can be used to facilitate the search flow. So, clicking the See All Button, will launch Search Flow with pre-filled search options, associated with this section.

Another newly added element is a Notifications preview. Located on the top part of the screen, this element displays notifications about the most recent activities like Notebooks updates. By clicking the notification user will be to view the Notebook details associated with this notification.



(a) Landing page v1



(b) Landing Page v2

Figure 2.4: Sharing Hub Wireframes

### 2.2.4 Searching Flow

One of the main responsibilities of this screen is to start Search flow. Search flow is either launched by a Search Menu Item located in the Top bar or by the expanding on expanding on the Notebooks Section. Here a list of some basic requirements while

- There are should several primary searching/filtering options available: Notebooks name, University/School, Topic/Category and Tags.
- All these options must be easily accessible with the idea that more searching options might be introduced during later phases of development.
- When none of searching options are active, empty state must be presented.

The resulting wireframe is shown on Figure 2.5 . This approach is using Top App bar to display active searching options. Each options is represented by Chip action, that can have two states: Chip is either inactive and displays the name of the search option, or it is active, displaying selected option's values. When there are a lot of values selected, in case of multiple Tags for example, last three selected will be shown.

Clicking the Chip will result in showing Modal Bottom Sheet, that will allow user to tweak selection of the active values and every change to a Search options values will result in updating the search results. Search results themselves appear in the scrollable list below the Top Bar.

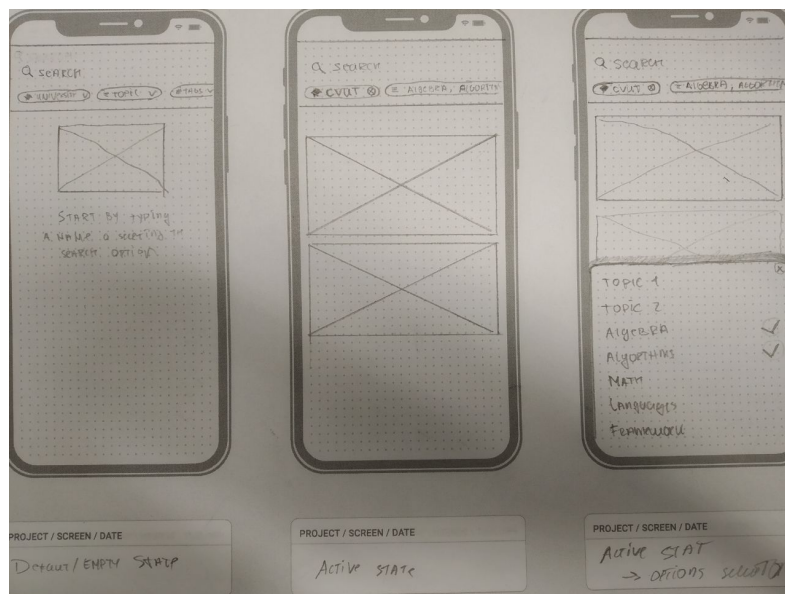


Figure 2.5: Search Flow Wireframes

### 2.2.5 Published Notebook Details

Published Notebook, as an entity, contains a large amount properties and there are a lot of actions associated with it. This screen will cover all actions regarding user comments (requirements F18-F21) and various Published Notebook actions such as saving, sharing, and copying. Including all these actions and flows into one screen would make UI very crowded, resulting in bad UX. *Tabs* can help solve this issue quite easy.

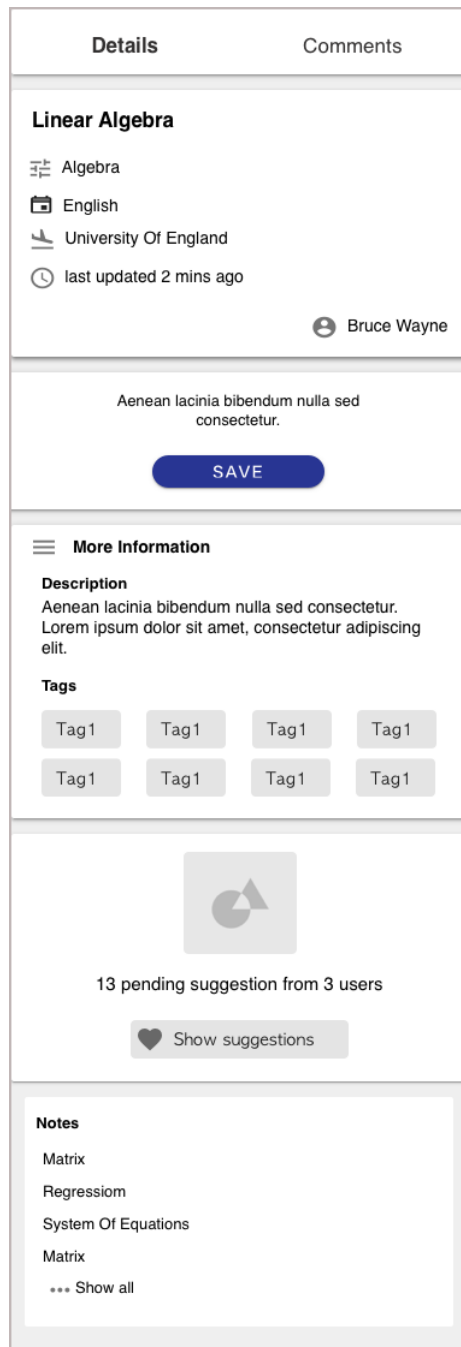
According to Material Design Guidelines, Tabs can organise content across different screens, data sets, and allow navigation between groups of content that are related and at the same level of hierarchy. [35]. By using Tabs, this screen can be split into two distinct sections: Detail Tab and Comments Tab, thus dividing responsibilities and making this screen more responsive.

Details Tab (Figure 2.6a) features key details about this Notebook. Due to the fact that there are a lot of details to display, all the properties are divided into several groups.

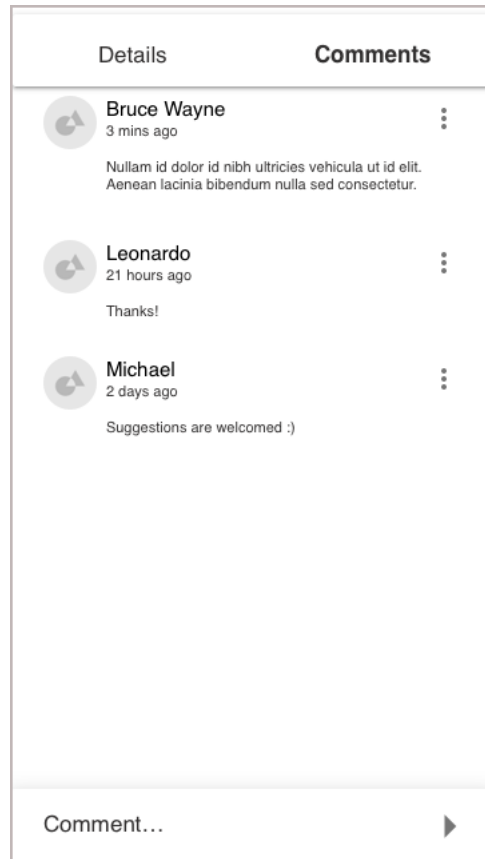
- **Basic details:** This group contains simple properties such as Notebook's name, its author, category/topic, language and associated university.
- **Description:** This group contains more dynamic content such as description, that can take multiple lines, and a Chip Group with tags.
- **Notebook Action:** This view represents a primary action available for this notebook in regards to user Library. This can either act as a Save button, that will save the latest Notebook version to the users Library or as an Update Button, that will result in applying changes from local notebook to the Published one, thus updating its contents. If none of the actions are available, this view will be hidden.
- **Suggestions:** This view group serves as preview and an entry point for suggestions left by other users.
- **Notes Preview:** Last view at the bottom of the screen serves as an entry point for notes browsing.

Comments tab, as shown on Figure 2.6b, contains a scrollable list of the user comments and an Input Field, that will be used to create a new comment, or edit the old one. Also, every comment contains an options button that will trigger either an edit or a delete action.

## 2. DESIGN



(a) Details Tab



(b) Comments Tab

Figure 2.6: Published Notebook Details Wireframes

### 2.2.6 Challenges

The current iteration of StudyPad provides a small amount of very straightforward challenges (or tests) where user can test his/her knowledge of one of the notebooks. **Challenges** section serves as an entry point for setting up these different tests.

Its landing screen (Figure 2.7a) features a list of recently completed challenges, including some statistics if available, and three buttons, that are responsible for launching three distinct challenges with add: Flashcards, Write and Self-check. There is also a button that will navigate the user to a more in-depth configuration of his/her trial - Setup Challenge screen. Clicking one of the recently completed challenges entries will launch this challenge with all the options pre-set automatically. If the user hasn't completed any challenge yet, the empty view will be featured.

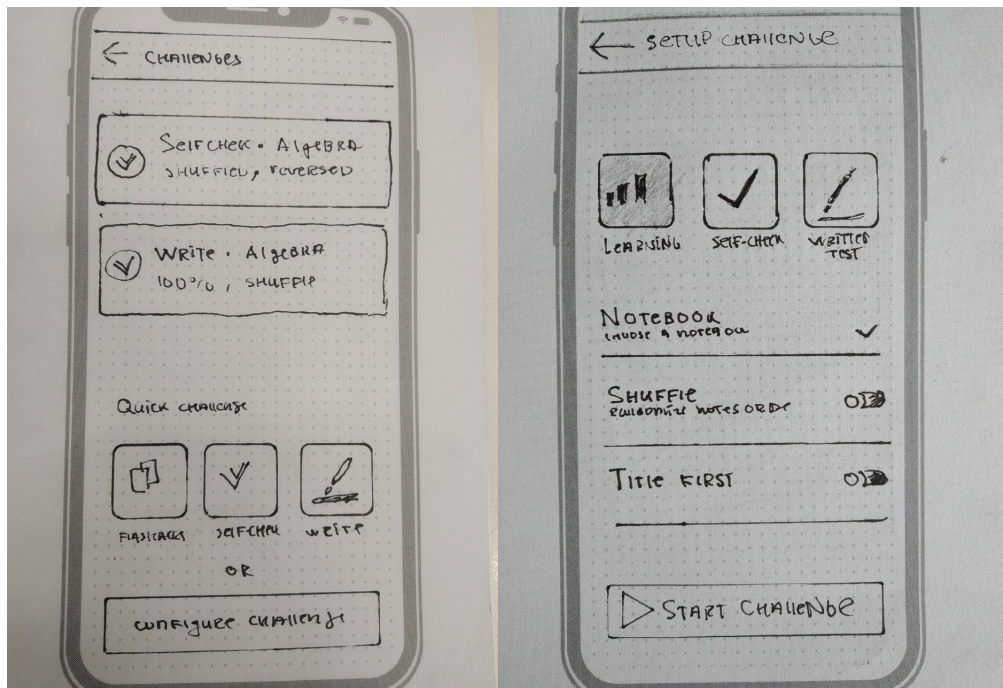
Besides choosing the type of the challenge and the notebook, Setup Challenge screen (Figure 2.7b) allows the user to select some additional options before generating his test such as order of the question and what will serve as an answer (either title of the note or its content). After all the details required are provided, most importantly challenge type and the notebook, the user will be able to start his test.

Flashcard challenge is one of the simplest tests and requires no validation. Wireframe of this challenge can be seen in Figure 2.8c. This challenge is just an interactive way to walk through the notes, featuring a scrollable list of notes. User can flip the displayed flashcard to show the answer to a question and back. Navigation here doesn't have any boundaries so that the user can scroll this in any direction. When reaching the end of the list, the user will see how many notes he/she studied.

Just like the Flashcards challenge, next test features a list of questions based on the users' notebooks. The idea of Self-check challenge ?? is straightforward. Given a list of questions, the user must ask himself a simple question: do know the answer? Two buttons below the flashcard are then self-explanatory: It's the user answer, either he/she know the answer or not. Clicking will "I know" will reveal the next question and clicking "Study again" will reveal the answer to the current question.

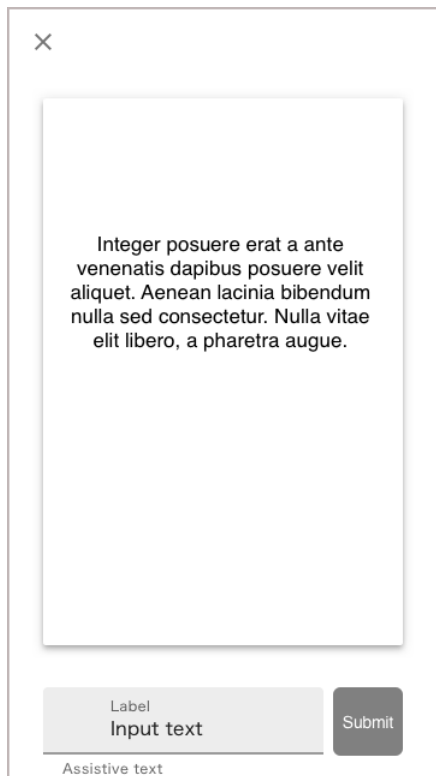
Write challenge (Figure 2.8a) is very similar to the Self-check challenge with the only distinction being that the user must provide his answer using an input field. His answer will then be validated, and, depending on how close user was to the correct answer, it will either show next question or will require the user to re-submit his answer, correcting his/her mistake.

## 2. DESIGN



(a) Challenges Landing Screen Wireframe

(b) Setup Challenge Wireframe



✕

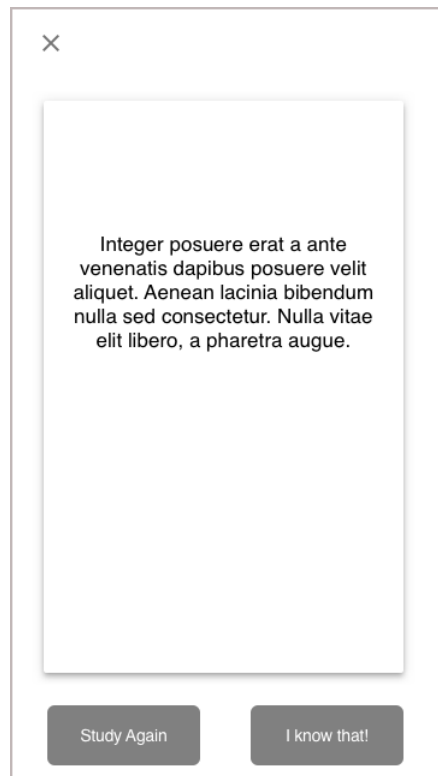
Integer posuere erat a ante venenatis dapibus posuere velit aliquet. Aenean lacinia bibendum nulla sed consectetur. Nulla vitae elit libero, a pharetra augue.

Label  
Input text

Submit

Assistive text

(a) Write Challenge Wireframe



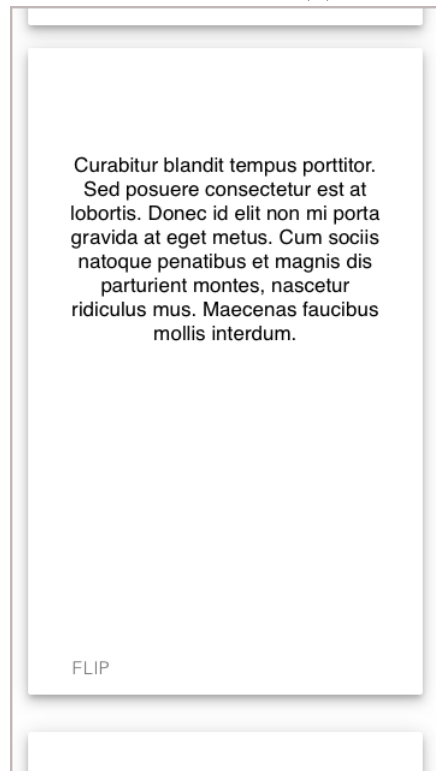
✕

Integer posuere erat a ante venenatis dapibus posuere velit aliquet. Aenean lacinia bibendum nulla sed consectetur. Nulla vitae elit libero, a pharetra augue.

Study Again

I know that!

(b) Self-check Challenge Wireframe



Curabitur blandit tempus porttitor. Sed posuere consectetur est at lobortis. Donec id elit non mi porta gravida at eget metus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Maecenas faucibus mollis interdum.

FLIP

(c) Flashcards Challenge Wireframe

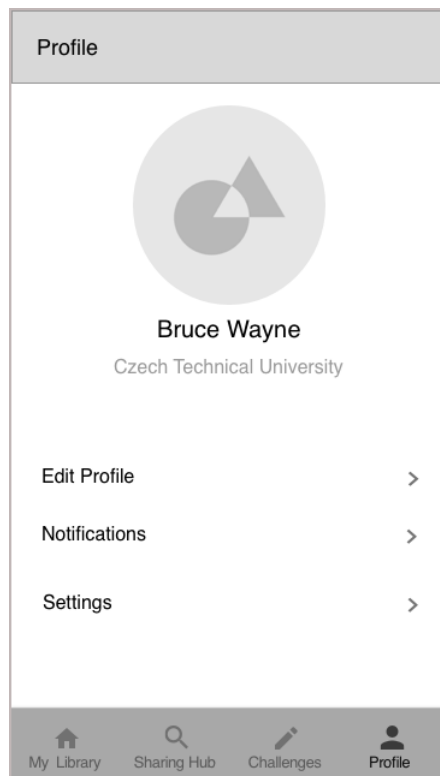
### 2.2.7 Profile

**Profile** is the last top-level destination available from the Navigation bar. As shown on Figure 2.9a, it features basic information about currently logged-in user and provides buttons to access three more subsections: Profile Edition, Notifications and Settings.

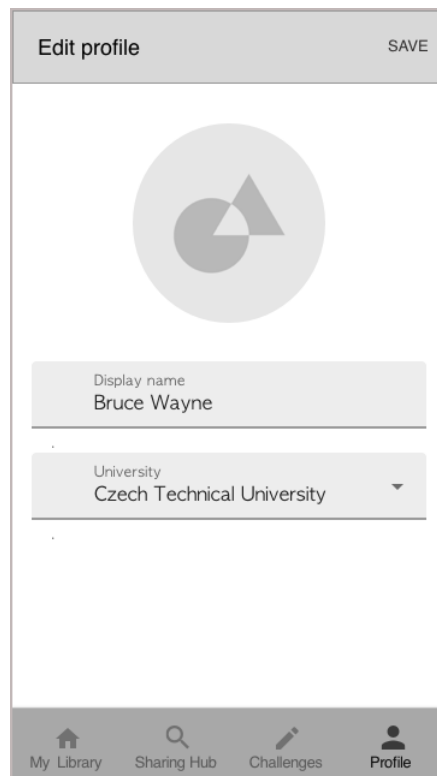
Profile edition screen (Figure 2.9b) allows user to change his/her name and university. Screen contains two input fields, that will be automatically populated with user information, and a Save button, that will only be enabled if one of the input field changes. Pressing Save button will try to update information and navigate back to profile screen in case of success.

Notifications screen contains a scrollable list of all previously received notifications. It is the same component used in the Sharing Hub screen.

Settings screen contains application-relevant information and preferences. On of the key functionalities here is a Logout button (requirement F34).



(a) Profile Screen Wireframe



(b) Edit Profile Screen Wireframe

## 2.3 Application architecture

One of the key requirements for StudyPad development is its architecture scalability. This section describes the approach chosen for its development and



patterns in use

### 2.3.1 MVVM

MVVM architecture was a rather easy choice for main architecture pattern. It is well supported by Android libraries and recommended by Google. View-Model serves as glue between Presentation Layer and Data Layer. In example provided by Google (Figure ??) Fragments/Activities represents the Presentation Layer and Repository acts as the data layer.

### 2.3.2 Repository

Repository is an abstraction that deals with the Entities. Repository can have one or multiple data sources. One of the main StudyPad data sources is its REST API. In-memory database will act as a secondary data source to allow some offline capabilities.

### 2.3.3 Interactors

Though the repository pattern is a great abstraction for data layer, it would make sense to go even further and apply more Clean architecture principles. Interactor, or a Use-case, is another abstraction that represents single use-case – one rule of business logic. Interactor is usually talks to the repository and the returns the result to whatever called it in the first place.

This will even more propagate the separation of concerns as every Interactor will only deal with one particular functionality, which will make code more readable and make uni-testing easier in the future.

### 2.3.4 Dependency Injection

Dependency injection (DI) is another technique that will help provide a more clean code and will be great help when providing a test-coverage.

### 2.3.5 StudyPad Architecture

Overall architecture will include following components:

- **Presentation Layer**, that is represented by **Fragments** and **Activities**. They are only aware about how to display information and does not contain any complex logic. All the interactions are getting handled by Domain Layer
- **Business Layer** is represented by **ViewModel**. **ViewModel** is injected into the Presentation Layer, it handles all user interactions and decides what and when to display, including navigation. **ViewModel** also executes **Interactors** and handles its result.

## 2. DESIGN

---

- **Data Layer** is represented by Interactors and Repositories. Interactors get injected into ViewModel and serves to execute certain request in the background and provide a result via callbacks. Repositories provide Entity-based operations using the data sources.

---

# Implementation

This chapter describes main pin points of StudyPad implementation, platform-specific technologies and libraries.

## 3.1 Used Libraries and Technologies

### 3.1.1 DI Framework

There are a few DI frameworks available for Android, **Dagger** being one of the most popular. Dagger is a fully static Java DI framework that uses annotations for dependencies resolution. Dagger got huge support from the developers' community, mainly because of its performance. All dependencies are resolved and generated during the building phase, thus making the application perform faster. But, Dagger is pretty old, complex and requires a lot of boilerplate code to function properly. Introduction of Kotlin paved a way for new libraries

**Koin** is a lightweight dependency injection framework written in pure Kotlin. Koin uses DSL (Domain Specific Language) for modules declaration and, as opposed to Dagger, it is very easy to use. And one of its most significant advantages: It provides **ViewModel** support out of the box us by using just a **ViewModel** keyword. Listings 3.2 and 3.1 shows the example of declaring and injecting **ViewModels** using Dagger and Koin respectively. This example shows quite well how easy it to use Koin comparing to Dagger.

Because of its great features, simplicity and the desire to make this application as modern as possible, Koin was chosen as the DI framework.

### 3. IMPLEMENTATION

---

Listing 3.1: ViewModel Injecting using Koin

```
// Define the dependency
viewModel { LoginViewModel(get(), get(), get(), get()) }

// Inject into the fragment/activity
private val viewModel: LoginViewModel by viewModel()
```

Listing 3.2: ViewModel injecting using Dagger

```
@Binds
@IntoMap
@ViewModelKey(MainActivityViewModel::class)
abstract fun bindViewModel(
    viewModel: MainActivityViewModel): ViewModel

@ContributesAndroidInjector
internal abstract fun mainActivity(): MainActivity

// Inject factory
@Inject lateinit var factory: ViewModelProvider.Factory

// Build viewModel
private val viewModel = ViewModelProviders.of(
    this, viewModelFactory)
    .get(VM::class.java)
```

#### 3.1.2 HTTP Client

Retrofit nowadays is pretty much an industry standard, and it is hard to argue with it. Retrofit provides a type-safe way to describe your HTTP API using Java/Kotlin interface and annotations. Annotations allow to m the requests options and parameters such as Request type, request body or query parameters. Retrofit also has support for some popular threading libraries like RxJava, and recently – Kotlin coroutines.

Listing 3.3: Retrofit Call Example

```
@PATCH("$API/notebooks/{id}")
fun updateNotebookAsync(
    @Path("id") id: String,
    @Body body: Body) : Deferred<NotebooksResponse>
```

### 3.1.3 Threading

Threading and background work has always been a problem in the world of Android development. Google provided some tools and APIs to solve this problem: `AsyncTasks` and `Loaders`. Both of them have a lot of issues, mainly because they are not very intuitive and not flexible to use. `AsyncTask` specifically, when not used properly, was the main cause for memory leaks.

Later on, `RxJava` gains popularity. `RxJava` is a library for composing asynchronous and event-based programs by using observable sequences. It is a very powerful tool, but the library itself is quite heavy (almost 2 MB), and when used only for threading it is often described as attempt to kill a spider with the bazooka.

Kotlin has its own way to handle background work. Kotlin 1.3 introduced Kotlin Coroutines. Coroutines can be described as light-weight threads and essentially its a small task that normal thread can execute. Kotlin uses a concept of **suspending** functions, that is a regular function with the **suspend** modifier which indicates that the function can suspend the execution of a coroutine. When it comes to StudyPad, Kotlin Coroutines are mainly used in the applications data layer: in `Interactors` and `Repositories`

### 3.1.4 Android Navigation Component

Navigation in Android can be done either by using `Activities` or `Fragment` transactions. `Android Architecture Components` introduced a new way to solve navigation – `Navigation Component`

`Navigation Component` serves as an abstraction layer above `FragmentManager` that is used to manage `fragment` transactions. `Navigation Component` handles a lot of things:

- Handles back stack management.
- Provides utility methods for commonly used UI Elements such as `ToolBar`, `BottomNavigationBar`, and e.t.c.
- Provides a type-safe way to pass arguments between destinations.

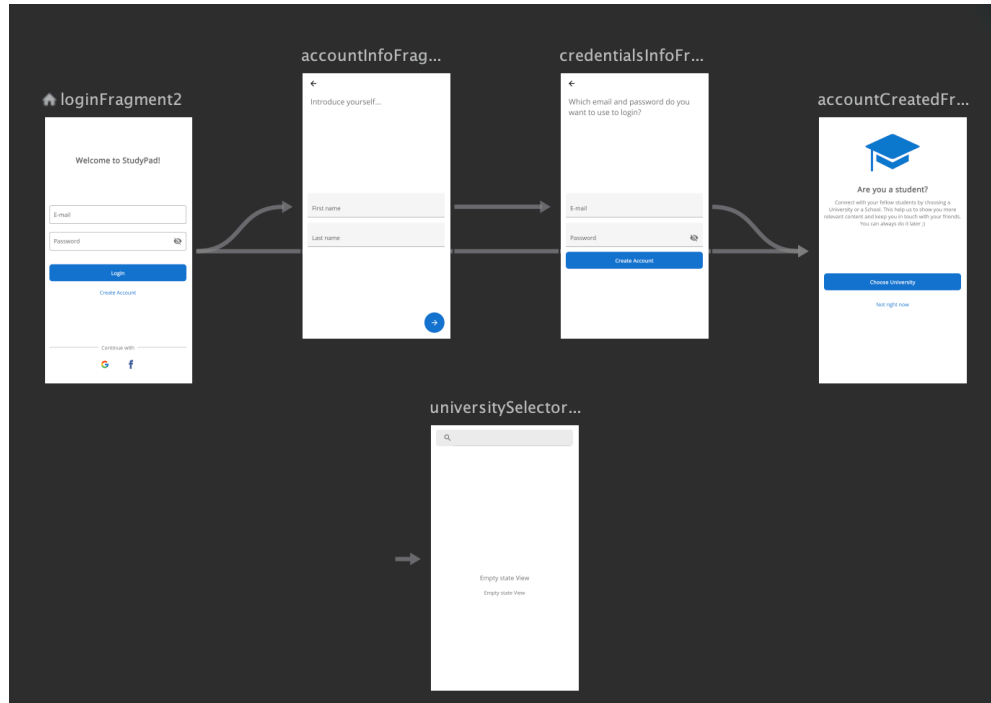
Navigation is described using XML, `Android Studio` also provides a visual editor for it to build your navigation graph(Figure 3.1). Navigation itself is handled by `Navigation Controller`, which resolves destinations available and does the `Fragment` transaction under the hood.

### 3.1.5 ViewModel

One of the essential parts of AAC is `ViewMode`. `ViewModel` is the implementation of the `ViewModel` pattern provided by Google as a part of AAC. One of the key features of the `ViewModel` – it can survive configuration changes.

### 3. IMPLEMENTATION

Figure 3.1: Navigation Editor

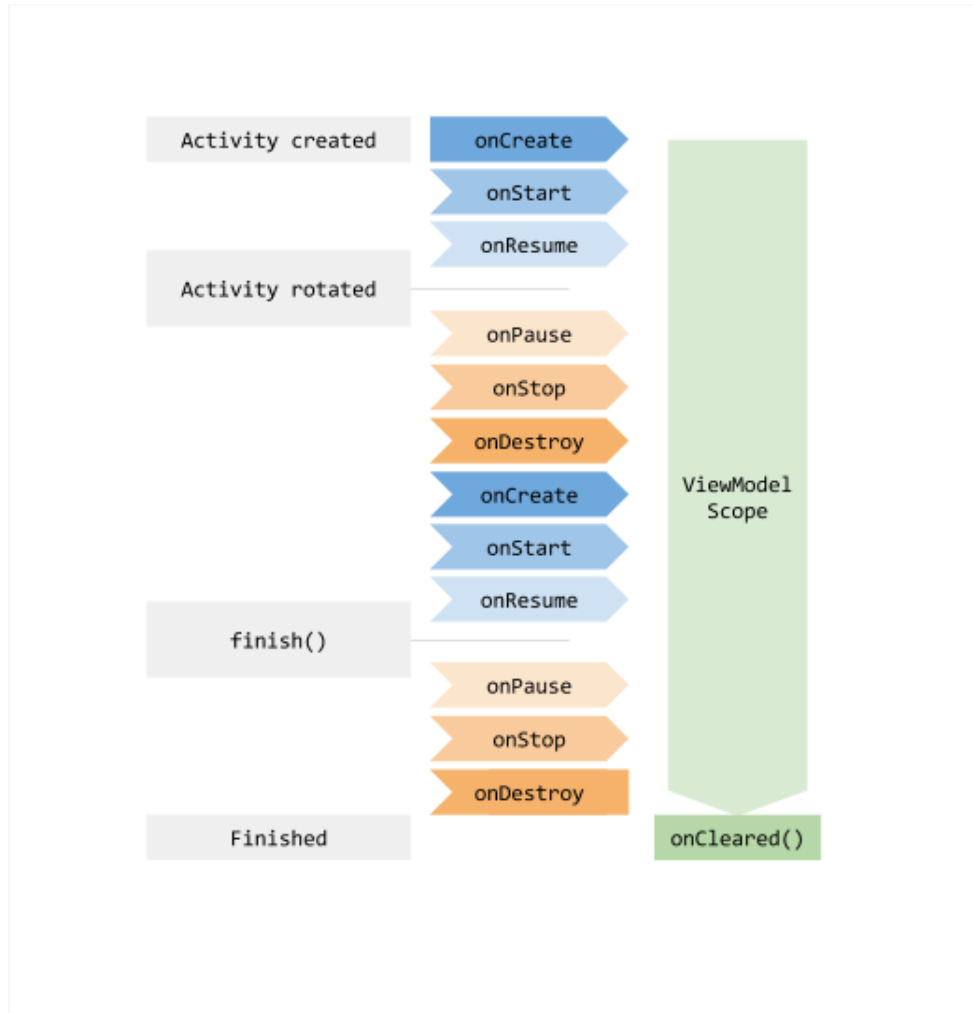


Activity's lifecycle is one of the most problematic concepts in Android Development, because every configuration change, such as phone rotation for example, destroys Activity and it must be recreated, thus causing a lot of issues. ViewModel lifecycle is different. As shown on Figure ??, the ViewModel remains in memory until the Lifecycle it's scoped to goes away permanently: in the case of an activity, when it finishes, while in the case of a Fragment when it's detached. The fact that ViewModel can survive configuration make ViewModel a perfect view state holder.

#### 3.1.6 LiveData

LiveData is another useful tool from the AAC. LiveData is an observable, life-cycle aware data holder class, that goes hand-in-hand with ViewModel. Fragment or Activity can subscribe for updates and LiveData will notify its subscribers when new data is available, allowing them to update themselves accordingly. Because LiveData is lifecycle-aware it allows avoiding memory leaks and can easily be used to restore view state after the configuration change.

Figure 3.2: ViewModel lifecycle



## 3.2 Architecture Implementation

There are four three important parts to the architecture implementation: ViewModels, Interactors, Repositories, and Fragments and Activities.

### 3.2.1 Repository

Repository acts as a single source of truth for the Interactors. Almost all of them are dependent on the remote data source - API class provided by Retrofit, with the only difference being Notebook repository, which also has a Room dependency.

Listing 3.4 demonstrates implementation of the `getNotebooks()` function, which is marked as suspend. This allows using all the power of the Kotlin

### 3. IMPLEMENTATION

---

Coroutines. Retrofit got a Kotlin Coroutines support, allowing use of generic `Deferred` class as a return type, that is then used to wait for the result.

Listing 3.4: Repository Implementation Example

```
class NotebookRepositoryImpl(
    private val localDataSource: AppDatabase,
    private val remoteDataSource: Api
) : NotebookRepository {

    /**
     * retrurns a list of the notebooks received from the api
     * and save the result into database
     */
    override suspend fun getNotebooks(): List<Notebook> {
        // wait for the api request to finish
        val list = remoteDataSource.getNotebooksAsync().await()
        localDataSource.notebookDao()
            .putAll(list.map { it.toDomain() })
        return list
    }
    // other functions
}
```

#### 3.2.2 Interactors

Interactors are used to perform one single use-case, or in other words, one single business logic operation. There are a lot of ways an interactor can be implemented. Sometimes it makes sense to make the interactor return `LiveData`, some times it is easier to handle the operation result via callbacks. Interactors in StudyPad relies on callbacks.

Every Interactor in StudyPad extends `BaseInteractor` or `BaseInputInteractor` (when the operation requires input). `BaseInteractor`, is an abstract class, it shortened implementation can be seen on Listing 3.5. This implementation fully relies on Kotlin Coroutines to execute the request in the background and provide a result using callback. This implementation of Interactors makes them very easy to declare, the only thing that has to be specified the actual action to be performed. The example of the resulting interactor can be seen on Listing 3.6



Listing 3.5: BaseInteractor Implementaion

```
abstract class BaseInteractor<T> {  
  
    private var parentJob: Job = Job()  
    var ioContext = Dispatchers.IO  
    var mainContext = Dispatchers.Main  
  
    protected abstract suspend fun runInBackground(): T  
  
    fun execute(block: CompletionBlock<T>) {  
        val response = Request<T>().apply { block() }  
        unsubscribe()  
        parentJob = Job()  
        CoroutineScope(mainContext + parentJob).launch {  
            try {  
                val result = withContext(ioContext) {  
                    runInBackground()  
                } catch (e: Exception) {  
                    handleException(e)  
                }  
            }  
        }  
    }  
}
```

Listing 3.6: LoginInteractor Example

```
class LoginInteractor(private val repo: KeychainRepo) :  
    BaseInputInteractor<LoginInteractor.Input, Unit>() {  
  
    data class Input(val email: String, val password: String)  
  
    override suspend fun runInBackground(input: Input) {  
        val email = input.email  
        val pwd = input.password  
        return keychainRepository.login(email, pwd)  
    }  
}
```

### 3.2.3 Applying ViewModel and LiveData

`ViewModel` in `StudyPad` manages three things:

- Navigation
- Interactors execution
- Store current view's state

To make development the development process a bit easier, I have created an abstract class `BaseViewModel` class for all the others `ViewModel`s to extend. `BaseViewModel` contains two `MutableLiveData` instances. First one is `baseViewStateLiveData`. It holds a view state, that is the most common for every view: loading progress and error event. The second and the last one is `navigationEventLiveData`, that handles navigation events and commands. This allows to easily change view state using helper functions provided by `BaseViewModel` (Listing 3.7) in its subclasses.

Listing 3.7: LoginInteractor Example

```
protected fun toggleLoading(loading: Boolean) {
    val oldState = viewStateLiveData.value
    val updatedState = oldState.copy(isLoading = loading)
    viewStateLiveData.postValue(updatedState)
}

fun navigateBack() {
    val backEvent = NavigationEvent.NavigateBack
    navigationEventLiveData.call(backEvent)
}

protected fun navigateTo(direction: NavDirections) {
    val navEvent = NavigationEvent.NavigateTo(direction)
    navigationEventLiveData.call(navEvent)
}
```

It is important to mention that the way `LiveData` manages its data is not perfect for some events. There are some events that should be triggered only once, like showing an error, confirmation message or navigating user to another screen. For this reason, I have implemented a `Event` class (Listing 3.8). `Event` is very simple generic class that holds a data and only allows to handle this data once. Event handling is done using `handle()` function, that accepts the lambda function, that will be invoked only once. All the error events and navigation events are using this class.

Listing 3.8: Event class

```
class Event<out T>(private val content: T) {  
  
    private var isAlreadyHandled: Boolean = false  
    fun handle(block: (T) -> Unit) {  
        if (!isAlreadyHandled) {  
            isAlreadyHandled = true  
            block.invoke(content)  
        }  
    }  
}
```

Interactor execution usually results in updating the current screens View State, an example of this process can be seen on Listing 3.9. This is just one of the many examples that illustrates the flow of the data in the application.

Listing 3.9: Usage of LiveData and Interactors in ViewModel

```
// Update base view state to show loading  
toggleLoading(true)  
// Execute the interactor  
getRelevantNotebooks.execute {  
    // Handle success  
    onComplete {  
        toggleLoading(false)  
        val data = it.map { it.toDomain() }  
        // post new valuee  
        dataSource.postValue(data)  
    }  
    onError {  
        // handle error  
        toggleLoading(false)  
        handleApplicationError(it)  
    }  
}
```

### 3.2.4 Present the data in Fragments

The Activities in StudyPad serves only as a container for Fragments, that's why the actual screens and flows are represented by Fragments. Just like with the `BaseViewModel`, I have created a basic definition of `Fragment`, that others fragments can extend – `BaseFragment`

### 3. IMPLEMENTATION

---

`BaseFragment` provides a couple of helper methods that are used across the application and automatically subscribes for the two `LiveData` instances available from the `BaseViewModel`.

The `baseViewStateLiveData` subscription allows to define `showLoading()` and `showError()` methods that others Fragments can override and manage this state without subscribing to it manually all the time.

The `navigationLiveData` subscription allows to handle the navigation in one place. The `LiveData` itself it implemented using `sealed class`. In Kotlin, `sealed classes` are used for representing restricted class hierarchies, which in a sense allows to use these classes as much more powerful enums.

The way the `NavigationEvent` is implemented is shown on Listing 3.10. This event is then handled in the `BaseFragment` and depending on the type of the event, the actual navigation is performed (Listing 3.11)

Listing 3.10: `NavigationEvent`

```
sealed class NavEvent {  
    data class NavigateTo(val direction: NavDir): NavEvent()  
    data class ChangeFlow(val flow: Flow) : NavEvent()  
    object NavigateBack: NavEvent()  
}
```

Listing 3.11: `NavigationEvent`

```
private fun handleNavigationEvent(event: NavEvent) {  
    val navController = view?.findNavController()  
    when (event) {  
        is NavEvent.NavigateTo -> navController?.navigate(event)  
        is NavEvent.NavigateBack -> navController?.popBackStack()  
    }  
}
```

## 3.3 Component diagram

## 3.4 Installation

# Testing



---

# Conclusion

The goal of this thesis was to develop a StudyPad client for Android OS, which involved domain, requirements and existing solutions analysis as the first step. Next, the application UI and architecture was designed to support the requirements established in the previous step. Finally, the application was developed applying best practices from the world of the modern Android application development. All requirements specified in the Analysis were implemented along with the couple new functionalities that were not a part of the requirements specifications:

- A screen to view latest notifications received.
- Introduction of real-time features when receiving. push-notifications when the application is in the foreground.
- The functionality that allows users to send feedback about the application.
- Offline functionality that covers Challenges and Local Notebooks.

The application has not been uploaded to the Play Store yet; hence it has not been tested by the broader set of users. Testing the application will be the last step to this thesis and it will give me more insight into what was done right and can be improved in the future.

## Future development

StudyPad development is far from over. The current iteration of the application had an emphasis on content creation, sharing, and collaboration, with just a few tests available for users to study and learn. The real process of studying has yet to be researched and executed. This could be a main focus for the future development of StudyPad.

There also some small changes that could be addressed such as:

## CONCLUSION

---

- Powerful Notes editor that would make notes creation easier.
- Implementing of Paging. Currently, neither API or the Application supports paging. Paging would improve loading time, which is especially important when there is a lot of content available.
- Unit and Instrumentation tests to ensure application stability.
- Implementation of Modular Architecture. It would enhance the quality of code and make test coverage easier.

Though there are still a lot of work to be done on the Android application, StudyPad could be much bigger. In the long run, the iOS client could be finalised, and the Web client implemented making it a fully cross-platform experience.



---

## Bibliography

- [1] Google. *Android Platform Architecture* [online]. Accessed on 2019-07-07. Available from: <https://developer.android.com/guide/platform>
- [2] Google. *Android SDK Versions* [online]. Accessed on 2019-07-07. Available from: <https://developer.android.com/training/basics/supporting-devices/platform>
- [3] Google. *AndroidX Overview* [online]. Accessed on 2019-07-07. Available from: <https://developer.android.com/jetpack/androidx>
- [4] Google. *Android Activity* [online]. Accessed on 2019-07-07. Available from: <https://developer.android.com/guide/components/activities/intro-activities>
- [5] Google. *Android Fragments* [online]. Accessed on 2019-07-07. Available from: <https://developer.android.com/guide/components/fragments>
- [6] Google. *Android broadcast receiver* [online]. Accessed on 2019-07-07. Available from: <https://developer.android.com/reference/android/content/BroadcastReceiver>
- [7] Google. *Android Services Overview* [online]. Accessed on 2019-07-07. Available from: <https://developer.android.com/guide/components/services>
- [8] Google. *Android Intents* [online]. Accessed on 2019-07-07. Available from: <https://developer.android.com/reference/android/content/Intent>
- [9] Google. *Android Resources Overview* [online]. Accessed on 2019-07-07. Available from: <https://developer.android.com/guide/topics/resources/providing-resources>

## BIBLIOGRAPHY

---

- [10] Google. *Android View Documentation [online]*. Accessed on 2019-07-07. Available from: <https://developer.android.com/reference/android/view/View>
- [11] Google. *Android Platform [online]*. Accessed on 2019-07-07. Available from: <https://developer.android.com/reference/android/view/ViewGroup>
- [12] Google. *Android Platform [online]*. Accessed on 2019-07-07. Available from: <https://developer.android.com/guide/topics/ui/layout/linear>
- [13] Google. *Android Constraint Layout [online]*. Accessed on 2019-07-07. Available from: <https://developer.android.com/reference/android/support/constraint/ConstraintLayout>
- [14] Google. *Android Build Configuration [online]*. Accessed on 2019-07-07. Available from: <https://developer.android.com/studio/build>
- [15] JetBrains. *Kotlin 1.0 Released [online]*. Accessed on 2019-7-14. Available from: <https://blog.jetbrains.com/kotlin/2016/02/kotlin-1-0-released-pragmatic-language-for-jvm-and-android/>
- [16] JetBrains. *Official Kotlin support for Android[online]*. Accessed on 2019-7-14. Available from: <https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/>
- [17] JetBrains. *Kotin Extensions [online]*. Accessed on 2019-7-14. Available from: <https://kotlinlang.org/docs/reference/extensions.html>
- [18] JetBrains. *Kotlin Android Extensions [online]*. Accessed on 2019-7-14. Available from: <https://kotlinlang.org/docs/tutorials/android-plugin.html>
- [19] Muntenescu, F. *Android Architecture Patterns Part 2:Model-View-Presenter*. 2016, [Online; accessed 23-March-2019]. Available from: <https://upday.github.io/blog/model-view-presenter/>
- [20] Muntenescu, F. *Android Architecture Patterns Part 3:Model-View-ViewModel*. 2016, accessed on 2019-07-07. Available from: <https://upday.github.io/blog/model-view-presenter/>
- [21] Google. *Android Architecture Components [online]*. Accessed on 2019-07-07. Available from: <https://developer.android.com/topic/libraries/architecture/>
- [22] Google. *Firebase*. Available from: <https://firebase.google.com/products>

- [23] Google. *Firebase Analytics Overview [online]*. Accessed on 2019-7-14. Available from: <https://firebase.google.com/products/analytics/>
- [24] Google. *Firebase Messaging Overview [online]*. Accessed on 2019-7-14. Available from: <https://firebase.google.com/products/cloud-messaging/>
- [25] Google. *Firebase Auth Overview [online]*. Accessed on 2019-7-14. Available from: <https://firebase.google.com/products/auth/>
- [26] Google. *Firebase Crashalytics Overview [online]*. Accessed on 2019-7-14. Available from: <https://firebase.google.com/products/crashlytics/>
- [27] Roman, L. *StudyPad Admin Panel*. Available from: <https://github.com/levinzonr/studypad-web>
- [28] Roman, L. *StudyPad iOS Client*. Available from: <https://github.com/levinzonr/studypad-ios>
- [29] Roman, L. *StudyPad Backend*. Available from: <https://github.com/levinzonr/studypad-spring>
- [30] SpringSource. *Introduction to Spring Framework [online]*. Accessed on 2019-07-07. Available from: <https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/overview.html>
- [31] Deleuze, S. *Introducing Kotlin support in Spring Framework 5.0 [online]*. 2017, accessed on 2019-07-07. Available from: <https://spring.io/blog/2017/01/04/introducing-kotlin-support-in-spring-framework-5-0>
- [32] Google. *Floating Action Button*. Accessed on 2019-07-07. Available from: <https://material.io/design/components/buttons-floating-action-button.html#>
- [33] Google. *Bottom App Bar*. Accessed on 2019-07-07. Available from: <https://material.io/design/components/app-bars-bottom.html#behavior>
- [34] Google. *Material Chips*. Accessed on 2019-07-07. Available from: <https://material.io/design/components/chips.html>
- [35] Google. *Understanding navigation*. Accessed on 2019-07-07. Available from: <https://material.io/design/components/tabs.html#usage>



## Acronyms

**GUI** Graphical user interface

**XML** Extensible markup language

**FAB** Floating Action Button

**UI** User Interface

**UX** User Experience

**FCM** Firebase Messaging Service

**GCM** Google Messaging Service

**DSL** Domain Specific Language

**DI** Dependency Injection

**IoC** Inversion of Control



## Contents of enclosed CD

	readme.txt .....	the file with CD contents description
	exe .....	the directory with executables
	src .....	the directory of source codes
	wbdcm .....	implementation sources
	thesis .....	the directory of $\text{\LaTeX}$ source codes of the thesis
	text .....	the thesis text directory
	thesis.pdf .....	the thesis text in PDF format
	thesis.ps .....	the thesis text in PS format





## Contents of enclosed CD

```

| readme.txt ..... the file with CD contents description
|_ exe ..... the directory with executables
|_ src ..... the directory of source codes
|   |_ wbdcm ..... implementation sources
|   |_ thesis ..... the directory of LATEX source codes of the thesis
|_ text ..... the thesis text directory
|   |_ thesis.pdf ..... the thesis text in PDF format
|   |_ thesis.ps ..... the thesis text in PS format

```