**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: Мобилно приложение за водене на бележки

Дипломант: Научен ръководител:

*Илиян Емилов Германов ст. преп . Любомир Чорбаджиев*

СОФИЯ

2016

**Увод**

В днешното забързано ежедневие сме принудени да помним много и различни работи като например списъка с покупки, задачите, които трябва да свършим и така нататък. Доста често се случва да забравим за нещо важно, за което ще съжаляваме по-късно. Лесен начин за справяне с подобни проблеми е записването на бележки на хартиен носител, но при нарастването на обема на нашите записки се получава „каша“ и ефективността на този подход рязко намалява. Дори и да успеем да се справим с организацията на хартиените бележки все още можем да се сетим за нещо прекалено късно или даже да го пропуснем. Разчитането на близък приятел или роднина да ни подсети не е много надеждно решение. Остава ни единствено константно да преглеждаме нашето тефтерче, което би било доста досадно и трудоемко. Вторият по-умен и ефективен начин е да прибегнем до помощта на техниката или по-точно смартфоните.

В XXI век почти всеки разполага със смартфон (мобилен телефон с разширена функционалност, сравнима с джобните компютри). Използвайки мобилно устройство получаваме хиляди възможности: можем да съхраним неограничен брой бележки без да настъпва хаос в организацията, да редактираме записаното без да се налага да драскаме или преписваме, да получим напомняне в точното време, да прикрепяме снимков материал и много други. Появява се нов проблем не всички производители разполагат с фабрично приложение за водене на бележки. Вариантът да използваме чернови на съобщения за записване и аларми за известяване е доста непрактичен и нелеп при наличието на специализирани приложения.

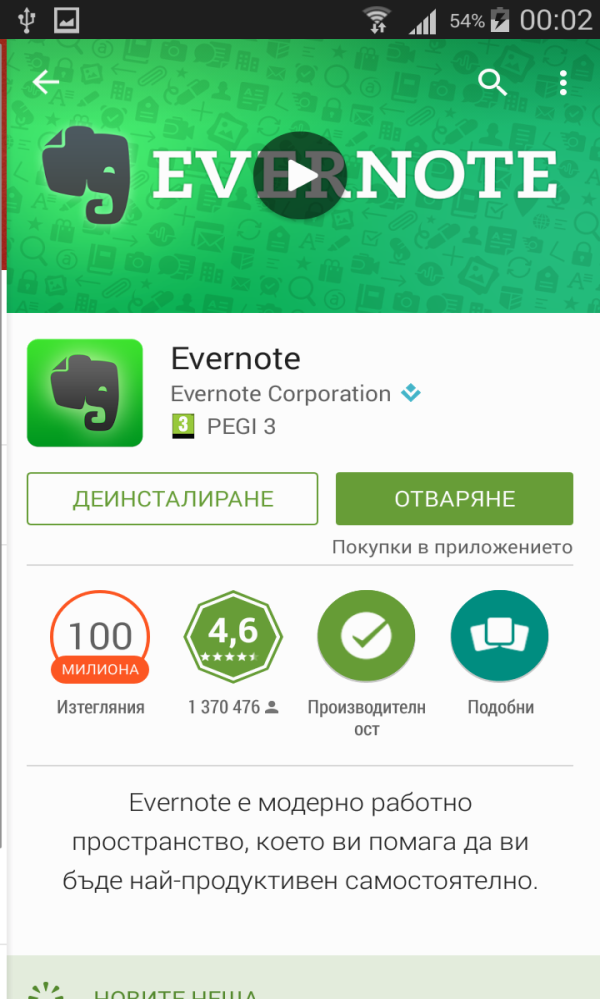
Целта на настоящата дипломна работа е да се създаде мобилно приложение за лесно, бързо и ефективно водене на бележки, позволяващо на потребителя да записва своите мисли, да получава напомняния по-късно, да прикрепя снимков материал и да организира своите бележки. Някои от най-важните функционалности на приложението са: създаване на текстова бележка, с възможност за добавяне на изображение (заснето с в момента с камерата или избрана от галерията); създаване на списъци; създаване на гласова бележка, която се транскрибира автоматично; раздел за важни бележки; раздел за частни бележки, които се криптират и се достъпват с парола; кошче; поставяне на подсещане. За целта ще бъде направено проучване, анализ и сравнение на съществуващи подобни продукти.

**I Глава**

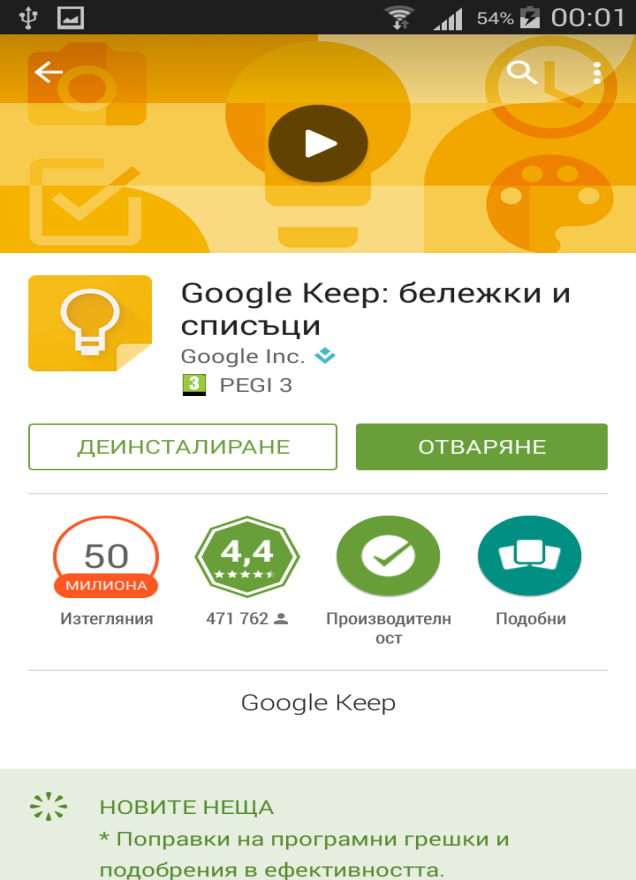
**Проучвателна част. Преглед на съществуващи продукти, развойни среди и развойни средства.**

* 1. **Преглед на съществуващи продукти**

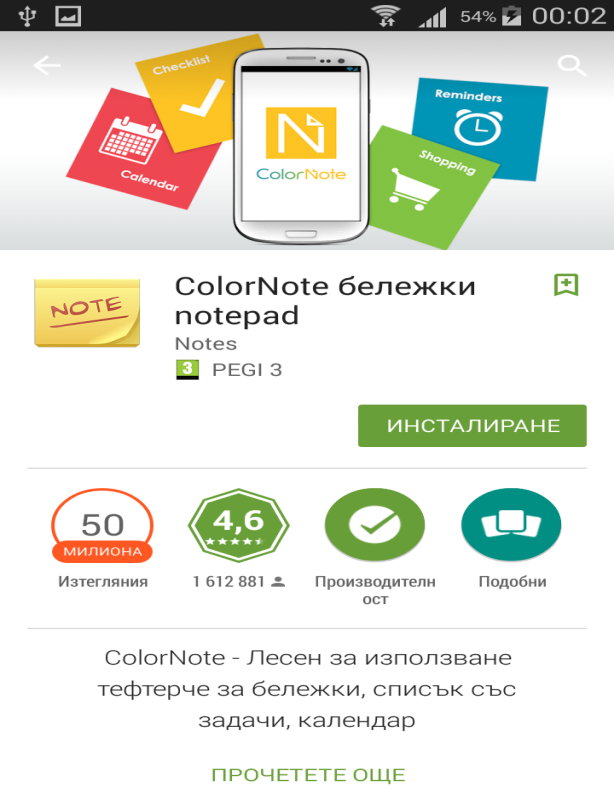
Едни от типовете приложения, които достигат най-голям брой сваляния в Google Play store са “note taking applications” или на български език приложения за водене на бележки. Примери за това са водещите приложения от тази категория като Evernote (100 000 000 – 500 000 000 инсталирания), Google Keep (50 000 000 – 100 000 000 инсталирания), ColorNote (50 000 000 – 100 000 000 инсталирания) и много други.



“Evernote в Google Play”



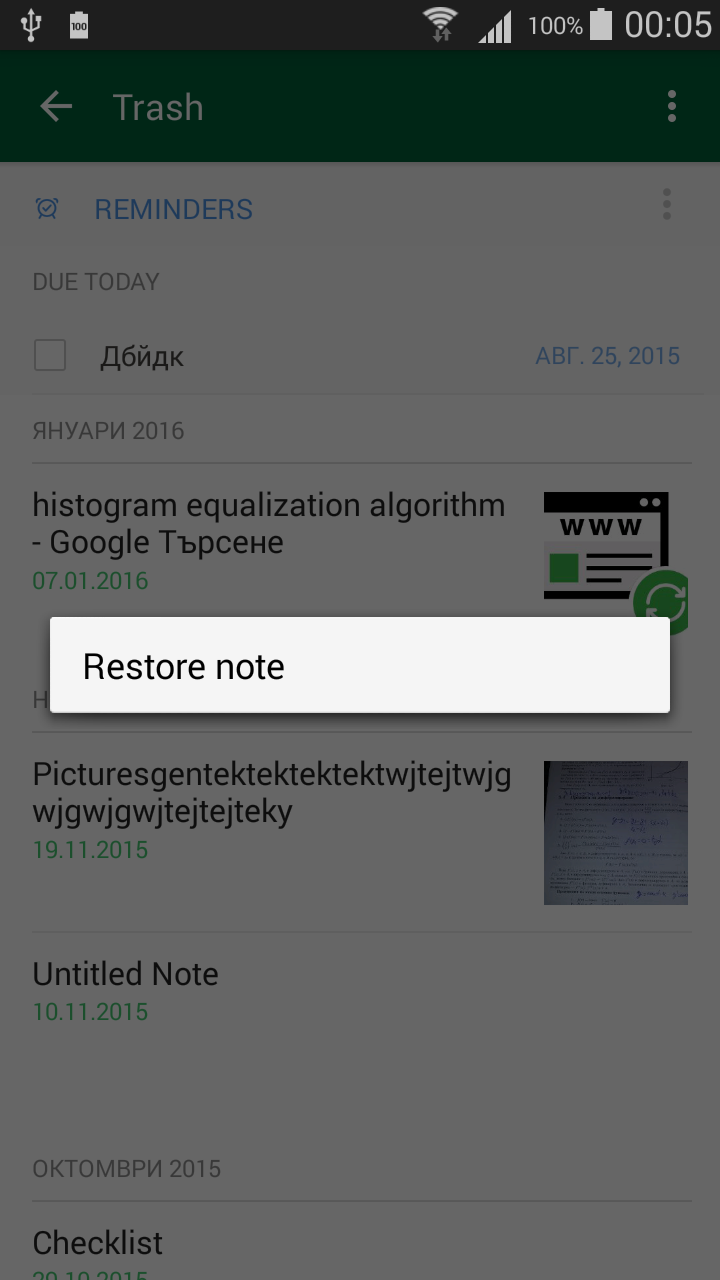
“Google Keep в Google Play”



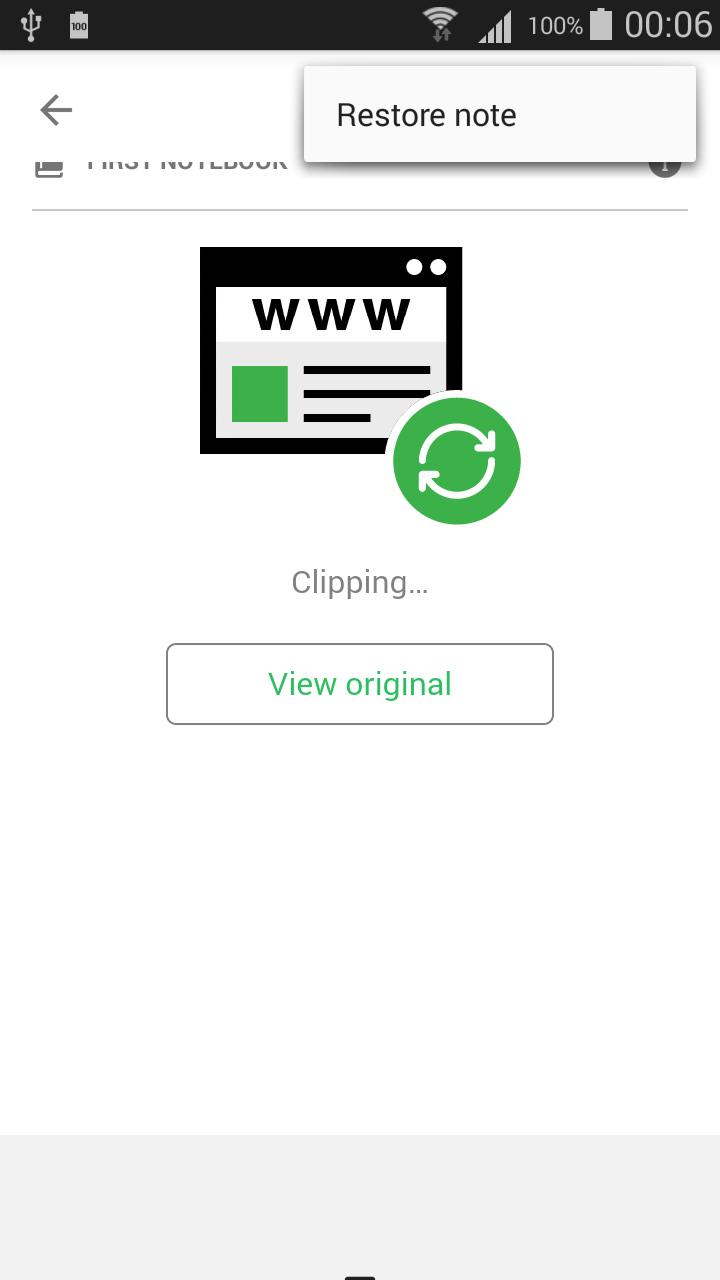
“ColorNote в Google Play”

Освен, че приложенията за водене на бележки достигат до много и различни по пол и възраст потребители, те съдържат на мобилните им устройства сравнително по-дълго от другите видове апликации. Например една игра от категорията “entertainment” (забавление) има голяма вероятност да бъде деинсталирана в момента, в който омръзне на потребителя, докато едно качествено приложение за водене на бележки би му вършило работа дълго време. В тази сфера изборът на добри на приложения е голям, но смятам, че може да бъде проектирано ново, което е да бъде по-лесно за използване, бързо и ефикасно. Идеята за настоящата ми дипломна работа се роди лятото, когато се нуждаех от приложение за удобно записване на просто бележки. Първо си инсталирах Evernote с идеята, че като е с най-много сваляния ще ми свърши най-добра работа. Още в самото начало не ми хареса това, че за да го ползваш си принуден да си направиш регистрация (ползваща се за синхронизирането между различни устройства, което е полезна функционалност), за да запиша просто един час с задача, който ще изтрия веднага след като премине. Друго, което ми направи негативно впечатление е сложността на приложението и хилядите функционалности, които няма да използвам като например Work Chat и т.н. Също така честите известия подканващи ме да си закупя “premium upgrade” на акаунта ме накараха да го деинсталирам. След това опитах с Google Keep. При него нямах проблем с входа към приложението, тъй като имам Google акаунт. Създаването на различни видове записки беше лесно, но тяхното управление и организиране беше трудно. Например, за да изпратиш бележка в кошчето имаш няколко варианта. При първият е необходимо да я отвориш и да намериш опцията за изтриване от менюто, което става сравнително бавно, тъй като отварянето зарежда нов екран. Вторият е да я изхвърлиш с “swipe” директно от главния екран, но вместо да отиде в коша се премества в архива, от който ми отне някъде около 10 минути да разбера как се трие.

Със същият проблем се сблъсках и в Evernote само където до момента, в който пиша тази документация така и не разбрах как се изтрива бележка завинаги и дали това въобще е възможно. Прилагам няколко екранни снимки от Evernote:



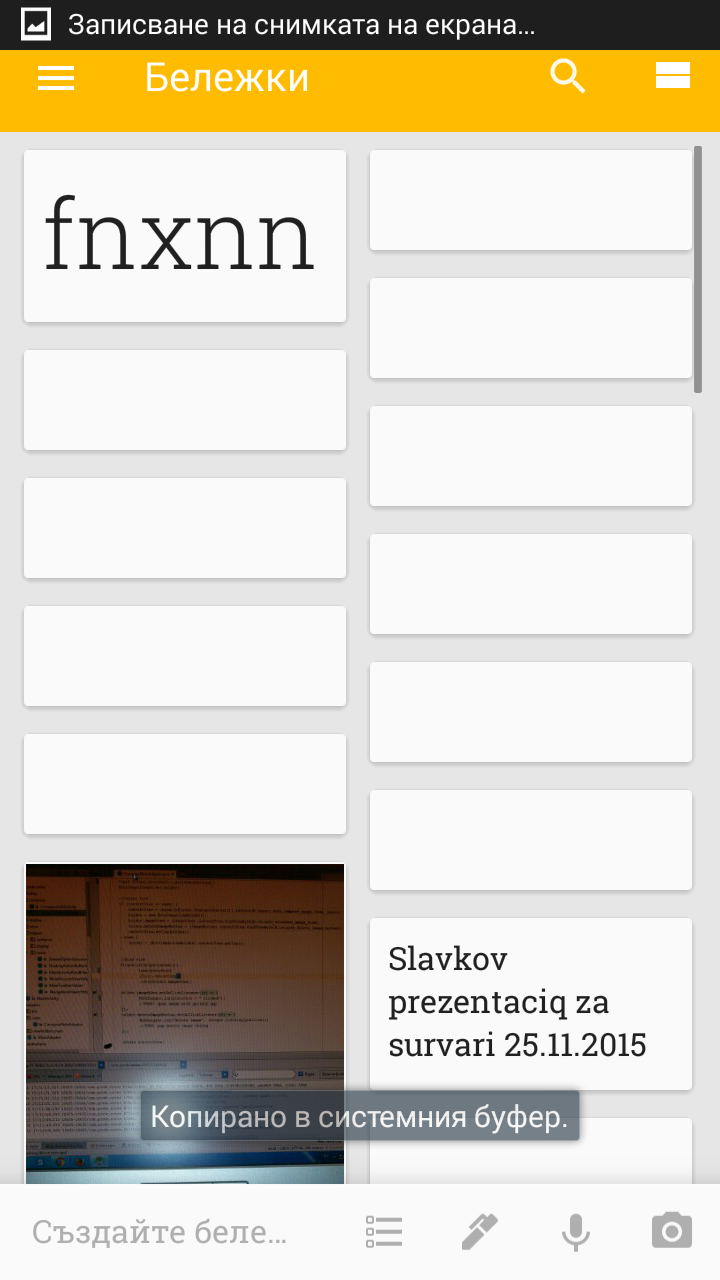
“Long click върху item в Trash”



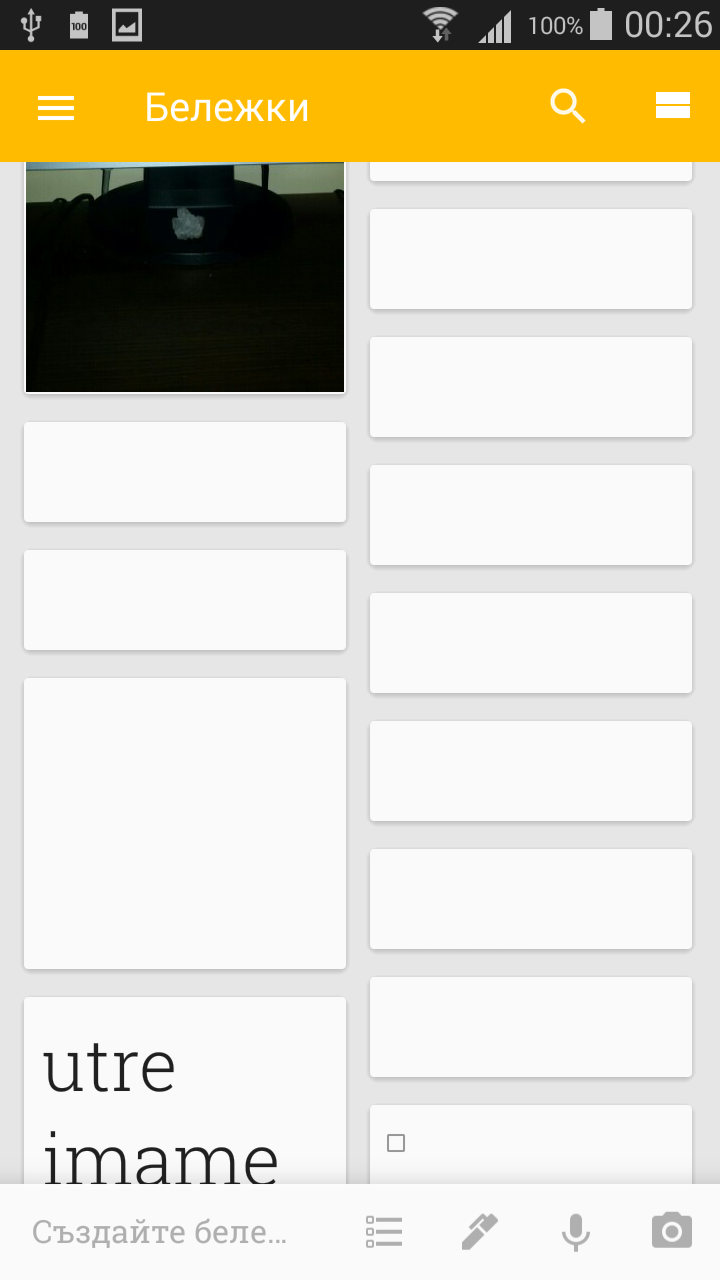
“Отворено меню от item в Trash”

Както ясно се вижда от екранните снимки опцията “Delete” просто липсва на всички места, за които се сетих.

Друг проблем, на който се натъкнах в Google Keep е, че при излизане от редактиране или създаване на нова бележка промените винаги се запазват. Заради тази функционалност немалко пъти запазих промени, които не исках да бъдат съхранени. Това лесно може да бъде забелязано на следните екранни снимки:



“Празни бележки в Google Keep”



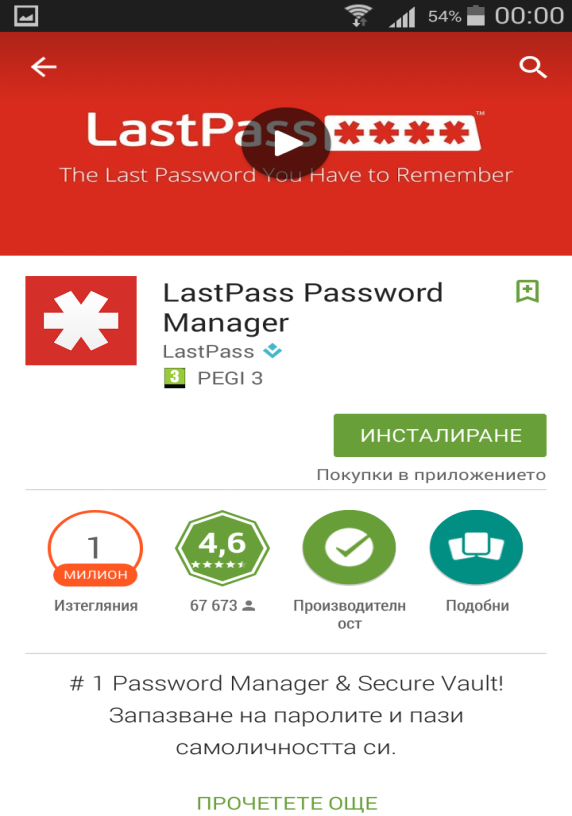
„Празни бележки в Google Keep (2)”

По време на проучването на Google Keep забелязах, че при натрупването на повече бележки и използването на приложението от по-слаб телефон (тествано върху Samsung Galaxy S3 Neo) се появява известно забавяне, което разваля потребителското усещане. Нещо, което ми направи добро впечатление в Google Keep, но отсъстваше в Evernote е това, че има възможност да се разместват бележките с “drag and drop”.

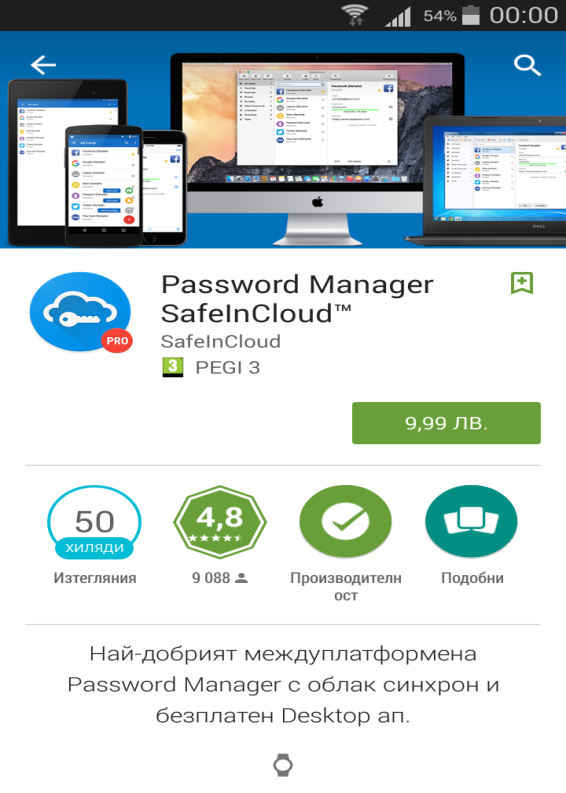
Друга важна за мен функционалност, която не открих в гореспоменатите водещи приложения е създаването на “private” (частни) бележки, които да не могат да се видят от всеки, който има достъп до мобилното устройство. Частните бележките биха могли да се използват за много цели и биха направили приложението много по-полезно за потребителя. Например с тяхна помощ може да съхранявате лична информация и пароли. Освен това приложенията проектирани специално за съхранение на пароли са доста сваляни в Google Play. Примери за това са Keeper® Password Manager (10 000 000 – 50 000 000), LastPass Password manager (1 000 000 – 5 000 000) и платеното Password Manager SafeInCloud™ с цена 9,99лв. в България (50 000 – 100 000).



“Keeper® Password Manager в Google Play”



“LastPass Password Manager в Google Play”



“Password Manager SafeInCloud™ в Google Play

**1.2 Развойни среди**

**1.2.1 ADT**

ADT (Android Development Tools) е развойна среда под формата на плъгин за Eclipse. Тя се предлага безплатно и съдържа набор от полезни за разработването инструменти, към които и визуализация, и улеснена работа с UI (User Interface), с цел по-добро разработване и по-добре оформен дизайн.

**Някои от предлаганите инструменти са :**

* автоматично дописване на код
* анализатор за XML
* анализатор на JAVA
* поддръжка на всички основни Java библиотеки
* debugger
* Dalvik Virtual Machine – виртуална машина за Android
* компилатор
* добавка за Version Control Systems (Git).

**1.2.2 “Android studio”**

„Android studio“ е официалното IDE предлагано от Google.

То има поддръжка на най-новият Android 6.0 (Marshmallow) като предлага и емулатор за него. Подобно на “Elcipse” има подходящ графичен редактор, като повечето от функционалностите му се припокриват с ADT плъгина.

**„Android studio“ предлага :**

* редактор
* примерни кодове на прости програми
* интеграция на Version Control System (Git)
* създаване на виртуални устройства с всякакви размери и форми (телевизори, телефони, таблети и др.)
* възможност за създаване на различни .apk файлове с различни добавки и версии на базата на един и същ проект

**1.2.3 SDK**

SDK (Software Development Kit) е добавка както за Eclipse така и за Android studio. Тя предоставя услугата Android manager, чрез която удобно могат да се свалят инструменти и библиотеки за разработването на различните версии на Android операционната система, както и различни конфигурации спомагащи за по-добра симулация на виртуалната машина.

**1.3 Развойни средства**

**1.3.1 Основни компоненти на Android приложение**

Основните компоненти на едно Android приложение са :

* „Activity“ - презентационен слой на приложението (екрани)
* „Content Provider“ - предоставя интерфейс към данните на приложението
* „Broadcast Receiver“ - компонент който получава и отговаря на системни съобщения и „Intents“
* „Services“ - услуги, които изпълняват задачи на зададен фон без да предоставят потребителски интерфейс.
* „Views“ (изгледи) - представляват различни части от потребителския интерфейс като бутони, текстови полета и други. Базовия клас на всички изгледи е „android.view.View“.
* „Intent“ - това са асинхронни съобщения, които изискват някаква функционалност от други компоненти като „Activities“.

**II глава**

**Изисквания към програмния продукт. Избор на езика за програмиране и софтуерните средства. Описание на структурата на програмата и базата данни**

**2.1. Изисквания към програмния продукт**

Създаване на мобилно приложение за Android платформата, което да служи за бързо, лесно и ефективно водене на бележки. Приложението трябва да може поддържа създаването и организирането на различни видове бележки – текстови, текстови с прикрепен снимков материал, списъци и автоматично транскрибирани гласови бележки. Да има раздели спомагащи за по-добра организация– „All notes” в него се съхраняват всички бележки, „Starred” – тук се намират всички важни за потребителя бележки, които той е отбелязал със звездичка, “Private” – раздел, който се достъпва само с парола и бележките в него са криптирани и “Bin” – кошче, в което попадат всички изтрити бележки, които не са частни (от “Private” раздела). Да има функционалност за напомняне в зададено време и да има търсене за различните раздели. Потребителския интерфейс да е опростен и разбираем за потребителя, и да позволява приятна работа с приложението.

**2.2. Избор на език за програмиране и софтуерните средства**

За създаването на приложението са използвани мобилната операционна система “Android”, езикът “Java” и средата “Android Studio”. Това е така поради три причини. Първо “Java” е официалният език под който се разработват Android приложения, а “Android Studio” е средата, която Google препоръчва. Второ, вече от година и половина разработвам Android приложения използвайки “Java” и “Android Studio”. Натрупаният опит и познания би ми помогнал да се съсредоточа изцяло върху проектирането на приложението и така да създам един завършен продукт готов да се конкурира с водещите от този тип. И трето както в момента, така и за в бъдеще искам да продължа да се занимавам с разработката на native Android приложени, тъй като тази област е една от най-бързо развиващите се такива в технологичния свят и предлага неограничен брой възможности както на потребителите, така и на програмистите.

**2.3. Структура на приложението**

**2.3.1. Зависимости (dependencies) на проекта**

В приложението са използвани 14 външни библиотеки за по-разбираемо описани нека ги разделим на два вида – “com.android.support” (официални зависимости, невлизащи в SDK-а) и “3rd party” (както от името става ясно библиотеки принадлежащи на трети лица, пуснати с лиценз за свободни използване).

**“com.android” зависимости:**

* 'com.android.support:appcompat-v7:23.1.1' – предлага класове съвместими и поддържащи по-стари версии на Android
* 'com.android.support:recyclerview-v7:23.1.1' – предлага оптимизиран контейнер подобен на класът ListView от Android SDK-а, но с тази разлика, че в него е имплементиран “View holder pattern”, което води до по-добро представяне при голям обем от данни
* 'com.android.support:cardview-v7:23.1.1' – съдържа класът CardView, който представлява повърхност с “Material Design”
* 'com.android.support:design:23.1.1' – помощна библиотека за дизайн, съдържащи основните практики в Android

* 'com.google.code.gson:gson:2.4' – не влиза изцяло в този тип, но пак едно от официалните библиотеки, тъй като неин създател е Google; предоставя възможност за превръщане на обекти в “JSON” формат

**“3rd party” зависимости:**

* 'com.github.clans:fab:1.6.2' – предоставя “Floating Action Button” и интерактивно меню към него, отговарящи на “Material Design”
* 'com.jakewharton:butterknife:7.0.1' – улеснява инициализирането на изгледи чрез идентификационен номер използван в xml файловете като предлага анотации, с които това да става с доста по-малко и разбираем код. Единственото изискване е да извика статичния метод на класа “ButterKnife” – ButterKnife#bind(activity);
* 'com.wdullaer:materialdatetimepicker:2.1.1' – предлага диалогови прозорци за избиране на дата или час, отговарящи на “Material Design”
* 'com.github.frankiesardo:linearlistview:1.0.1@aar' – съдържа контейнер, работещ вграден в друг “scrollable” контейнер
* 'com.squareup.picasso:picasso:2.5.2' – улеснява и оптимизира зареждането и представянето на изображения
* 'com.scottyab:aescrypt:0.0.1' – предоставя клас с възможност за криптиране и декриптиране на низови с “AES 256 encryption”
* ':libs:draglinearlayout-1.10' - локално добавена библиотека в проекта, предлага линейно оформление с възможност за влачене на елементите в него
* 'com.github.afollestad.material-dialogs:core:0.8.5.3@aar' - предоставя често използвани диалогови прозорци, отговарящи на “Material Design”
* 'com.github.afollestad.material-dialogs:commons:0.8.5.3@aar' – добавка към 'com.github.afollestad.material-dialogs:core:0.8.5.3@aar'

**2.3.2. Пакети на при приложението**

Приложението е разделено в 15 главни пакети, всеки от който отговаря за отделен проблем.

* Пакетът “com.gcode.notes”:

Главен пакет на приложението, съдържащ всички останали пакети.

* Пакетът “com.gcode.notes.activities”:

В този пакет се съдържат всички Activity класове на проекта и техните помощни класове.

* Пакетът “com.gcode.notes.adapters”:

Пакет, съдържащ всички адаптери използвани за управлението на контейнерите в приложението.

* Пакетът “com.gcode.notes.controllers”:

В този пакет са разположени всичките Controller класове, отговарящи за промяната и държанието на потребителския интерфейс в различните раздели на приложението.

* Пакетът “com.gcode.notes.data”:

В този пакет се съдържат всички логически единици, които представляват различните видове бележки. Това са базовите “ContentBase”, “ContentDetails”, “MyLocation”; от помощният пакет “list”, използвани за представянето на списъци – “ListDataItem” и “ListData”; “NoteData”, реализиращ бележка

* Пакетът “com.gcode.notes.database”:

Отговаря за създаването на база данните. Осигурява връзката между нея и приложението и контролира всички операции свързани с базата данни. В него се разполага класът “DatabaseController”, от който могат да бъдат достъпени публично всички методи за работа с базата. Съдържа и помощният пакет “extras” чрез който е реализиран класът “DatabaseController”.

* Пакетът “com.gcode.notes.extras”:

Съдържа всички помощни класове използвани в повече от един пакети. В него се намират пакетите: “utils” (от utilities – комунални услуги, предлага класове решаващи често срещани проблеми в приложението), “values” (буквално преведено стойности, в него са разположени всички класове, отговарящи за константите в проекта) и “builders” (съдържа класове, които служат за инстанциранието на други класове).

* Пакетът “com.gcode.notes.fragments”:

В този пакет се съдържат фраментите използвани в приложението.

* Пакетът “com.gcode.notes.helper”:

Временен пакет, отговарящ за разместването на бележки чрез влачене в главния контейнер. След реализиране на всички функционалности ще бъде преструктуриран.

* Пакетът “com.gcode.notes.motions”:

Съдържа всички класове реализиращи анимациите и преходите в приложението.

* Пакетът “com.gcode.notes.notes”:

В този пакет се намира singleton класът “MyApplication” , предоставящ достъп до контекста на приложението и инстанция на класът “DatabaseController” от всеки един пакет в проекта.

* Пакетът “com.gcode.notes.receivers”:

В него се съдържат всички „receivers” (приемници) използвани в дипломната работа.

* Пакетът “com.gcode.notes.serialization”:

Отговаря за сериализацията на обекти. Може да превърне дадена инстанция на обект в символен низ (string) в “JSON” формат.

* Пакетът “com.gcode.notes.services”:

Съдържа всички услуги (services) , използвани в приложението.

* Пакетът “com.gcode.notes.tasks”:

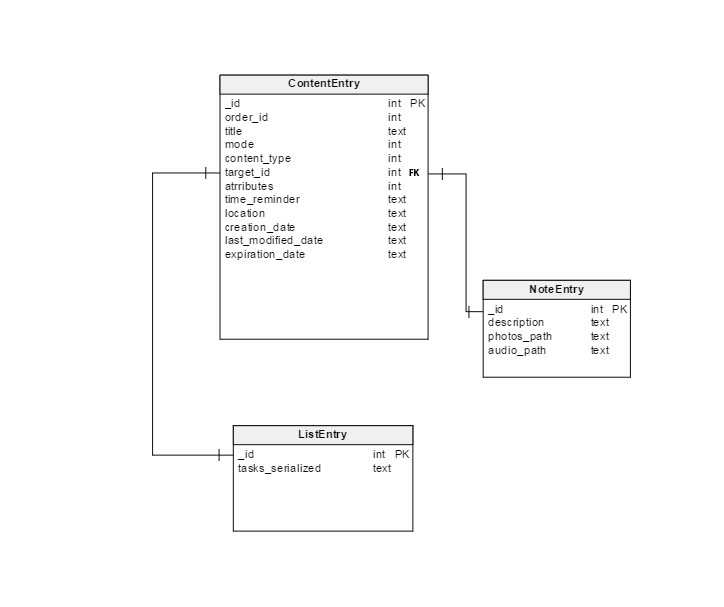
Съдържа всички задачи (tasks), използвани в приложението. Има 2 пакета – “async” (в него се намират всички класове, които са наследници на класът “AsyncTask”) и “other” (отговаря за всички, задачи които не са наследници на класът “AsyncTask”).

* Пакетът “com.gcode.notes.ui”:

В този пакет се разполагат класът “ActionExecutor” (отговарящ за всички действия свързани с потребителския интерфейс на приложението напр. изтриване на бележка и т.н.) и пакетът “helpers” (в него се намират всички помощни класове , реализиращи диалоговите прозорци и детайлите по потребителския интерфейс).

**2.4. Структура на базата данни**

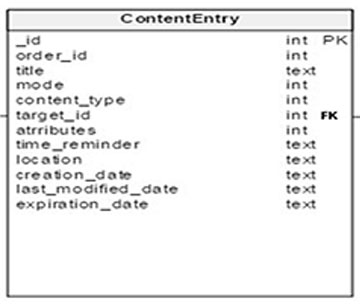
За база данни приложението използва “SQLite” със структура представена на „Фигура 2.4.1“:

****

Фигура 2.4.1

Базата данни се състои от следните 3 таблици – ContentEntry (съдържа общата информацията за бележка и списък), NoteEntry (съдържа атрибутите, които има една бележката), ListEntry (съдържа атрибутите, които има един списък).

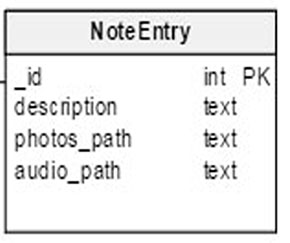
Таблицата „Content Entry” („Фигура 4.2.2):

****

**„**Фигура 4.2.2“

* \_id – първичен ключ на таблицата, уникален идентификационен номер на записа
* order\_id – идентификационен номер използван за определяне на мястото в подредбата на бележките
* title – заглавие на бележката
* mode – тип (режим) на бележката определящ към кой раздел принадлежи и как трябва да се третира от приложението ( normal, private, important, deleted\_normal, deleted\_important)
* content\_type – показва типа на записа, тоест дали е списък или бележка
* target\_id – вторичен ключ, служа за връзка с таблицата за атрибути на записа (NoteEntry / ListEntry)
* attributes – флаг показващ, дали даденият запис има атрибути и дали необходимо те да се извлекат
* time\_reminder – времето и датата, когато потребителя трябва да бъде подсетен, може да бъде NULL
* location – сериализирана инстанция на класа “MyLocation”, показваща местоположението, където е създадена бележката, може да бъде NULL
* creation\_date – датата на създаване на записа
* last\_modified\_date – датата на последна промяна на записа
* expiration\_date – датата на изтичане на записа (при попадане на запис в mode „deleted” той получава дата на изтичане след, която бива автоматично изтрит при следващото пускане на приложението), може да бъде NULL

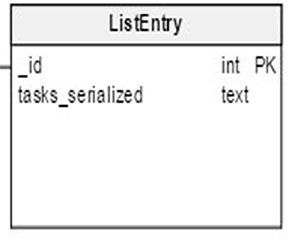
Таблицата “NoteEntry” („Фигура 4.2.3“):



„Фигура 4.2.3“

* \_id – първичен ключ на таблицата, уникален идентификационен номер на записа
* description – описание на бележката
* photos\_path – сериализиран лист от символни низове, представляващи път до прикрепеното изображение към бележката, може да бъде NULL
* audio\_path – път към прикрепеният звуков файл към бележката, може да бъде NULL

Таблицата “ListEntry” („Фигура 4.2.4“):



„Фигура 4.2.4“

* \_id – първичен ключ на таблицата, уникален идентификационен номер на записа
* tasks\_serialized – сериализирани лист от инстанции на класа “ListDataItem”, представляващи един предмет в списъка

За съхраняване на информация в приложението също са използвани “SharedPreferences” (физически представляващи един xml файл), които са подходящи за запазване на “key-value pairs”. В случая на текущата дипломна работа са в тях се запазва паролата на потребителя за частните бележки (криптирана с SHA1 алгоритъм), броят на грешните опити при въвеждане на паролата и флаг показващ дали потребителя е научил за навигационното меню.

**III глава**

**Описание на начина на реализация на приложението**

Нека разгледаме текущата дипломна работа, описвайки работата, която бива извършена във всеки един от екраните (Activity класовете), които са единадесет на брой – “MainActivity”, “ComposeNoteActivity”, “ComposeListActivity”, “DisplayNoteNormalActivity”, “DisplayNotePrivateActivity”, “DisplayNoteBinActivity”, “DisplayListNormalActivity”, “DisplayListPrivateActivity”, “DisplayListBinActivity”, “ExploreActivity”, “SettingsActivity”. Преди да започнем с това трябва да се спомене начинът, по който се достъпва контекста на приложението и базата данни от различните екрани. За тази цел служи класът „MyApplication” (споменат в 2.3.2 в пакетът „com.gcode.notes.notes”), който наследявайки Application, прави достъпването му възможно от всички Activity класове, които са част от това приложение. Класът “MyApplication” по същество представлява един “singleton” клас, като по този начин има една единствена инстанция и информация, която остава еднаква и непроменена за всички екрани.

**3.1. Екрани на приложението**

**3.1.1. MainActivity**

Това е главният екран, от който стартира и самото приложение (зададен в “Manifest” файла). Оформлението на този екран представлява “android.support.v4.widget.DrawerLayout”, който съдържа „android.support.design.widget.CoordinatorLayout“ и „android.support.design.widget.NavigationView“.

Нека започнем с изгледа “NavigationView”, която представлява изглед реализиращ често използвания за навигиране похват в Android, а именно “NavigationDrawer”. “NavigationDrawer” е повърхност с по-голяма издигнатост (elevation) спрямо другите повърхности, изграждащи оформлението на екрана. По подразбиране тя е скрито за потребителя и той има възможност да я види чрез „придърпване“ от някоя страна на екрана (в повечето случаи отдясно). В текущата дипломна работа изгледа “NavigationView” зарежда своите елементи от xml файла “menu\_drawer”, съдържащ навигационен бутон за всички раздели в приложението, включително допълнителните екрани за настройки (“Settings”) и опознаване на приложението (“Explore”). При първото стартиране на проекта навигационното меню се отваря автоматично, за да се потребителя да е наясно за неговото съществуване.

Да преминем към втората, по-голяма част изграждаща главния екран – визуалната структура “CoordinatorLayout”. Тя представлява “FrameLayout” (рамково оформление), позволяващо координирането на поведението и състоянието на своите “children” (деца) при често действия като превъртане на контейнер (“scrolling”), влачене на елементи (“drag”), “swipe” и други жестове. В нашия случай “CoordinatorLayout” съдържа : “AppbarLayout” (нужна, за да работи правилно координацията между децата), в която има лена с инструменти (“Toolbar”); контейнер, чрез който се визуализират бележките за избрания раздел (“RecyclerView”); текстов изглед, който бива показван, само когато контейнера за бележки е празен; меню за създаване на бележки / списъци, реализирано чрез класът от 3rd party библиотеката 'com.github.clans:fab:1.6.2' (описана в „точка 2.3.1, 3rd party зависимости“) - “com.github.clans.fab.FloatingActionMenu”. Менюто четири съдържа „mini fab” бутона, при избирането на които се методът “handlerItemClick(MainActivity, View)” от класът “FabMenuActionHandler”, който обработва събитието съобразно натиснатия изглед. Първият бутон отговаря за създаването на текстови бележки и стартира празно “ComposeNoteActivity”; вторият – за създаването на списъци и стартира празно “ComposeListActivity”; третият – стартира намерение (“intent”) за разпознаване на гласа - “RecognizerIntent.ACTION\_RECOGNIZE\_SPEECH”, което разпознава и транскрибира гласа потребителя, след което с помощта на класа “MainActivityResultHandler” зарежда “ComposeNoteActivity” с автоматично въведено описание отговарящо на казаното от потребителя. Езикът за разпознаване се извлича от настройките за език на устройство; третият – показва диалогов прозорец с опции за заснемане / избиране на снимков материал, който да бъде прикрепен към бележката. След успешно намиране на снимков материал се стартира с помощта на класа “MainActivityResultHandler” се стартира “ComposeNoteActivity” с автоматично добавен избраният / заснет материал. Лентата с инструменти съдържа бутон за отваряне на навигационното меню и изглед за търсене, в текущият раздел (реализиран чрез класа “MainSearchHandler”). Контейнерът чрез, който се визуализират бележките / списъците е изграден като решетка с 2 колони като отделните елементи представляват отделна повърхност –реализирана чрез “android.support.v7.widget.CardView” от 'com.android.support:cardview-v7:23.1.1' зависимостта (описана в 2.3.1). Във всички раздели освен кошчето има възможност за разместване на елементи чрез влачене, контролирано от класовете в пакетът “com.gcode.notes.helper” (описан в 2.3.2). В зависимост от избраният раздел с жеста “swipe” може да се изтриват / изпращат бележки в кошчето като се предлага възможност за „undo” (отменяне) с изскачането на „Snackbar” (представляващ повърхност показваща се временно най-отдолу на екрана). След натискането на елемент в контейнера се стартира съответното “DisplayActivity”, служещо за детайлно показване на елемента. Данните между екраните се предават като символен низ в JSON формат (сериализиран чрез класа “Serializer”, използващ „com.google.gson.Gson“ oт пакетът „com.google.gson“, описан в 2.3.2). При “scrolling” на контейнера бутона контролиращ менюто за създаване на бележки и лента инструменти се скриват / показват съобразно посоката на жестът, което улеснява и подобрява преглеждането на елементите. За зареждането и контролирането на различните раздели отговарят класовете от пакетът “com.gcode.notes.controllers” (описан в 2.3.2). За поддържането на правилното поведение и състояние на класа при смяната на ориентацията на екрана се грижи класът “MainActivityRotationHandler”.

**3.1.2. ComposeNoteActivity**

Този екран се стартира при създаването на нова бележка или при редактирането на вече съществуваща такава. Класът “ComposeNoteActivtiy” наследява класа “ComposeBaseActivity”, който съдържа общите функционалности между екрана за създаване на бележки и екрана за създаване на списъци. Главната визуална структура използвана за оформлението е “LinearLayout” (буквално преведено линейно оформление), която съдържа в себе си 2 деца – “AppBarLayout” и „ScrollView”. В “AppBarLayout” се разполага лента с инструменти, която дава възможност на потребителя да добавя изображение към бележката и бутон за запаметяване на направените промени. Контейнера „ScrollView” има едно дете – “LinearLayout” (линейно оформление) с вертикална ориентация, в което се разполага основната функционалност на екрана: поле за въвеждане на заглавие на бележка, бутон за даване на звезда, поле за въвеждане на описание, контейнер визуализиращ прикачените снимки („com.linearlistview.LinearListView“ от зависимостта 'com.github.frankiesardo:linearlistview:1.0.1@aar', описана в 2.3.1) ; линейно оформление за управление на прикаченото аудио (при бележките създадени чрез гласово разпознаване), контролирано от класа “AudioUtils”; фрагмент реализиращ функционалността за добавя на подсещане – „ComposeReminderFragment”. За задаването на първоначалното състояние на екрана служи класът “ComposeNoteStartStateHelper”, който в зависимост от намерението (“Intent”), от което е стартиран екрана, попълва автоматично полетата за заглавие и описание, добавя / премахва снимков или звуков материал, вдига съответните флагове показващи в какъв режим се намира екрана (създаване на нова бележка / редактиране на вече съществуваща такава; дали бележката е частна или не; дали режимът на бележката е променян – даване / отнемане на звездичка и т.н.). Когато се създава бележка за първи път и потребителят е включил услугата места (“places”) се взима текущото местоположение като използва най-точният снабдител снет от предпочитанията на потребителя (GPS Provider, Network Provider). Промените се запазват само когато потребителят изрично натисне бутона за запазване. За запазването отговаря класът „ComposeNoteSaveHelper“, който извлича цялата информация от оформлението на екрана (заглавие, описание, прикачени изображения, прикачен звуков материал, подсещане) и я прилага адекватно в базата данни, в зависимост от режима, в който се намира дадената бележка. Също така запазва датата на последна промяна на бележката и датата на създаване ако това е нова бележка. Ако бележката е частна преди да се запази в базата данни съдържанието и бива криптирано асинхронно с помощта на класа “EncryptNoteTask”, който наследява “AsyncTask”. При завършване на екрана се установява резултатен (“intent”), чрез който класът стартирал “ComposeNoteActivity” може правилно да установи своят интерфейс и поведение. За тази цел служи класът “ComposeNoteResultHandler”. За поддържането на правилното поведение и състояние на класа при смяната на ориентацията на екрана се грижи класът “ComposeNoteRotationHandler”.

**3.1.3. ComposeListActivity**

До този екран може да се достигне при създаването на нова бележка или при редактирането на вече съществуваща такава. Класът “ComposeListActivtiy” наследява класа “ComposeBaseActivity”, който съдържа общите функционалности между екрана за създаване на бележки и екрана за създаване на списъци. Главната визуална структура използвана за оформлението е “LinearLayout”, която съдържа в себе си 2 деца – “AppBarLayout” и „ScrollView”. В “AppBarLayout” се разполага лента с инструменти, която има бутон за запаметяване на направените промени. Контейнера „ScrollView” има едно дете – “LinearLayout” (линейно оформление) с вертикална ориентация, в което се разполага основната функционалност на екрана: поле за заглавие на списъка, бутон за даване на звезда, два контейнера за елементите на списъка (за маркирани и немаркирани елементи), позволяващи разместване на елементите чрез натискане върху бутон и влачене – представляващи инстанции от класа „com.jmedeisis.draglinearlayout.DragLinearLayout“; бутон за добавяне на нов елемент към списъка; фрагмент реализиращ функционалността за добавя на подсещане – „ComposeReminderFragment”. За задаване на стартовото състояние на екрана „ComposeListActivity” се грижи класът “ComposeListStartStateHelper”, който в зависимост от намерението (“Intent”), от което е стартиран екрана, попълва автоматично полето за заглавие на списъка, добавя всички елементи, които съдържа списъка; вдига съответните флагове показващи в какъв режим се намира екрана (създаване на нов списък / редактиране на вече съществуваща такъв; дали списъкът е частен или не; дали режимът на списъкът е променян – даване / отнемане на звездичка и т.н.). Когато се създава списък за първи път и потребителят е включил услугата места (“places”) се взима текущото местоположение като използва най-точният снабдител снет от предпочитанията на потребителя (GPS Provider, Network Provider). Всеки елемент от списъка е представен от класа “ListDataItem”, който има 2 полета – булева променлива дали даденият елемент е маркиран и символен низ, държащ съдържанието на елемента. Оформлението в този екран дава възможност са премахване на елемент от списъка чрез натискане на бутон намиращ се най-вляво от всеки елемент. След изтриването на елемент временно се появява “snackbar”, чрез който изтрият артикул може да бъде възстановен на предишната си позиция. Заради вложената в “ScrollView” позиция на контейнерите, визуализиращи елементите на списъка са използвани специално проектирани за целта адаптери – “ListComposeContainerAdapter” и “TickedListComposeContainerAdapter”. Промените се запазват само когато потребителят изрично натисне бутона за запазване. За запазването отговаря класът „ComposeListSaveHelper“, който извлича цялата информация от оформлението на екрана (заглавие, елементи на списъка, подсещане) и я прилага адекватно в базата данни, в зависимост от режима, в който се намира даденият списък. Също така запазва датата на последна промяна на списъка и датата на създаване ако това е нов списък. Ако списъкът е частен преди да се запази в базата данни съдържанието му бива криптирано асинхронно с помощта на класа “EncryptNoteTask”, който наследява “AsyncTask”. При завършване на екрана се установява резултатен (“intent”), чрез който класът стартирал “ComposeListActivity” може правилно да установи своят интерфейс и поведение. За тази цел служи класът “ComposeListResultHandler”. За поддържането на правилното поведение и състояние на класа при смяната на ориентацията на екрана се грижи класът “ComposeListRotationHandler”.

**3.1.4. Display activities:**

За улеснение нека разгледаме “display” екрани в една точка като започнем с базовия клас, който всички от тях наследяват – “DisplayBaseActivity”. В него се връзват всички общи изгледи за екраните изгледи с помощта на 'com.jakewharton:butterknife:7.0.1' (описана в 2.3.1). Настройва се лентата с инструменти и се визуализират общите за всички екрани елементи на бележката като заглавие и детайли (дата на създаване, дата на последна промяна, местоположение и дата на изтичане ако са налични). Ако локацията е налична се поставя слушател за натискане (“OnClickLisntener”), който при кликване стартира намерение (“intent”), показващо точното местоположение на създаване на бележката в приложението “Google Maps” (ако е инсталирано на устройството). Съдържа методи, с които могат да бъдат достъпени вече инициализираните изгледи и метод за даване и отнемане на звезда. Класът “DisplayBaseActivity” има 2 публични (public) полета представляващи булеви променливи, показващи дали бележката има звезда и дали режимът на бележката е променян. За да няма повторение в описанието на екраните първо ще разгледаме базовите класове визуализиращи бележки или списъци – “DisplayNoteBaseActivtiy” и “DisplayListBaseActivtiy”, които наследяват класа “DisplayBaseActivtiy”, който вече споменахме.

Класът “DisplayNoteBaseActivtiy” инициализира всички изгледи използвани за показването на една бележка, използвайки “ButterKnife” (описан подробно в 2.3.1) – текстови изглед за описание на бележката, контейнер за прикрепени изображения, линейно оформление за прикрепено аудио и неговите изгледи. Въвежда екрана в съответното начално състояние , визуализира атрибутите на бележката – описание, прикачен снимков и аудио материал, и тяхното управление. При кликване на изображение го отваря с галерия приложението по подразбиране като, докато то бъде заредено показва прогрес диалог. С помощта на класа “AudioUtils” предлага интерфейс за управление на прикрепеното аудио, ако е налично – трансформиращ се бутон за стартиране / паузиране, и бар показващ прогресът, който е изминал. При минимизиране на приложението ако звуковото е стартирано, автоматично го паузира и запазва прогрес. Грижи се за запазването на общите полета за всички “Display Note” класове при смяна на ориентацията на екрана.

Класът “DisplayListBaseActivity” инициализира всички изгледи използвани за показването на един списък, използвайки “ButterKnife” (описан подробно в 2.3.1) – изгледа от типа “ScrollView”, съдържащ основното съдържание на списъка, който позволява оптимално показване на по-големи списъци върху устройства с по-малко екрани, контейнер за елементите на списъка, които не са зачеркнати, контейнер за зачертаните елементи. Настройва адаптерите отговарящи за контейнерите, в които се добавят елементите на списъка. С помощта на класът “DisplayListBaseTaskHelper” контролира функционалността на бутона за скриване и показване на зачеркнатите елементи от списъка. Поставя дисплей екрана в правилно начално състояние чрез класът “DisplayListBaseStartStateHelper”. Отговаря за запазването на общите полета за всички “Display List” класове при смяна на ориентацията на екрана.

**3.1.4.1. DisplayNoteNormalActivity**

До този екран може да се достигне след като се кликне бележка, която се намира в разделите “All notes”, “Starred” или след натискане на известие от бележка, чийто режим е “MODE\_NORMAL” или „MODE\_IMPORTANT”. Класът “DisplayNoteNormalActivity” непряко наследява класа “DisplayNoteBaseActivity”, което му дава достъп до всички негови методи и полета. Пряко наследен е класът “DisplayNoteEditableActivity”, който обработва резултата при завършването на екрана “ComposeNoteActivity” , когато е стартиран от “DisplayNoteNormalActivity” (това събитие се случва при редактиране на бележки). “DisplayNoteNormalActivity” съдържа функционалността за „заключване на бележка“ , тоест криптирането й, и превръщането в „частна“. За реализацията е използван класът „ActionExecutor”, който е упоменат в 2.3.2 пакетът “com.gcode.notes.ui”. Отговаря за даването и отнемането на звезда на съответната бележка и запазването на промените в базата.

**3.1.4.2. DisplayNotePrivateActivity**

До този екран може да се достигне след като се кликне бележка, която се намира в разделa “Private” или след натискане на известие от бележка, чийто режим е “MODE\_PRIVATE”. Класът “DisplayNotePrivateActivity” непряко наследява класа “DisplayNoteBaseActivity”, което му дава достъп до всички негови методи и полета. Пряко наследен е класът “DisplayNoteEditableActivity”, който обработва резултата при завършването на екрана “ComposeNoteActivity”, когато е стартиран от “DisplayNotePrivateActivity” (това събитие се случва при редактиране на бележки). В този екран се намира функционалността за „отключване на бележка“, тоест променянето на режима й от “MODE\_PRIVATE” (частен) към “MODE\_NORMAL” (нормален) и декриптирането на бележката. За реализацията е използван класът „ActionExecutor”, който е упоменат в 2.3.2 пакетът “com.gcode.notes.ui”.

**3.1.4.3. DisplayNoteBinActivity**

Екранът може да бъде достигнат след като се кликне бележка, която се намира в разделa “Bin” (кошчето) или след натискане на известие от бележка, чийто режим е “MODE\_DELETED\_NORMAL” или “MODE\_DELETD\_IMPORTANT”. Класът “DisplayNoteBinActivity” непряко наследява класа “DisplayNoteBaseActivity”, което му дава достъп до всички негови методи и полета. От екранът “DisplayNoteBinActivity” има възможност бележката да възстанови предишния си режим и да бъде преместена в съответният раздел (“AllNotes” или “Starred”). Това се случва благодарение на класa “ActionExecutor“, който показва необходимите диалогови прозорци за операцията и я изпълнява.

**3.1.4.4. DisplayListNormalActivity**

До този екран може да се достигне след като се кликне списък, който се намира в разделите “All notes”, “Starred” или след натискане на известие от списък, чийто режим е “MODE\_NORMAL” или „MODE\_IMPORTANT”. Класът “DisplayListNormalActivity” непряко наследява класа “DisplayListBaseActivity”, което му дава достъп до всички негови методи и полета. Пряко наследен е класът “DisplayListEditableActivity”, който отговаря за три неща – за запазването на промените в списъка в базата данни преди завършването на екрана и за поставянето за резултатно намерение (“intent”), чрез което класът стартирал екрана да може да се ориентира правилно за направените промени и за обработката на резултата при завършването на екрана “ComposeListActivity”, когато е стартиран от “DisplayListNormalActivity” (това събитие се случва при редактиране на списъци). Отговаря за даването и отнемането на звезда на съответния списък и запазването на промените в базата.

**3.1.4.5. DisplayListPrivateActivity**

Екранът може да бъде достъпен след като се кликне списък, който се намира в разда “Private” или след натискане на известие от списък, чийто режим е “MODE\_PRIVATE” . Класът “DisplayListPrivateActivity” непряко наследява класа “DisplayListBaseActivity”, което му дава достъп до всички негови методи и полета. Пряко наследен е класът “DisplayListEditableActivity”, който отговаря за три неща – за запазването на промените в списъка в базата данни преди завършването на екрана и за поставянето за резултатно намерение (“intent”), чрез което класът стартирал екрана да може да се ориентира правилно за направените промени и за обработката на резултата при завършването на екрана “ComposeListActivity”, когато е стартиран от “DisplayListNormalActivity” (това събитие се случва при редактиране на списъци). В този екран се намира функционалността за „отключване на списък“, тоест променянето на режима му от “MODE\_PRIVATE” (частен) към “MODE\_NORMAL” (нормален) и декриптирането на списъка. За реализацията е използван класът „ActionExecutor”.

**3.1.4.6. DisplayListBinActivity**

Екранът може да бъде достигнат след като се кликне списък, който се намира в разделa “Bin” (кошчето) или след натискане на известие от списък, чийто режим е “MODE\_DELETED\_NORMAL” или “MODE\_DELETD\_IMPORTANT”. Класът “DisplayListBinActivity” непряко наследява класа “DisplayNoteBaseActivity”, което му дава достъп до всички негови методи и полета. От екранът “DisplayListBinActivity” има възможност списъкът да бъде възстановен в предишния си режим и да бъде преместен в съответният раздел (“AllNotes” или “Starred”). Това се случва благодарение на класa “ActionExecutor“, който показва необходимите диалогови прозорци за операцията и я изпълнява. Класът “DisplayListBinActivity” отговаря за изключването на кликването върху елемент от списъка, което в другите раздели го зачерква или премества с контейнера с неизпълнени елементи от списъка.

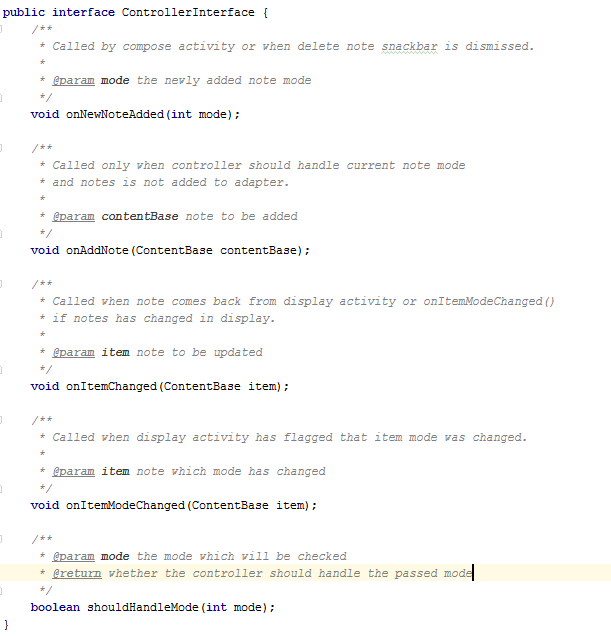
**3.2. Основни помощни класове, използвани за реализацията на екраните на приложението**

**3.2.1. Класът “ActionExecutor”:**

Този клас отговаря за изпълняването на действия в екраните на приложението и изграждането на съответният потребителски интерфейс, свързан с тях. За реализирането на функционалността си класът “com.gcode.ui.ActionExecutor” използва помощните класове “DialogBuilder” и “SnackbarHelper” от пакетът “com.gcode.ui.helpers” (упоменат в 2.3.2). Действията, които класът изпълнява са: показване на “snackbar” с опция за отменяне на операцията при изтриване на бележка от “MainActivity”, показване на “snackbar” с опция за отменяне на операцията при изтриване на елемент на списък от “ComposeListActivity”, изпразване на кошчето (раздела “Bin”), изтриване или възстановяване на всички видове бележки и списъци (частни, обикновени и със звезда), отключване или заключване на бележки и списъци в различни режими, и изтриване или добавяне на снимков материал към бележка.

**3.2.2. Класът “BaseController”**

Това е класът отговарящ съдържанието и поведението на главният екран – “MainActivity”. В своята същност “BaseController” е един “singleton” кла, чиято инстанция може да бъде достъпена от всеки един екран на приложението. За първоначалното инстанцииране на класа “BaseController” е необходима инстанция на класа “MainActivity”, от която контролера да може да получи достъп до полетата, необходими за управлението на главния екран. Имплементира интерфейса “ControllerInterface”, който съдържа „обажданията“ (callbacks) използвани за контролирането на разделите на приложението показани на „Фигура 3.2.1“.



„Фигура 3.2.1“

Класът “BaseController” реализира в себе си подмяната на съдържанието в главния контейнер при смяна на раздел и анимирането на прехода. Също съдържа методи добавяне, обновяне на елемент към адаптера на главния контейнер (класът „MainAdapter”).

**3.2.3. Класът “DatabaseController”**

Този клас осъществява връзката между екраните и базата данни. За инстациирането му е необходим контекстът на приложението. Класът съдържа две полета – mContext (контекстът на приложението) и mDatabase (поле от типа SQLiteDatabase, чрез което се изпълнят заявки към базата). При първото стартиране на приложението базата данни се създава с помощта на класа “NotesDbHelper”, който съдържа необходимите заявки за създаването, изтриването и мигрирането на базата. Именно чрез него се инциализира полето mDatabase. Класът “DatabaseController” предоставя методи за избиране на различни по брой и режим бележки и списъци от базата данни, добавя, редактиране и изтриване на вече съществуващи такива.