

ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

ДИПЛОМНА РАБОТА

Тема: Мобилно приложение за водене на бележки

Дипломант:

Илиян Емилов Германов

Научен ръководител:

ст. преп . Любомир Чорбаджиев

С О Ф И Я

2 0 1 6

Съдържание

Увод	5
I Глава	7
Проучвателна част. Преглед на съществуващи продукти, развойни среди и развойни средства.	7
1.1. Преглед на съществуващи продукти	7
1.2 Развойни среди	16
1.2.1 ADT	16
1.2.2 “Android studio”	17
1.2.3 SDK	18
1.3 Развойни средства	18
1.3.1 Основни компоненти на Android приложение	18
II глава	19
Изисквания към програмния продукт. Избор на езика за програмиране и софтуерните средства.	
Описание на структурата на програмата и базата данни	19
2.1. Изисквания към програмния продукт	19
2.2. Избор на език за програмиране и софтуерните средства	19
2.3. Структура на приложението	20
2.3.1. Зависимости (dependencies) на проекта	20
2.3.2. Пакети на приложението	22
2.4. Структура на базата данни	25
III глава	30
Описание на начина на реализация на приложението	30
3.1. Екрани на приложението	30
3.1.1. MainActivity	31
3.1.2. ComposeNoteActivity	34
3.1.3. ComposeListActivity	37
3.1.4. Display activities:	39
3.2. Основни помощни класове, използвани за реализацията на екраните на приложението	49
3.2.1. Класът “ActionExecutor”	49
3.2.2. Класът “BaseController”	50
3.2.3. Класът “DatabaseController”	51

IV глава	53
Ръководство на потребителя	53
4.1. Main (главен екран).....	53
4.2. Екрани за показване (Display activities)	60
4.2.1. Екран за показване на бележка (Display note).....	60
4.2.2. Екран за показване на списъци (Display list)	61
4.3. Екрани за композиране на бележки списъци (Compose note / list).....	62
4.3.1. Екран за композиране на бележка (Compose note)	62
4.3.2. Екран за композиране на списък (Compose list)	63
Заключение.....	64
Използвана литература	65

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

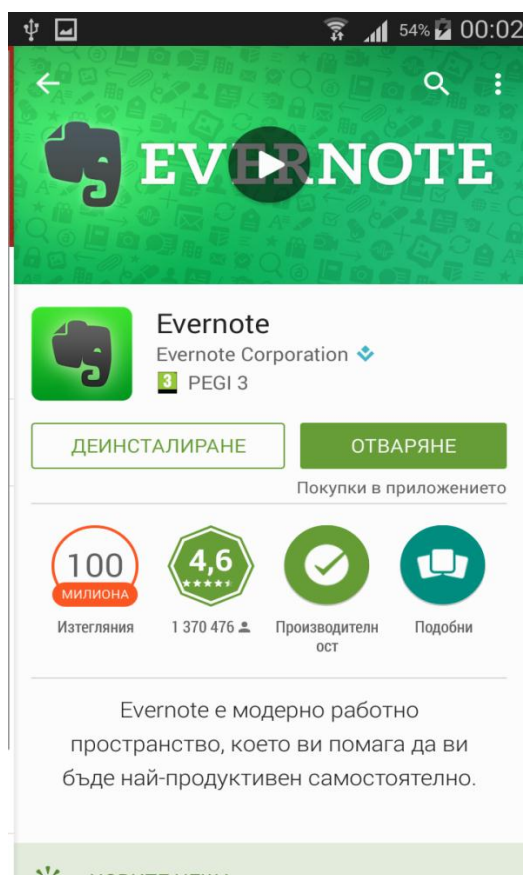
камерата или избрана от галерията); създаване на списъци; създаване на гласова бележка, която се транскрибира автоматично; раздел за важни бележки; раздел за частни бележки, които се криптират и се достъпват с парола; кошче; поставяне на подсещане. За целта ще бъде направено проучване, анализ и сравнение на съществуващи подобни продукти.

I Глава

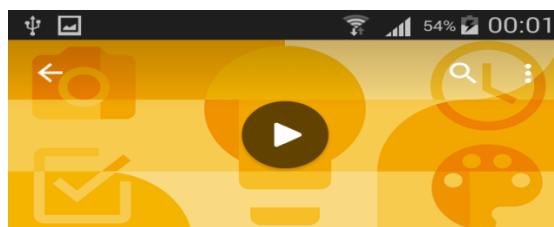
Проучвателна част. Преглед на съществуващи продукти, развойни среди и развойни средства.


1.1. 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”





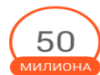
Google Keep: бележки и списъци

Google Inc. 


 PEGI 3

ДЕИНСТАЛИРАНЕ


ОТВАРЯНЕ




Изтегляния



471 762




Производителност

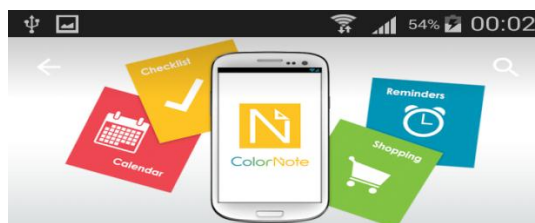



Подобни

Google Keep


НОВИТЕ НЕЩА
 * Поправки на програмни грешки и подобрения в ефективността.


“Google Keep в Google Play”





ColorNote бележки notepad

Notes

 PEGI 3

ИНСТАЛИРАНЕ



Изтегляния



1 612 881



Производителност



Подобни

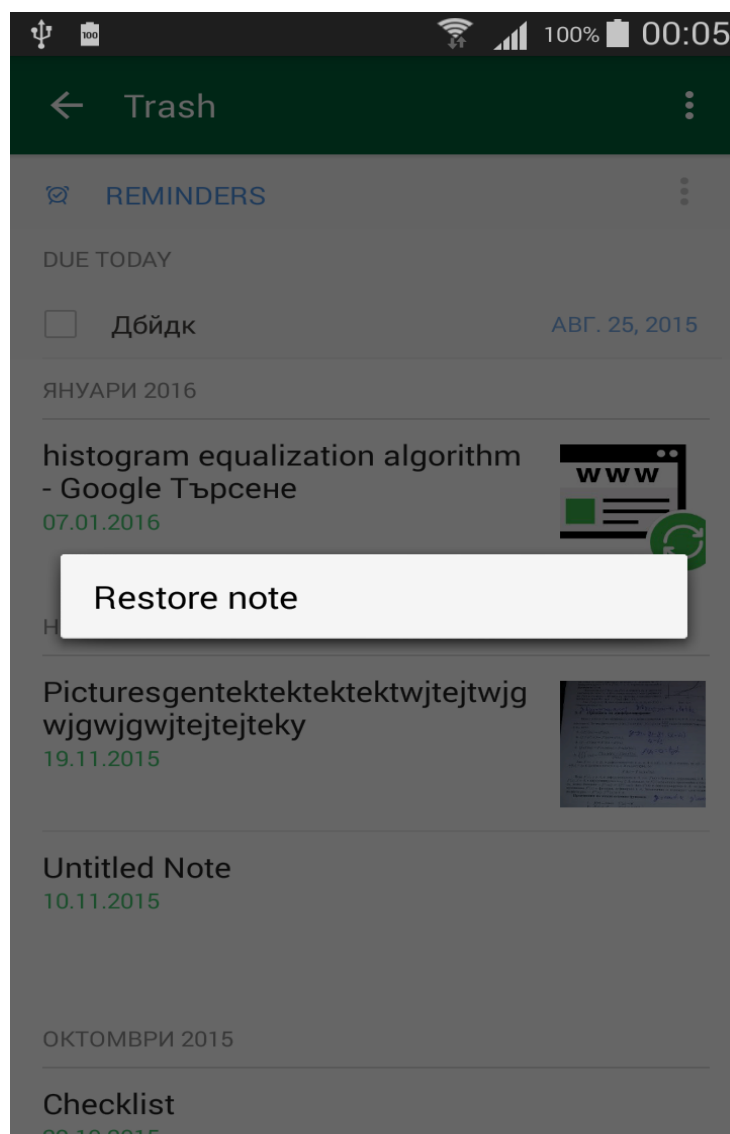
ColorNote - Лесен за използване тефтерче за бележки, списък със задачи, календар

[ПРОЧЕТЕТЕ ОЩЕ](#)

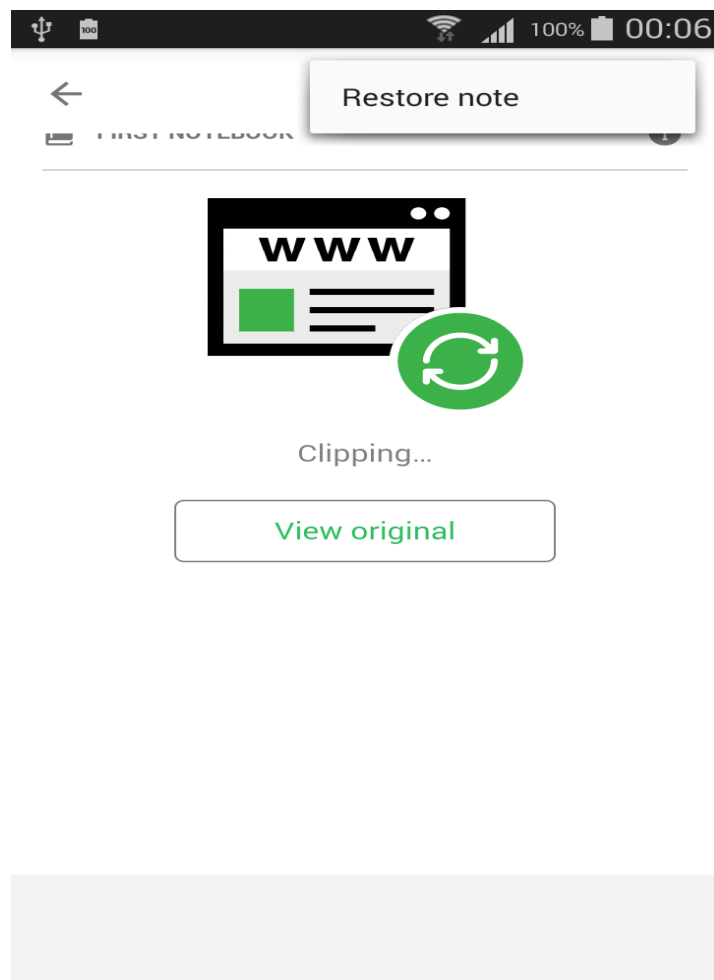
“ColorNote в Google Play”

Освен, че приложенията за водене на бележки достигат до много и различни по пол и възраст потребители, те остават инсталирани на мобилните им устройства сравнително по-дълго от другите видове приложения. 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 приложения е голям, но смятам, че може да бъде проектирано ново, което е да бъде полезно за използване, бързо и ефикасно. Идеята за настоящата ми дипломна работа се роди лятото, когато се нуждаех от приложение за удобно записване на прости 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 изтрие 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 нямаш 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“:



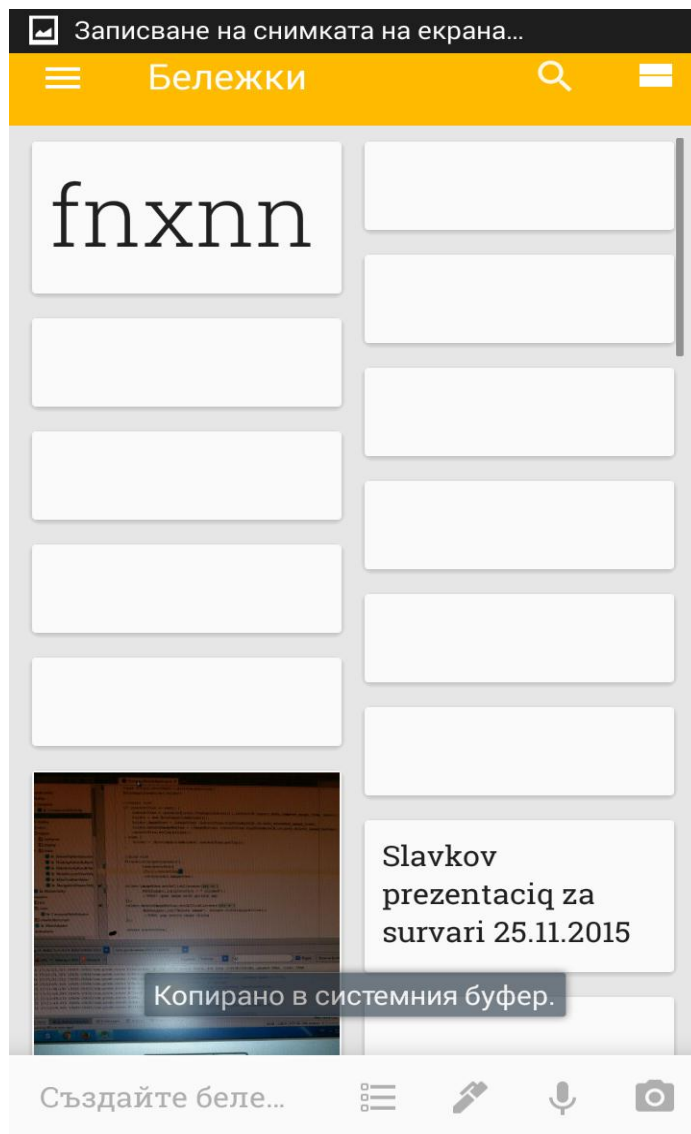
“Long click върху item в Trash”



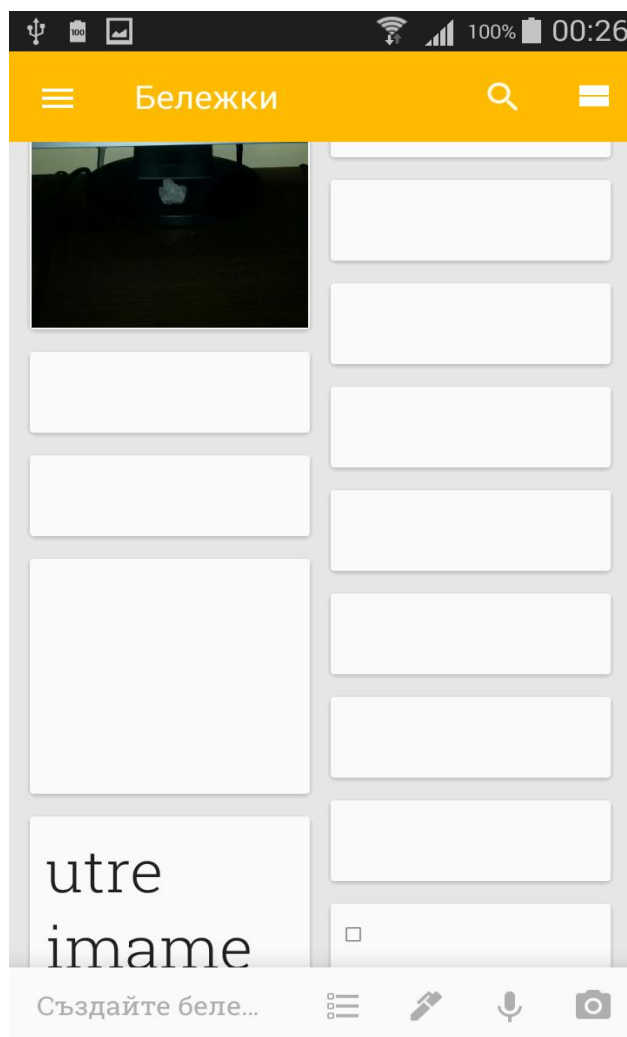
“Отворено меню от item в Trash”

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”



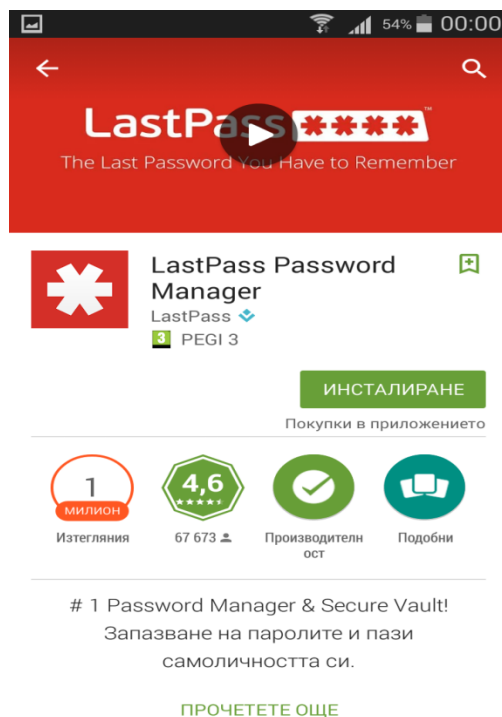
“Празни бележки в Google Keep (2)”

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“.

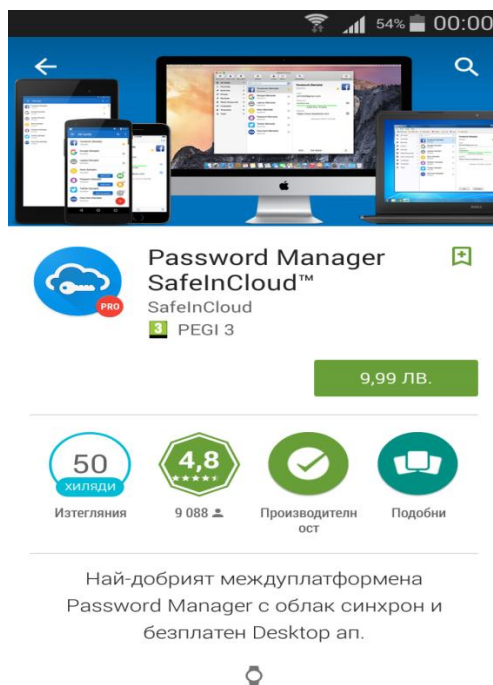
Друга важна за мен функционалност, която не открих в гореспоменатите водещи приложения е създаването на „private“ (частни) бележки, които да не могат да се видят от всеки, който има достъп до мобилното устройство. Частните бележки биха могли да се използват за много цели и биха направили приложението 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 свалени 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 емулятор за него. Подобно на “Eclipse” 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 да 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 на записките – „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 изтрети 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, 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 познанията ми биха помогнали 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“ приложения, 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 описание, нека ги разделим на два вида – „com.android.support“ (официални зависимости, невлизащи в SDK-a) и „3rd party“ (както от името става ясно – библиотеки, принадлежащи на трети лица, пуснати с лиценз за свободно 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 класа „ListView“ от Android SDK-a, но с тази разлика, че в него е имплементиран „viewholder pattern“, което води до по-добро представяне при голям обем от данни
- 'com.android.support:cardview-v7:23.1.1' – съдържа класа „CardView“, който представлява повърхност с „Material Design“
- 'com.android.support:design:23.1.1' – помощна библиотека за дизайн, съдържаща основните практики за потребителски интерфейс в Android
- 'com.google.code.gson:gson:2.4' – не влиза изцяло в този тип, но пък е една 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 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 приложението

Приложението е разделено в 15 главни пакета, всеки от които отговаря за отделен проблем.

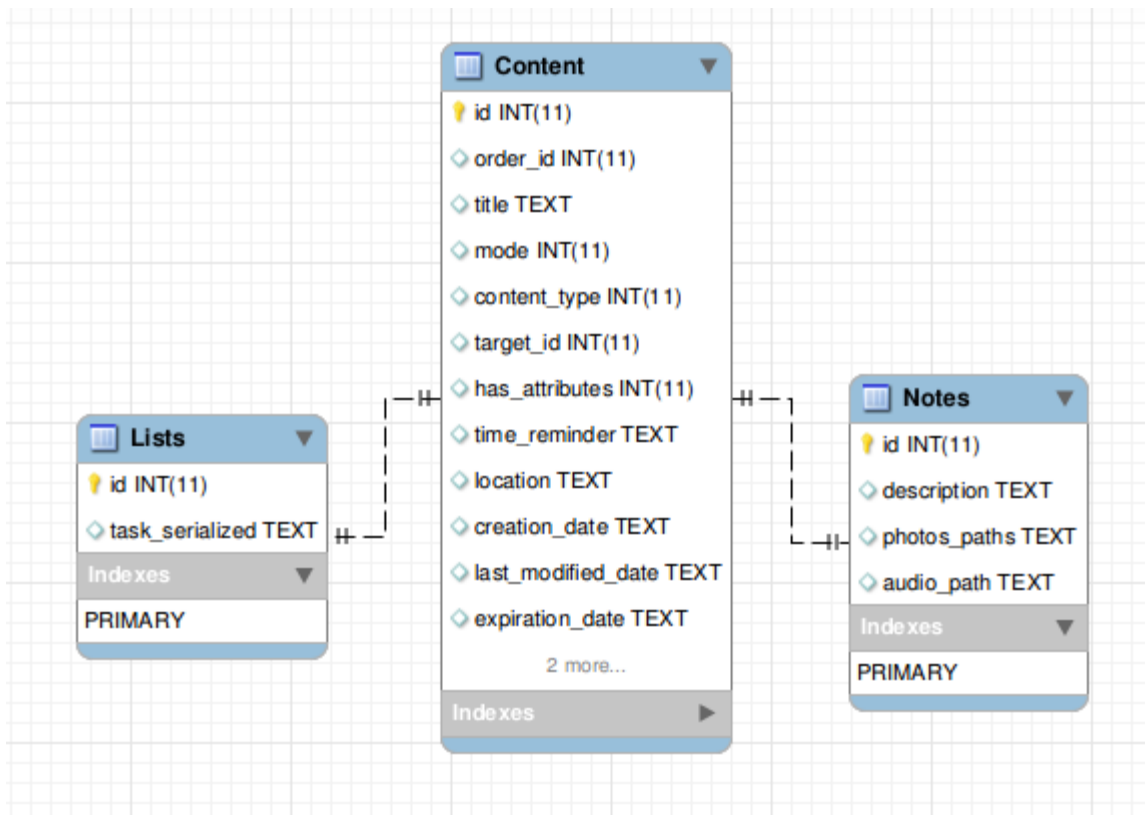
- Пакетът „com.gcode.notes“:
Главен пакет на приложението, съдържащ всички останали пакети.
- Пакетът „com.gcode.notes.activities“:
В този пакет се съдържат всички Activity класове на проекта и техните помощни класове.
- Пакетът „com.gcode.notes.adapters“:
Пакет, съдържащ всички адаптери, използвани 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 инстанцирането на други класове).
- Пакетът „com.gcode.notes.fragments“:
В този пакет се съдържат фрагментите, 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 реструктуриран.
- 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 класа „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

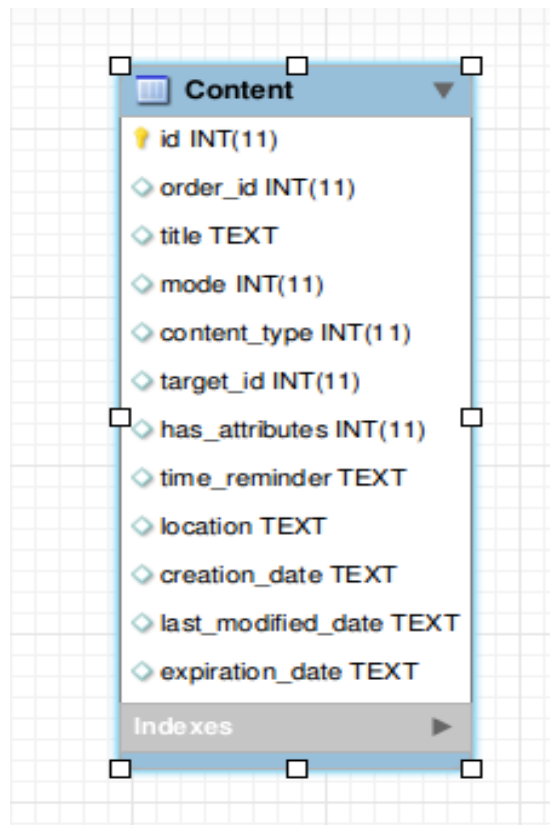
За база данни приложението използва „SQLite“ със структура представена на „Фигура 2.4.1“:



„Фигура 2.4.1“

Базата данни се състои от следните 3 таблици – „Content“ (съдържа общата информацията за бележка и списък), „Notes“ (съдържа атрибутите, които има една бележката), „Lists“ (съдържа атрибутите, които има един списък).

Таблицата „Content” („Фигура 4.2.2“):

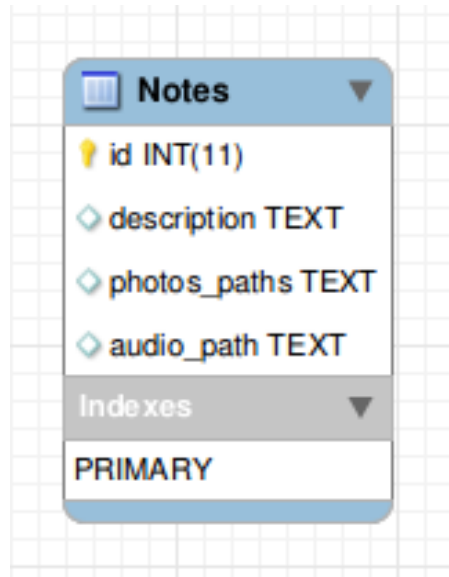


„Фигура 4.2.2“

- 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, служещ за връзка с таблицата за атрибути на записа („Notes“ / „Lists“)
- attributes – флаг, показващ дали даденият запис има атрибути и дали е 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 режим „DELETED“ той получава дата на изтичане след, която бива автоматично изтрит при следващото пускане на приложението), може да бъде NULL

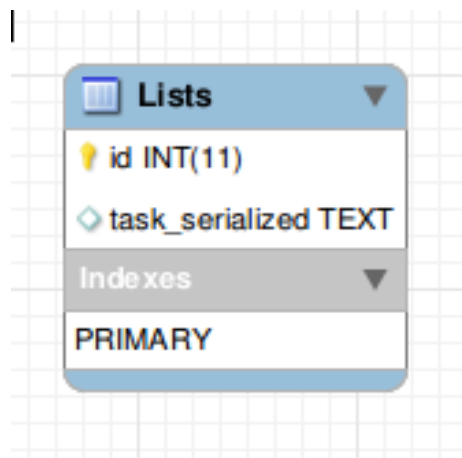
Таблицата “Notes” („Фигура 4.2.3“):



„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 прикрепения звуков файл към бележката, може да бъде NULL

Таблицата „Lists“ („Фигура 4.2.4“):



„Фигура 4.2.4“

- `_id` – първичен ключ на таблицата, уникален идентификационен номер на записа
- `tasks_serialized` – сериализиран лист от инстанции на класа „`ListDataItem`“, представляващ 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 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 споменем 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

```
public class MainActivity extends AppCompatActivity {

    ...

    @Bind(R.id.main_toolbar)
    Toolbar mToolbar;
    @Bind(R.id.main_app_bar_layout)
    AppBarLayout mAppBarLayout;
    @Bind(R.id.main_drawer_layout)
    DrawerLayout mDrawerLayout;
    @Bind(R.id.main_coordinator)
    CoordinatorLayout mCoordinatorLayout;
    @Bind(R.id.main_navigation_drawer)
    NavigationView mDrawer;
    @Bind(R.id.fab_menu)
    FloatingActionMenu mFabMenu;
    @Bind(R.id.main_content_recycler_view)
    RecyclerView mRecyclerView;
    @Bind(R.id.main_notes_recycler_view_empty_text_view)
    TextView mRecyclerViewEmptyView;

    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ButterKnife.bind(this);
        MainActivityRotationHandler.handleScreenRotation(this, savedInstanceState);
        setup();

        ...
    }

    @Override
    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);
        if (mReminderNotificationStartHelper == null) {
            //secure
            mReminderNotificationStartHelper = new
ReminderNotificationStartHelper(this);
        }

        mReminderNotificationStartHelper.handleIfStartedFromReminderNotification(intent);
    }

    private void setup() {
        mDrawerOptionExecutor = new DrawerOptionExecutor(this);
        new MainToolbarHelper(this).setupToolbar();
        FabMenuHelper.setupFabMenu(this);
        new NavDrawerHelper(this, mDrawerOptionExecutor).setupNavigationDrawer();
        new MainRecyclerViewHelper(this).setupRecyclerView();

        ...
    }
    ...
}
```

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“, който 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 скрита за потребителя и той има възможност да я види чрез „придърпване“ от някоя страна на екрана (в повечето случаи отлясно). 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 е наясно потребителя 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 чести 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 лента с инструменти („Toolbar“); контейнер, чрез който се визуализират бележките за избрания раздел („RecyclerView“); текстов изглед, който бива показван, само когато контейнерът 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 съдържа четири „mini fab“ бутона, при избирането на които се извиква методът „handlerItemClick(MainActivity, View)“ от класа „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 на потребителя, след което с помощта на класа „MainActivityResultHandler“ се зарежда „ComposeNoteActivity“ с автоматично въведено описание, отговарящо на казаното от потребителя. Езикът за разпознаване се извлича от настройките за език на устройство; четвъртият бутон – показва диалогов прозорец с опции за заснемане / избиране на снимков материал, който да бъде прикрепен към бележката. След успешно намиране на снимков материал с помощта на класа „MainActivityResultHandler“ се стартира „ComposeNoteActivity“ с автоматично добавен избрания или 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 и 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 или 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“ от пакета „com.google.gson“, описан в 2.3.2). При жеста „scrolling“ върху контейнера, бутонът контролиращ менюто за създаване на бележки и лентата с инструменти се скриват или показват, съобразно посоката на жеста, което улеснява и подобрява преглеждането на елементите. За зареждането и контролирането на различните раздели отговарят класовете от пакета „com.gcode.notes.controllers“ (описан в 2.3.2). За поддържането на правилното поведение и състояние на екрана при смяната на ориентацията на устройството се грижи класът „MainActivityRotationHandler“.

3.1.2. ComposeNoteActivity

```
public class ComposeNoteActivity extends ComposeBaseActivity {
    @Bind(R.id.compose_note_description_edit_text)
    EditText mDescriptionEditText;
    @Bind(R.id.compose_note_images_linear_list_view)
    LinearLayout mImagesLinearLayout;
    @Bind(R.id.compose_note_audio_layout)
    LinearLayout mAudioLayout;
    @Bind(R.id.compose_audio_play_pause_button)
    ImageButton mAudioPlayPauseButton;
    @Bind(R.id.compose_audio_progress_bar)
    ProgressBar mAudioProgressBar;
    @Bind(R.id.compose_audio_duration_text_view)
    TextView mAudioDurationTextView;

    ...

    public NoteData mNoteData;
    public AudioUtils mAudioUtils;
    public ComposeNoteImagesAdapter mImagesAdapter;
    public MaterialDialog mOpenImageInGalleryProgressDialog;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_compose_note);
        ButterKnife.bind(this);
        setup(savedInstanceState);
    }

    private void setup(Bundle savedInstanceState) {
        super.setup(); //setups base in ComposeBaseActivity
        new ComposeNoteStartStateHelper(this).setupStartState(savedInstanceState);
        if (!mIsOpenedInEditMode) {
            //its new note, obtain creation location if possible
            ComposeLocationHelper.getLocation(this);
        }
    }

    ...

    @SuppressWarnings("unused")
    @OnClick(R.id.compose_audio_play_pause_button)
    public void playPauseAudio() {
        mAudioUtils.toggle();
    }

    @SuppressWarnings("unused")
    @OnClick(R.id.compose_audio_delete_button)
    public void deleteAudio() {
        ActionExecutor.deleteAudioFromNote(this);
    }

    ...
}
```

Този екран се стартира при създаването на нова бележка или при редактирането на вече съществуваща такава. Класът „ComposeNoteActivity“ 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 две деца – „AppBarLayout“ и „ScrollView“. В „AppBarLayout“ се разполага лента с инструменти, която дава възможност на потребителя да добавя изображение към бележката, и бутон за запазване 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 или премахва снимков или звуков материал, вдига съответните флагове, показващи в какъв режим се намира екрана (създаване на нова бележка или редактиране на вече съществуваща такава; дали бележката е частна или не; дали режимът на бележката е променен – даване или 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-точния 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 ѝ 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 своя 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

```
public class ComposeListActivity extends ComposeBaseActivity {
    public ListComposeContainerAdapter mContainerAdapter;
    public TickedListComposeContainerAdapter mTickedContainerAdapter;
    public ListData mListData;
    @Bind(R.id.compose_list_scroll_view)
    ScrollView mScrollView;
    @Bind(R.id.compose_list_last_divider_view)
    View mLastDividerView;
    @Bind(R.id.compose_list_container_layout)
    DragLinearLayout mContainer;
    @Bind(R.id.compose_list_container_ticked_layout)
    DragLinearLayout mTickedContainer;

    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_compose_list);
        ButterKnife.bind(this);
        setup(savedInstanceState);
    }

    private void setup(Bundle savedInstanceState) {
        super.setup();

        ...
        new ComposeListStartStateHelper(this).setupStartState(savedInstanceState);
        if (!mIsOpenedInEditMode) {
            //its new note, obtain creation location if possible
            ComposeLocationHelper.getLocation(this);
        }
    }

    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        ComposeListRotationHandler.saveInstanceState(this, outState);
    }

    @OnClick(R.id.compose_list_add_list_item_text_view)
    public void addListInputItem() {
        //cast it to String in order to not be ambiguous
        mContainerAdapter.addInputItem((String) null, true);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_compose_list, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        return super.onOptionsItemSelected(item) ||
        ComposeListMenuOptionsHelper.optionsItemSelected(this, item);
    }
}
```

До този екран може да се достигне при създаването на нова бележка или при редактирането на вече съществуваща такава. Класът

„ComposeListActivity“ 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 две деца – „AppBarLayout“ и „ScrollView“. В „AppBarLayout“ се разполага лента с инструменти, която има бутон за запамятаване на направените промени. Контейнерът „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 добавяне 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 или редактиране на вече съществуваща такъв; дали списъкът е частен или не; дали режимът на списъкът е променен – даване или 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-точния снабдител, снет от предпочитанията на потребителя („GPS Provider“, „Network Provider“). Всеки елемент от списъка е представен от класа „ListItem“, който има две полета – булева променлива, показваща дали даденият елемент е маркиран, и символен низ, държащ в себе си съдържанието на елемента. Оформлението в този екран дава възможност за 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 своя 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“.

```

public class DisplayBaseActivity extends AppCompatActivity {
    public boolean mIsStarred;
    public boolean mNoteModeChanged;

    @Bind(R.id.display_toolbar)
    Toolbar mToolbar;
    @Bind(R.id.display_title_text_view)
    TextView mTitleTextView;
    @Bind(R.id.display_dates_text_view)
    TextView mDatesTextView;
    @Bind(R.id.display_reminder_text_view)
    TextView mReminderTextView;
    @Bind(R.id.display_location_button)
    Button mLocationButton;
    @Bind(R.id.display_action_image_button)
    ImageButton mActionImageButton;

    ...

    public void setStarredState() {
        mIsStarred = true;
        mActionImageButton.setImageResource(R.drawable.ic_star_orange_36dp);
    }

    public void setNotStarredState() {
        mIsStarred = false;
        mActionImageButton.setImageResource(R.drawable.ic_star_border_black_36dp);
    }

    protected void setup() {
        DisplayToolbarHelper.setupToolbar(this, mToolbar); //setup activity's toolbar
    }

    public void displayBase(final ContentBase contentBase) {
        mTitleTextView.setText(contentBase.getTitle()); //display note's title
        mDatesTextView.setText(contentBase.getDateDetails()); //display Creation, Last
        modified and if has Expiration dates
        if (contentBase.hasReminder()) {
            //there is reminder, display it
            mReminderTextView.setVisibility(View.VISIBLE); //reminder text view is
            currently gone
            ...
        } else {
            //note hasn't reminder, hide it
            mReminderTextView.setVisibility(View.GONE);
        }
        if (contentBase.hasLocation()) {
            //there is location, display it
            mLocationButton.setVisibility(View.VISIBLE); //by default is gone
            new DecodeLocTask(mLocationButton).execute(contentBase.getMyLocation());
            ...
        }
    }
}

```

В него се връзват всички общи изгледи за екраните 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 променен. За да няма повторение в описанието на екраните първо ще разгледаме базовите класове, визуализиращи бележки или списъци – „DisplayNoteBaseActivitiy“ и „DisplayListBaseActivitiy“, които наследяват класа „DisplayBaseActivitiy“, който вече споменахме.

Класът „DisplayNoteBaseActivitiy“ 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, класът го отваря с приложението галерия, зададено по подразбиране на мобилното устройство, 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 или паузиране, и бар, показващ прогреса, който е изминал. При минимизиране на приложението, ако звуковото е стартирано, автоматично го паузира и запазва прогреса. 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 зачеркнатите 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

```
public class DisplayNoteNormalActivity extends DisplayNoteEditableActivity {

    @Override
    public void displayNoteData() {
        super.displayNoteData();
        if (mNoteData.isImportant()) {
            setStarredState();
        } else {
            setNotStarredState();
        }
    }

    @SuppressWarnings("unused")
    @OnClick(R.id.display_action_image_button)
    public void starImageButtonClicked() {
        if (mIsStarred) {
            setNotStarredState();
        } else {
            setStarredState();
        }
        mNoteData.setModeImportant(mIsStarred);
        mNoteModeChanged = !mNoteModeChanged;
        MyApplication.getWritableDatabase().updateNoteMode(mNoteData);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        return super.onOptionsItemSelected(item) ||
            DisplayBaseNormalOptionsHelper.optionItemSelected(this, item,
mNoteData);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_display_note_normal, menu);
        return true;
    }
}
```

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 „заклучване на бележка“, тоест криптирането ѝ, 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

```
public class DisplayNotePrivateActivity extends DisplayNoteEditableActivity {

    @Override
    public void displayNoteData() {
        super.displayNoteData();
        getActionImageButton().setImageResource(R.drawable.ic_lock_open_black_24dp);
    }

    @SuppressWarnings("unused")
    @OnClick(R.id.display_action_image_button)
    public void starImageButtonClicked() {
        ActionExecutor.unlockNote(this, mNoteData);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_display_note_private, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        return super.onOptionsItemSelected(item) ||
            DisplayBasePrivateOptionsHelper.optionItemSelected(this, item,
mNoteData);
    }
}
```

До този екран може да се достигне след като се кликне бележка, която се намира в раздела „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 ѝ 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

```
public class DisplayNoteBinActivity extends DisplayNoteBaseActivity {

    @Override
    public void displayNoteData() {
        super.displayNoteData();
        getActionImageButton().setImageResource(R.drawable.ic_restore_deleted_24dp);
    }

    @SuppressWarnings("unused")
    @OnClick(R.id.display_action_image_button)
    public void restoreNote() {
        ActionExecutor.restoreDeletedNote(this, mNoteData);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_display_note_bin, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        return super.onOptionsItemSelected(item) ||
            DisplayNoteBinMenuOptionsHelper.optionItemSelected(this, item);
    }
}
```

Екранът може да бъде достигнат след като се кликне бележка, която се намира в раздела „Bin“ (кошчето) или след натискане на известие от бележка, чийто режим е „MODE_DELETED_NORMAL“ или „MODE_DELETED_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 съответния раздел („AllNotes“ или „Starred“). Това се случва благодарение на класа „ActionExecutor“, който показва необходимите диалогови прозорци за операцията и я изпълнява.

3.1.4.4. DisplayListNormalActivity

```
public class DisplayListNormalActivity extends DisplayListEditableActivity {

    @Override
    public void displayListData() {
        super.displayListData();
        if (mListData.isImportant()) {
            setStarredState();
        } else {
            setNotStarredState();
        }
    }

    @SuppressWarnings("unused")
    @OnClick(R.id.display_action_image_button)
    public void starImageButtonClicked() {
        if (mIsStarred) {
            setNotStarredState();
        } else {
            setStarredState();
        }
        mListData.setModeImportant(mIsStarred);
        mNoteModeChanged = !mNoteModeChanged;
        MyApplication.getWritableDatabase().updateNoteMode(mListData);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        return super.onOptionsItemSelected(item) ||
            DisplayBaseNormalOptionsHelper.optionItemSelected(this, item,
mListData);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_display_list_normal, menu);
        return true;
    }
}
```

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 на 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

```
public class DisplayListPrivateActivity extends DisplayListEditableActivity {
    @Override
    public void displayListData() {
        super.displayListData();
        getActionImageButton().setImageResource(R.drawable.ic_lock_open_black_24dp);
    }

    @SuppressWarnings("unused")
    @OnClick(R.id.display_action_image_button)
    public void starImageButtonClicked() {
        ActionExecutor.unlockNote(this, mListData);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_display_list_private, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        return super.onOptionsItemSelected(item) ||
            DisplayBasePrivateOptionsHelper.optionItemSelected(this, item,
mListData);
    }
}
```

Екранът може да бъде достъпен след като се кликне списък, който се намира в раздела „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 на 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

```
public class DisplayListBinActivity extends DisplayListBaseActivity {
    @Override
    public void displayListData() {
        super.displayListData();
        getActionImageButton().setImageResource(R.drawable.ic_restore_deleted_24dp);
    }

    @Override
    public void setupLinearListViews(boolean isDeactivated) {
        super.setupLinearListViews(true);
        //TODO: onItemClick show restore note snackbar
    }

    @SuppressWarnings("unused")
    @OnClick(R.id.display_action_image_button)
    public void restoreNote() {
        ActionExecutor.restoreDeletedNote(this, mListData);
    }

    @Override
    public void finish() {
        DisplayListBaseResultHandler.setResult(this);
        super.finish();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_display_list_bin, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        return super.onOptionsItemSelected(item) ||
        DisplayListBinMenuOptionsHelper.optionItemSelected(this, item);
    }
}
```

Екранът може да бъде достигнат след като се кликне списък, който се намира в раздела „Bin“ (кошчето) или след натискане на известие от списък,

чийто режим е „MODE_DELETED_NORMAL“ или „MODE_DELETED_IMPORTANT“. Класът „DisplayListBinActivity“ непряко наследява класа „DisplayListBaseActivity“, 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 класа „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 пакета „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 от списъка от „ComposeListActivity“, изпразване на кошчето (раздела „Bin“), изтриване или възстановяване на всички видове бележки и списъци, отключване или заключване на бележки и списъци в различни режими, и изтриване или добавяне на снимков материал към бележка. На фрагмента по-долу е представена част от реализацията на класа.

```
public class ActionExecutor {

    public static void popNoteDeletedSnackbar(MainAdapter adapter, int position,
ContentBase note) {

        NoteDeletedUndoOnClickListener undoOnClickListener =
            new NoteDeletedUndoOnClickListener(position, note);
        Snackbar.Callback snackbarCallback =
            new NoteDeletedSnackbarCallback(adapter, note, position,
undoOnClickListener);
        SnackbarHelper.buildNoteDeletedSnackbar(adapter.getRootView(),
            undoOnClickListener, snackbarCallback).show();
    }

    public static void popListItemDeletedSnackbar(BaseComposeContainerAdapter
containerAdapter, View removedItem) {
        ListItemDeletedUndoOnClickListener undoOnClickListener =
            new ListItemDeletedUndoOnClickListener(containerAdapter, removedItem);

        SnackbarHelper.buildListItemDeletedSnackbar(containerAdapter.getRootScrollView(),
undoOnClickListener).show();
    }

    public static void emptyRecyclerBin(Activity activity) {
        SingleButtonCallback emptyRecyclerBinCallback = new
EmptyRecyclerBinCallback();
        DialogBuilder.buildEmptyBinDialog(activity, emptyRecyclerBinCallback);
    }

    ...
}
```

3.2.2. Класът „BaseController“

Това е класът, отговарящ за съдържанието и поведението на главния екран – „MainActivity“. В своята същност „BaseController“ е един „singleton“ клас, чиято инстанция може да бъде достъпена от всеки един екран на приложението. За първоначалното инстанциране 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, представена в следния фрагмент.

```
public interface ControllerInterface {  
    /**  
     * Called by compose activity or when delete note snackbar is dismissed.  
     *  
     * @param mode the newly added note mode  
     */  
    void onNewNoteAdded(int mode);  
  
    /**  
     * Called only when controller should handle current note mode  
     * and notes is not added to adapter.  
     *  
     * @param contentBase note to be added  
     */  
    void onAddNote(ContentBase contentBase);  
  
    /**  
     * Called when note comes back from display activity or onItemModeChanged()  
     * if notes has changed in display.  
     *  
     * @param item note to be updated  
     */  
    void onItemChanged(ContentBase item);  
  
    /**  
     * Called when display activity has flagged that item mode was changed.  
     *  
     * @param item note which mode has changed  
     */  
    void onItemModeChanged(ContentBase item);  
  
    /**  
     * @param mode the mode which will be checked  
     * @return whether the controller should handle the passed mode  
     */  
    boolean shouldHandleMode(int mode);  
}
```

Класът „BaseController“ реализира в себе си подмяната на съдържанието в главния контейнер при смяна на раздел и анимирането на прехода. Също

съдържа методи за добавяне и обновяване на елемент към адаптера на главния контейнер (класът „MainAdapter”). Показани в следния фрагмент.

```
public abstract class BaseController implements ControllerInterface {
    ...

    public void setNewContent(ArrayList<ContentBase> newContent, boolean scrollToTop) {
        MainAdapter mainAdapter = getMainAdapter();
        if (mainAdapter != null) {
            mainAdapter.updateContent(newContent);
            //mRecyclerView.invalidate();
            if (scrollToTop) {
                mRecyclerView.smoothScrollToPosition(0);
            }
        }
    }

    public void addItemAsFirst(ContentBase item) {
        MainAdapter mainAdapter = getMainAdapter();
        if (mainAdapter != null) {
            mainAdapter.addItem(0, item);
            mRecyclerView.smoothScrollToPosition(0);
            if (SearchViewHelper.isSearchViewOpened(mMainActivity)) {
                //search view is open, callback to update
                mMainActivity.mSearchHandler.onNewItemAdded(item);
            }
        }
    }

    public void updateItem(ContentBase item) {
        MainAdapter mainAdapter = getMainAdapter();
        if (mainAdapter != null) {
            if (mainAdapter.updateItem(item)) {
                //item was updated successfully, if search is started callback
                if (SearchViewHelper.isSearchViewOpened(mMainActivity)) {
                    //search view is opened, callback
                    mMainActivity.mSearchHandler.onItemChanged(item);
                }
            } else {
                //item not updated, so it not exists, add it
                onAddNote(item);
            }
        } else {
            MyDebugger.log("Failed to update item, mainAdapter is null.");
        }
    }
    ...
}
```

3.2.3. Класът „DatabaseController“

Този клас осъществява връзката между екраните и базата данни. За инсталирането 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 инициализира 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. За осъществяването на функционалностите на класа е използван помощният пакет „com.gcode.notes.database.extras“. На фрагмента от код е показана реализацията на част от изброените методи.

```
public class DatabaseController {
    private Context mContext;
    private SQLiteDatabase mDatabase;

    public DatabaseController(Context context) {
        mContext = context;
        NotesDbHelper mHelper = new NotesDbHelper(context);
        mDatabase = mHelper.getWritableDatabase();
    }

    public ArrayList<ContentBase> getAllVisibleNotes() {
        Cursor cursor = getCursorForAllNoteForModes(Constants.MODE_NORMAL,
Constants.MODE_IMPORTANT);
        return DataBuilder.buildItemList(mDatabase, cursor);
    }

    /**
     * Inserts note in database. !WARN: Note order id and target id are NOT set to
contentBase
     *
     * @param contentBase note to be inserted
     * @return the newly inserted row id or Constants.DATABASE_ERROR
     */
    public long insertNote(ContentBase contentBase) {
        mDatabase.beginTransaction();
        long newlyInsertedRow = InsertHelper.insertNote(mDatabase, contentBase);
        mDatabase.setTransactionSuccessful();
        mDatabase.endTransaction();
        return newlyInsertedRow;
    }

    public void updateNoteMode(ContentBase contentBase) {
        UpdateHelper.updateNoteMode(mDatabase, contentBase);
    }

    public boolean emptyRecyclerBin() {
        return DeleteHelper.deleteNotesList(mDatabase, getAllDeletedNotes());
    }
}
```

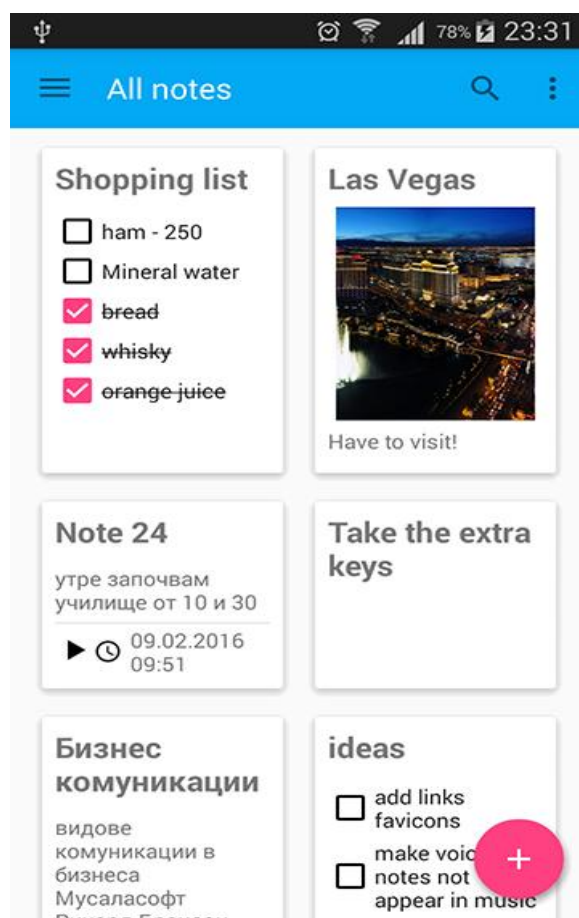
IV глава

Ръководство на потребителя

За да улесним и направим максимално ефективна работата на потребителя с приложението, нека разгледаме поотделно всеки екран и възможностите, които той предлага. За да инсталирате приложението, стартирайте „notes.apk“ на мобилно устройство с версия на Android по-голяма или равна на 4.0.3 (KitKat).

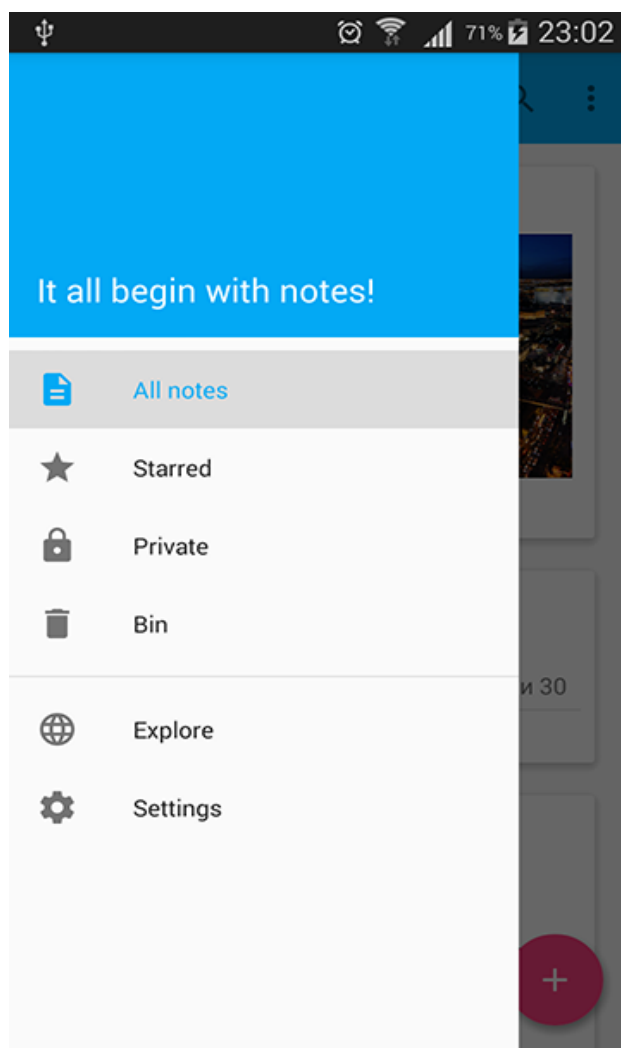
4.1. Main (главен екран)

На главния екран може да видите запазени бележки и списъци за избрания раздел, чието име е изписано на лентата с инструменти на приложението („Toolbar“). Примерът може да се види на „Фигура 4.1.1“.



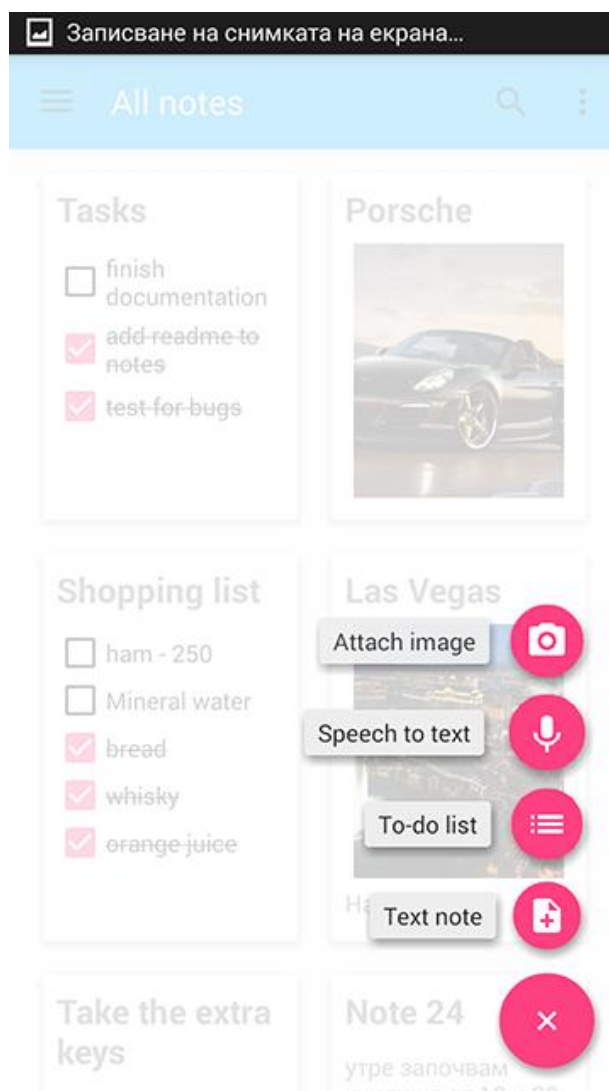
„Фигура 4.1.1“

При придърпване отляво-надясно на екрана или при натискане на хамбургер иконата (трите черти на в началото на лентата с инструменти) се отваря навигационното меню на приложението. От навигационното меню може да се сменят различните раздели, да се отваря екран за настройки или екран за обучаване на потребителя за работа с приложението („Фигура 4.1.2“).



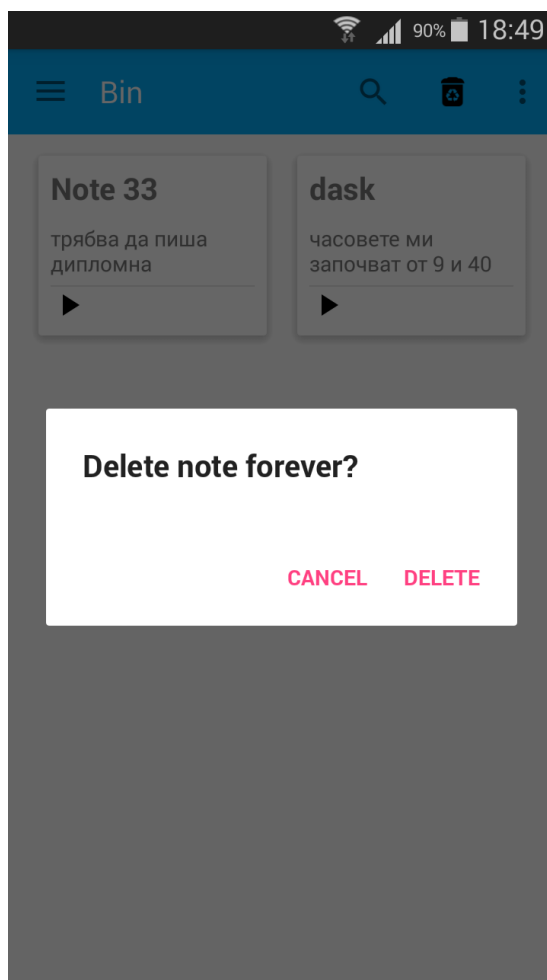
„Фигура 4.1.2“

При натискане на плаващия бутон за действия („Floating Action Button“), намиращ се в долната дясна част на екрана за потребители със стандартна „LTR“ (Left To Right) ориентация, се отваря менюто за създаване на бележки и списъци. При избиране на опция от него се активира съответният екран за съставяне на бележка или списък, попълнен с нужната информация, предоставена от избраната опция (напр. при „Speech to text“ полето „description“, тоест описанието на бележката ще бъде попълнено с транскрибираната реч). Описаното меню може да се види на „Фигура 4.1.3“.



„Фигура 4.1.3“

При жестът „swipe“ върху елемент от главния контейнер наляво или надясно (изхвърляне на бележка или списък в избраната посока) в зависимост от раздела, в който се намира, се случват различни, но подобни по смисъл действия. В „All notes“ и „Starred“ елементът, върху който е извършен „swipe“ бива изпращан в раздела „Bin“ (представляващ кошче) като най-отдолу на екрана временно се появява „snackbar“, позволяващ ни да възстановим изтрения елемент на предишната му позиция. В разделите „Private“ и „Bin“ това действие показва на потребителя диалогов прозорец с опция за перманентно изтрива на списъка или бележката („Фигура 4.1.4“).



„Фигура 4.1.4“

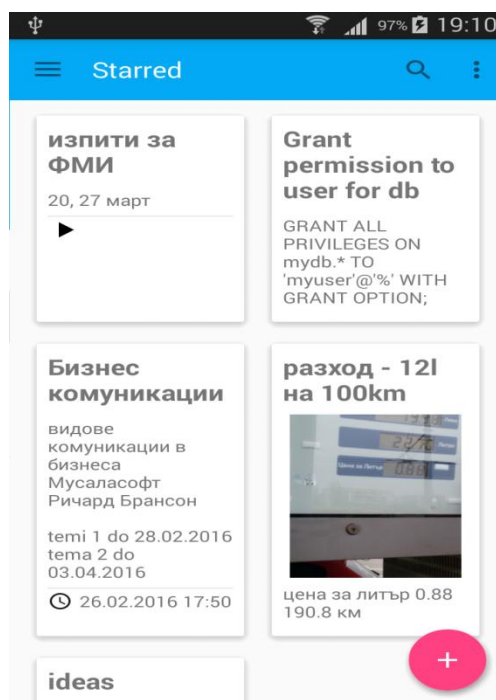
Във всички раздели освен „Bin“ (кошчето) е позволено разместването на елементи чрез влачене и пускане („drag and drop“). На лентата с инструменти на главния екран е налична търсачка, независимо от избрания от потребителя раздел. Поради различното поведение на приложението в различните раздели, нека разгледаме всеки един от тях поотделно.

Разделът „All note“:

В този раздел попадат всички бележки и списъци, които не са частни или изтрити. При добавяне на нов елемент той се появява като първи в контейнера. При стартиране на приложението разделът „All notes“ е автоматично избран.

Разделът „Starred“:

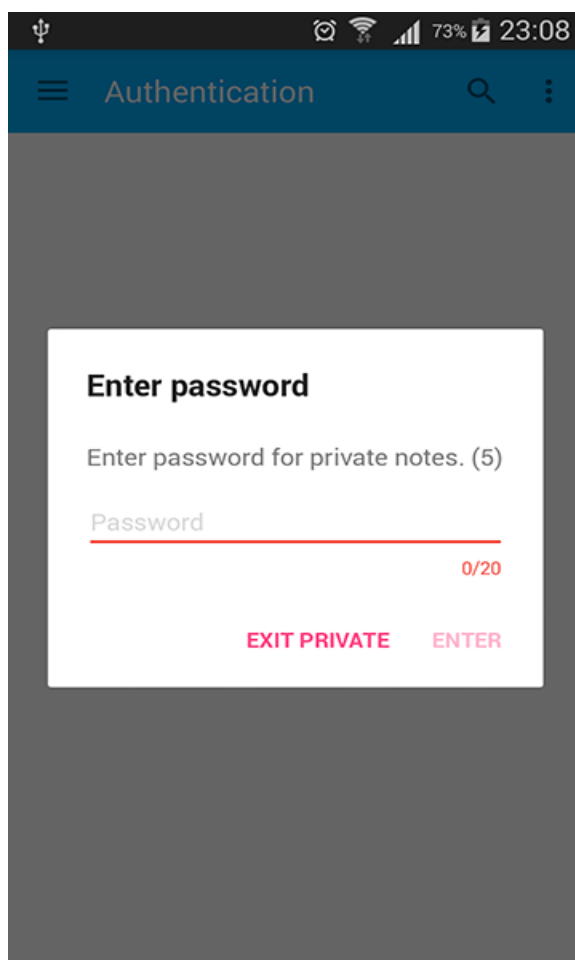
Тук са всички неизтрити бележки и списъци, на които им е дадено звезда (това може да стане от екраните за създаване или показване на бележка). „Starred“ служи за по-лесно разделение между по-маловажните и значими бележки и списъци („Фигура 4.1.5“).



„Фигура 4.1.5“

Разделът „Private“:

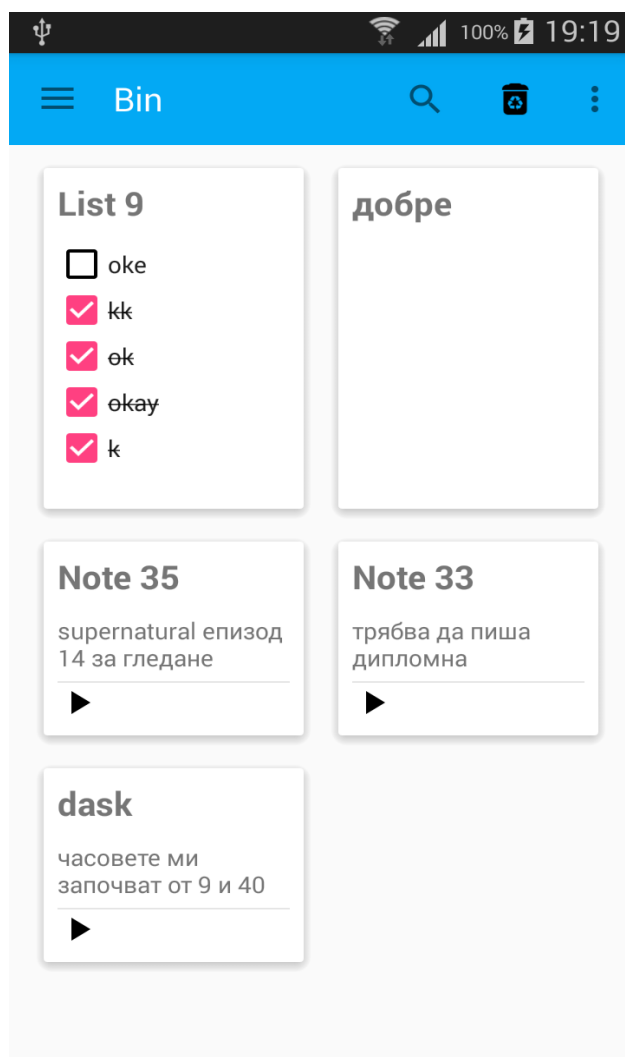
Този раздел служи за отделяне на частните от обикновените записки, той може да бъде достъпен само с парола („Фигура 4.1.6“). Ако паролата все още не е зададена, при първото негово избиране потребителят получава възможност да избере такава. При 5 опита с грешна парола всички частни елементи се изтриват и паролата се нулира. Всички бележки и списъци в раздела „Private“ са криптирани с избраната от потребителя парола. При получаване на напомняне за частен елемент заглавието на известието е „Private“ и неговото съдържание е в криптиран вид.



„Фигура 4.1.6“

Разделът „Bin“:

Този раздел представлява кошче за приложението. В него попадат всички изтрити бележки и списъци, които не са частни. Когато един елемент престрои повече от седмица в кошчето, той бива автоматично изтрит при следващото стартиране на приложението. На лентата с инструменти в този раздел освен търсачка има опция за изпразване на цялото кошче („Фигура 4.1.7“).



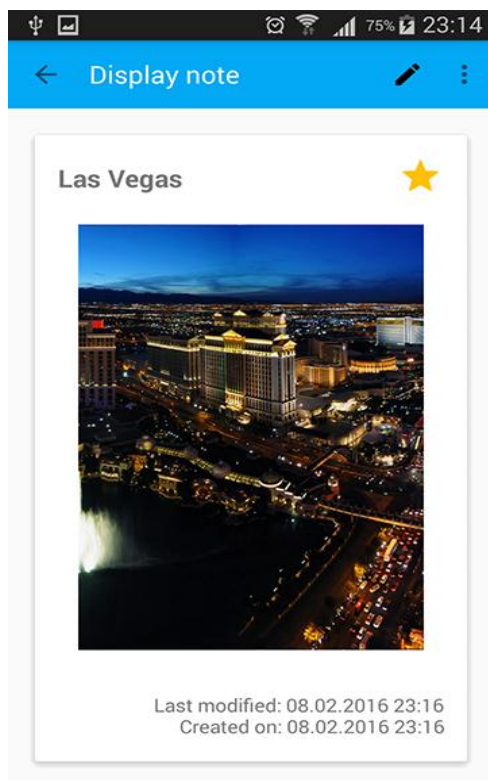
„Фигура 4.1.7“

4.2. Екрани за показване (Display activities)

При кликване върху изгледа или известие за напомняне на бележка или списък се стартира съответният екран за показване, от който може да се види детайлна информация за записката, да се отвори екран за редактиране, да се смени нейният режим (да и се даде или отнеме звезда, да се направи от обикновена частна - „заклучване“ и обратното - „отключване“) или пък да се изтрие.

4.2.1. Екран за показване на бележка (Display note)

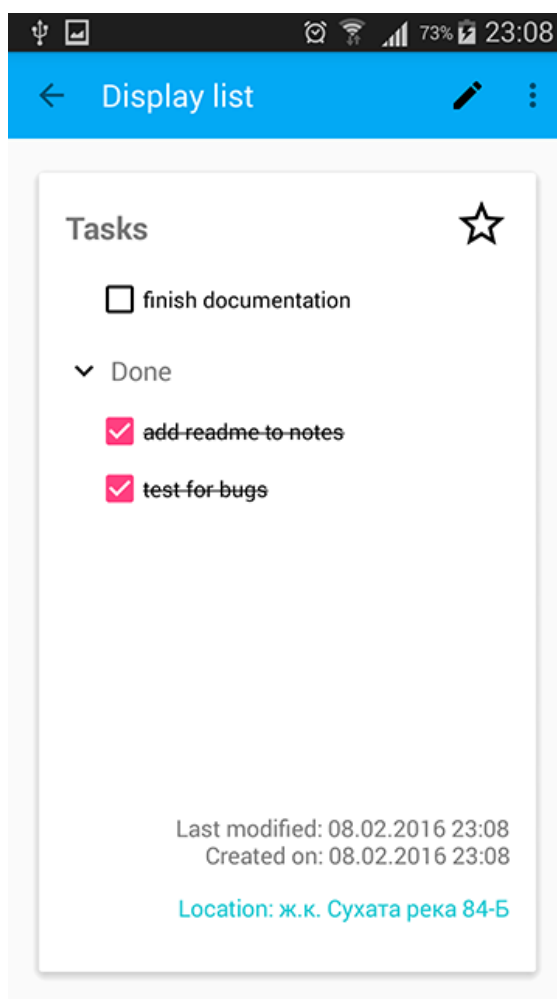
От този екран може подробно да се види съдържанието на една бележка като: цялото описание, дата на създаване, дата на последна промяна, дали има подсещане и кога, и други детайли като местоположението, на което е създадена бележката (ако потребителят е разрешил взимането на локация). Потребителят може да види всички прикрепени материали към бележката (снимков или звуков) и при натискането му да го види в галерията или прослуша. Този екран е показан на „Фигура 4.2.1“.



„Фигура 4.2.1“

4.2.2. Екран за показване на списъци (Display list)

В този екран могат да се видят всички елементи на един списък, детайли за него (дата за създаване, дата на последна промяна, локация и други). При кликване върху елемент на списък той може да бъде зачеркнат или обратното. Има бутон при натискането, на който може да се скриват или показват всички зачеркнати елементи на списъка. Екранна снимка на „Display list“ може да се види на „Фигура 4.2.2“.



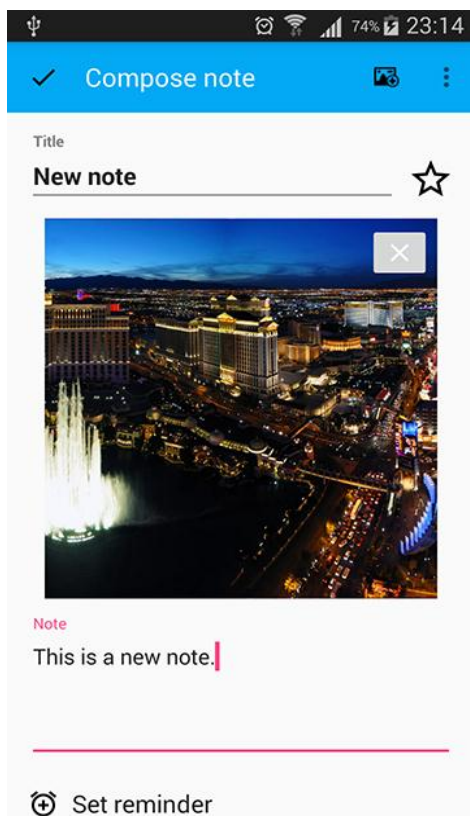
„Фигура 4.2.2“

4.3. Екрани за композиране на бележки списъци (Compose note / list)

Екраните за композиране могат да бъдат достигнати при създаване на нова записка или при редактиране на вече съществуваща такава. За да се запазят направените промени в тях трябва да се натисне чавката на лентата с инструменти. От бутона “Set reminder” може да се добави или премахне подсещане за дадената бележка или списък.

4.3.1. Екран за композиране на бележка (Compose note)

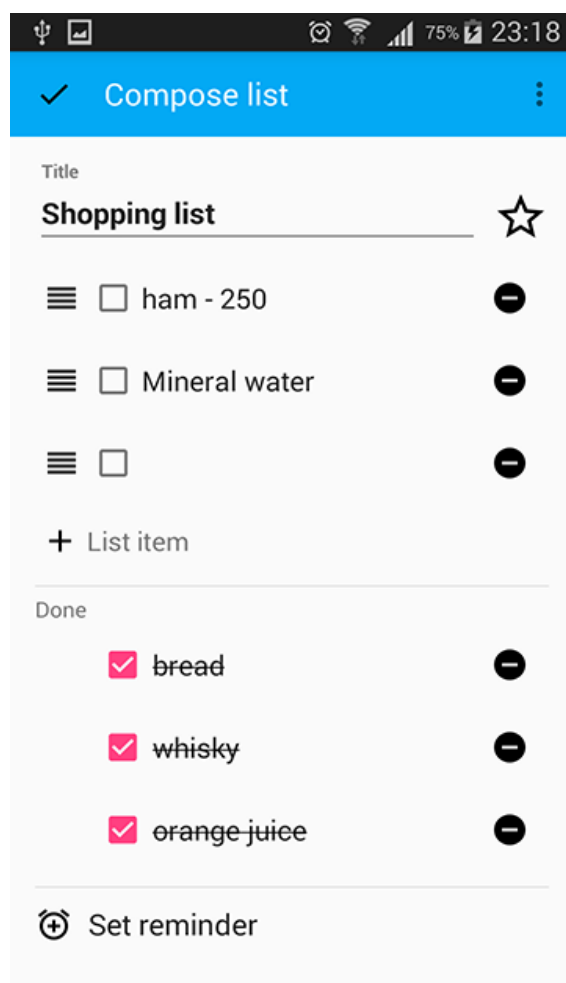
От този екран може да се променят заглавието и описанието на бележката. Да се преглежда вече прикрепен снимков материал, да се изтрива вече съществуващ или да се добавя нов такъв. При наличието на звуков материал потребителят има възможност да го преслуша и изтрие. Екранът „Compose note“ може да се види на „Фигура 4.3.1“.



„Фигура 4.3.1“

4.3.2. Екран за композиране на списък (Compose list)

В този екран може да се променя заглавието на списъка, да се редактират вече съществуващи елементи, да се добавят или премахват нови такива. Чрез влачене на бутона за разместване, намиращ се най-отляво на всеки елемент, потребителят може да пренарежда своя списък. При натискане върху чековото квадратче на елемент, той може да бъде зачеркнат или върнат в предишното си състояние. Най-вдясно на всеки елемент има бутон за премахването му от списъка, като при натискането му временно се показва „snackbar“, позволяващ операцията да бъде отменена. Екранна снимка на „Compose list“ може да се види на „Фигура 4.3.2“.



„Фигура 4.3.2“

Заклучение

Разработеното приложение покрива всички изисквания на даденото задание, като неговата функционалност е разширена и предоставя много допълнителни възможности на потребителя, които му позволяват да бъде използвано в най-различни реални ситуации. Софтуерният продукт е приложим за всякакви видове нужди от най-обикновено и просто водене на бележки до записване на частни работи като пароли, банкови сметки и т.н.

Бъдещите версии на приложението предвиждат подобрен потребителски интерфейс, включващ много и различни анимации и преходи, които да направят използването му по-приятно и интерактивно. Програмата ще бъде преведена на няколко езика включително и български, което ще позволи на повече потребители да работят с него.

Използвана литература

1. Официалния сайт на Android за разработчици
<http://developer.android.com/guide/index.html>
2. Google Material Design guide
<https://www.google.com/design/spec/material-design/introduction.html>
3. Udacity - Advanced Android App Development
<https://www.udacity.com/courses/ud855>
4. Udacity - Material Design for Android Developers
<https://www.udacity.com/course/material-design-for-android-developers--ud862>
5. The Busy Coder's Guide to Android Development, Mark L. Murphy