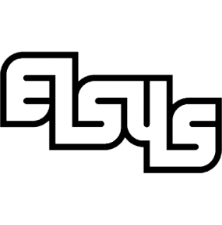
****

**TЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИСИСТЕМИ****към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

**ДИПЛОМНА РАБОТА**

**по професия код 481020 „Системен програмист“** **специалност код 4810201**  **„Системно програмиране“**Тема: Социална платформа за споделяне на лични класации   
в различни категории

Дипломант: Дипломен ръководител:  
Владислав Александров Павлов Милен Спасов

СОФИЯ

2 0 2 3

# Увод

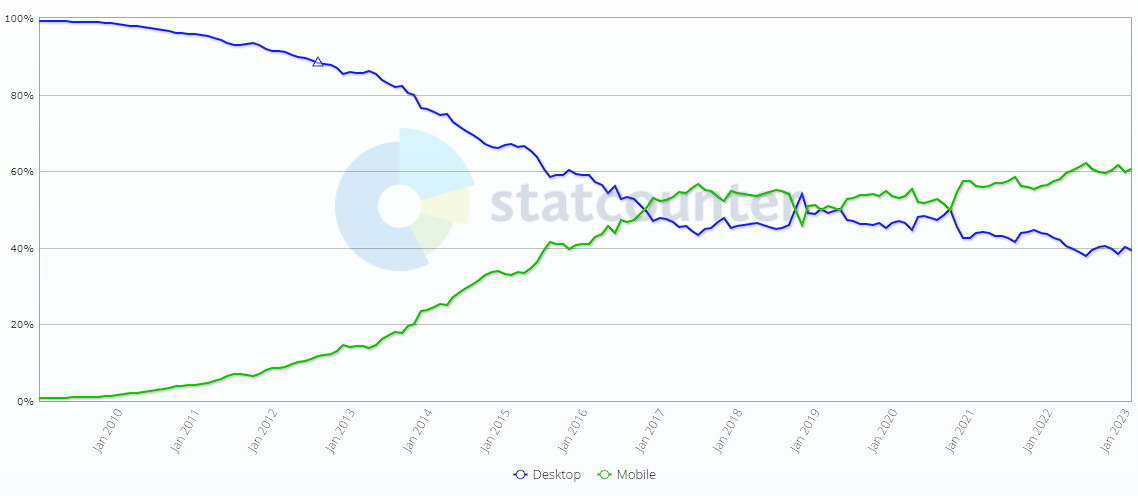
В днешната дигитална ера, социалните мрежи са съществена част от нашето ежедневие, предоставяйки начин да се свързваме с други хора, да споделяме нашите мисли, да откриваме нови интереси и др. С бързото нарастване на популярността на смартфоните и другите мобилни устройства, социалните мрежи също са изместили фокуса си към мобилни приложения, позволявайки на потребителите им да имат достъп до своите акаунти и да взаимодействат с други хора в движение.

Тази дипломна работа има за цел да се разработи приложение, социална мрежа, което позволява на потребителите да създават и споделят свои собствени колекции от подредени елементи с други потребители, както и да разглеждат колекциите им. Приложението не е предназначено да бъде широкообхватна платформа като Facebook или Instagram, а по-скоро по-малко пространство, където потребителите могат да разгледат интересите и мненията на своите приятели.

# Използвани термини и съкращения

* Frontend
* Backend
* API
* Framework - инструмент, който предоставя готови компоненти или решения, които са персонализирани с цел ускоряване на разработката
* Widget – компонент във Flutter

# ПЪРВА ГЛАВА

* 1. Възможни среди и технологии за реализация:  
     Защо точно мобилно приложение?   
     Както се вижда на фиг.1[[1]](#_Използвана_литература), в последните години мобилните устройства заемат значително по-голям дял от настолните. А за самите мобилни устройства, потребителите им прекарват много по-голям дял от времето на устройството в приложения, отколкото в браузъри[[2]](#_Използвана_литература).

1.1.1 Frontend

1.1.1.1 Разработка за специфична платформа

iOS със Swift  
разработването на приложение за iOS изисква дълбоко потапяне в екосистемата на Apple. Всеки аспект е контролиран изцяло от тях – трябва да имате акаунт на разработчик, разработва се на MacOS машина в Xcode, като за препоръчване като по-нов и поддържан език е Swift. За да се качи приложението в платвормата на Apple трябва да се плати такса и да премине обстоен преглед.

Android със Kotlin  
Kotlin е език за програмиране, разработен от JetBrains и официално се поддържа за разработка на приложения за Android. Kotlin предлага модерен синтаксис и е проектиран да бъде по-сбит и изразителен от Java - традиционния език, използван за разработване на приложения за Android.

1.1.1.2 Разработка за множество платформи чрез cross-platform технология

Flutter   
 Flutter е framework за разработка на мобилни приложения, разработен от Google. Той използва езика за програмиране Dart и предлага богат набор от предварително изградени widget-и и инструменти за създаване на красиви и интерактивни потребителски интерфейси. Flutter е известен с бързите си времена за разработка и способността си да създава изключително адаптивни и визуално привлекателни приложения.

React Native   
React Native е framework за разработка на мобилни приложения, разработен от Facebook. Той използва React JavaScript библиотеката и предлага начин за създаване на собствени мобилни приложения с помощта на познати уеб технологии.

Xamarin   
Xamarin е framework за разработка на мобилни приложения, който използва C# и позволява на разработчиците да създават собствени приложения за платформи iOS, Android и Windows. Сравнително лесно се интегрира със съществуващи кодови бази.

1.1.2 Backend

Node.js с Express:   
Node.js е популярен избор за изграждане на мащабируеми и високопроизводителни бекенд системи. Когато се комбинира с Express библиотеката, той се превръща в мощен инструмент за изграждане на RESTful API. Освен това е относително лесен за научаване и има голяма и активна общност.

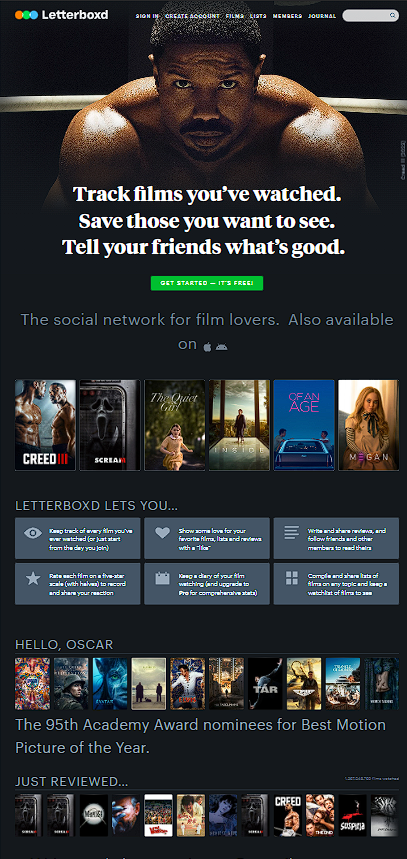
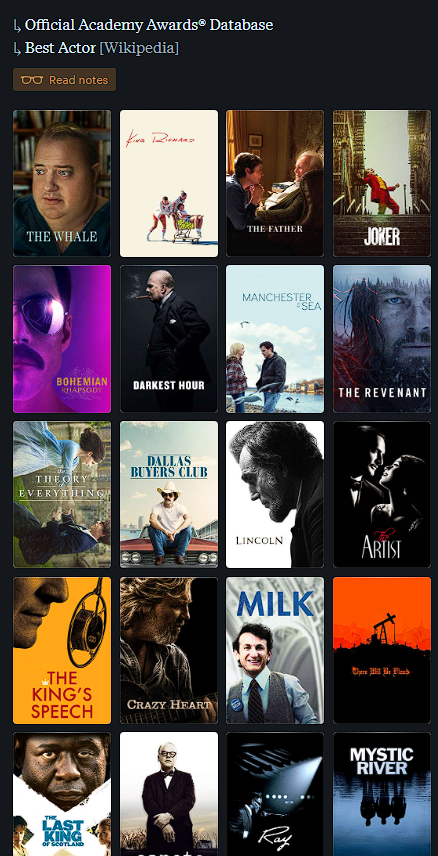
Ruby on Rails:   
Ruby on Rails е популярен избор за уеб разработка и също така е подходящ за изграждане на бекенда на мобилно приложение. Това е зряла рамка, която набляга на конвенцията над конфигурацията, което улеснява разработката с нея.

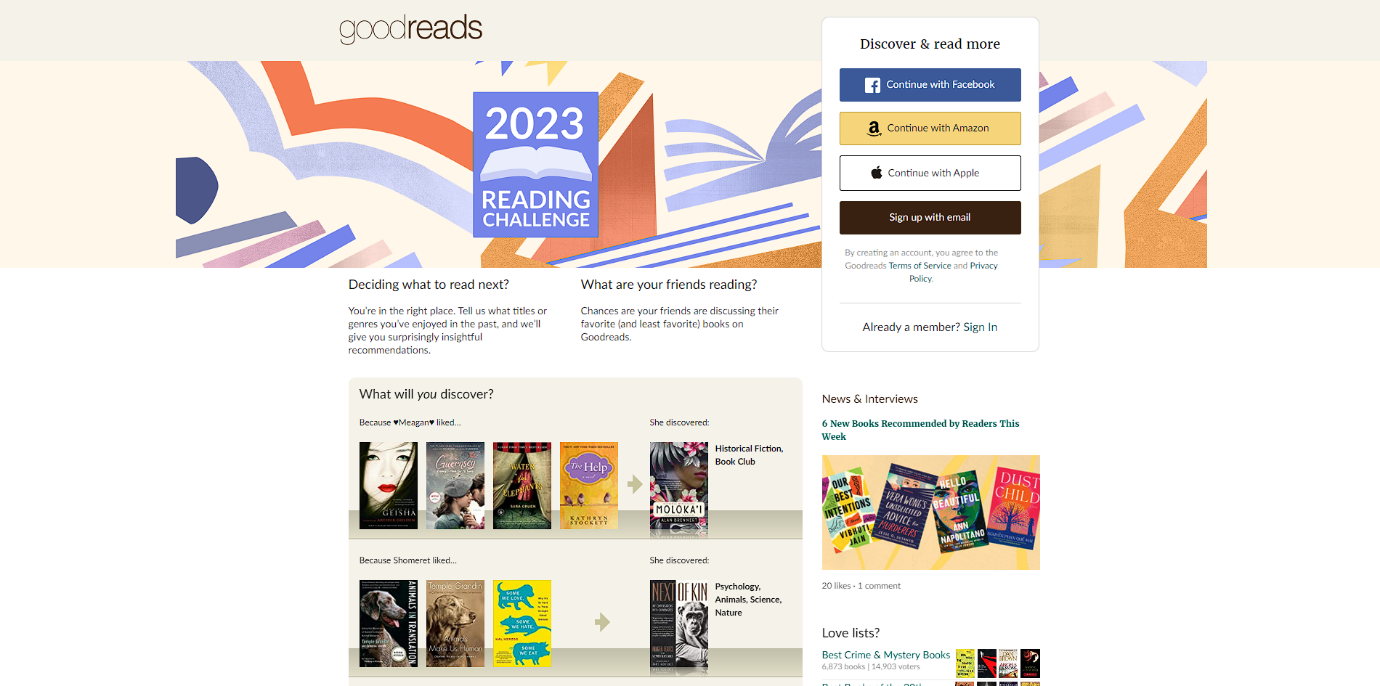
Django:   
Django е базирана на Python уеб рамка, която е много подходяща за изграждане на сложни и мащабируеми приложения. Той предоставя много вградени функции като удостоверяване, администраторски интерфейс и ORM, което го прави мощен инструмент за изграждане на бекенда на мобилно приложение.

Java с Spring Boot:   
Java е популярен език за програмиране за изграждане на приложения на корпоративно ниво и когато се комбинира с библиотеката Spring Boot, той се превръща в отличен избор за изграждане на бекенда на мобилно приложение. Spring Boot предоставя много функции наготово, което улеснява разработването.

Firebase:   
Firebase е платформа за мобилна разработка, която предоставя набор от бекенд услуги, като база данни в реално време, съхранение в облак, удостоверяване и хостинг. Той е проектиран да улесни разработчиците да създават висококачествени мобилни приложения, без да се притесняват за сървърната инфраструктура.

* 1. Конкурентни продукти

1.2.1 Letterboxd  
Letterboxd е приложение, основно за споделяне на мнения за филми, имащо огромен брой функционалности, една от които – създаване на списъци с филми.

1.2.2 Goodreads  
Goodreads е приложение, позволяващо на потребителите си да търсят в неговата база данни с книги, анотации, цитати и отзиви. Те могат също така да създават свои собствени групи от предложения за книги, анкети, анкети, блогове и дискусии.

* + 1. Listy (1) TODO

1.2.4 Listy (2) TODO  
  
Въпреки че има многобройни конкуренти в контекста на социалните мрежи, моето приложение се отличава, като предлага по-малко, лично пространство за споделяне на интереси с приятели и същевременно не е специално за конкретна тема.

# ВТОРА ГЛАВА

2.1 Изисквания

* Логин/регистрация
* Търсене на потребител
* Следване на потребител
* Създаване на класация/колекция
* Добавяне на елемент в класация

2.2 Използвани технологии

2.2.1 Frontend – Flutter  
Изборът на правилната технология за разработка на мобилното приложение е критично решение, което може да повлияе на успеха и ефективността на процеса на разработка. След като оцених различните налични опции за разработване на приложението, реших да използвам Flutter като технология за изграждане на frontend-а на приложението. Ето някои от предимствата на технологията:

Разработка за различни платформи: Flutter е мощна рамка за разработка на различни платформи, която позволява да се създават приложения за iOS и за Android посредством общ код. Това значително намалява времето за разработка.

Бърз процес на разработка: Flutter предлага функционалности като Hot Reload, което позволява на разработчиците да виждат промените в кода, рефлектирани в приложението в реално време, без да се налага да го рестартират. Това прави процеса на разработка много по-бърз и по-ефективен.

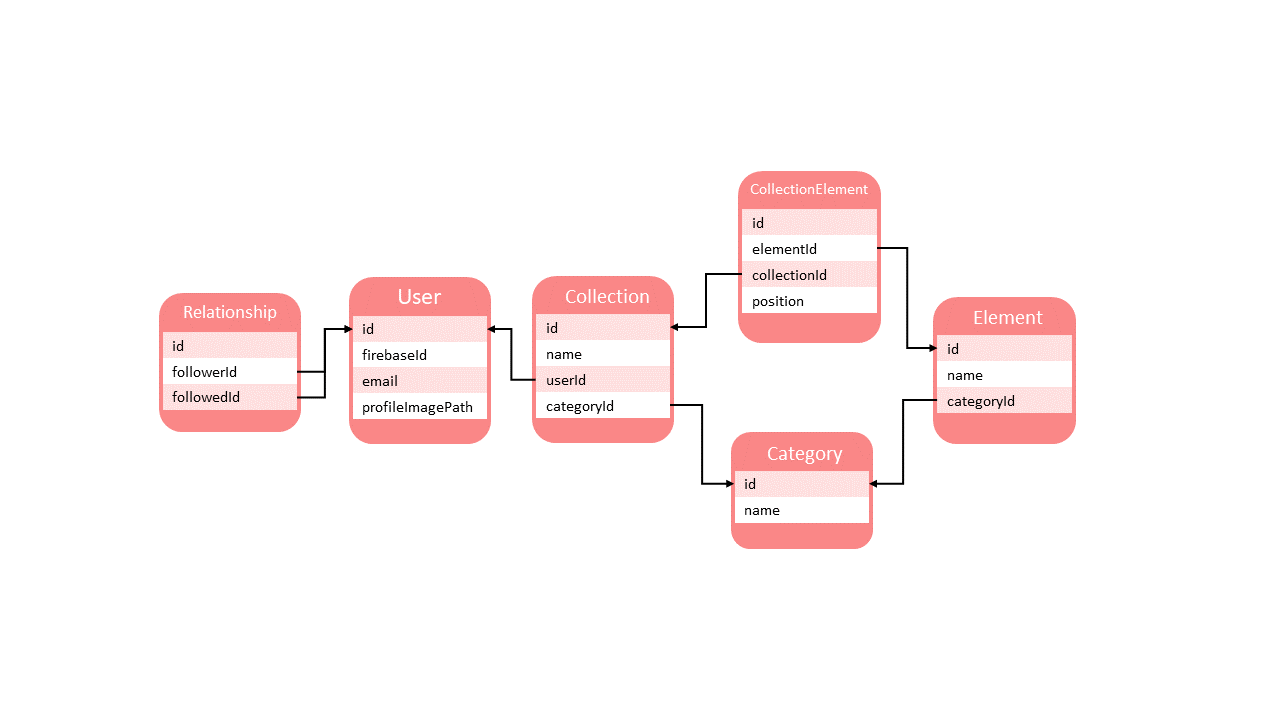
Богат потребителски интерфейс: Flutter разполага с богат и адаптивен набор от уиджети и инструменти, които позволяват на разработчиците да създават красиви и интерактивни потребителски интерфейси. Библиотеките на Material Design и Cupertino предоставят предварително изградени уиджети и елементи на потребителския интерфейс както за Android, така и за iOS платформи, като гарантират, че приложението изглежда и се чувства естествено и на двете платформи.

Силна подкрепа от общността: Flutter има голяма и нарастваща общност от разработчици и сътрудници, която осигурява достъп до широк набор от ресурси, уроци и плъгини на трети страни. Тази поддръжка от общността помага на разработчиците да учат и решават проблеми по-бързо и прави процеса на разработка по-ефективен.

* + 1. Backend – Nodejs TODO

2.2.3 Authentication – Firebase TODO

* 1. Структура на базата от данни

  
На фиг.? е показана структурата на базата данни в приложението. Тя се състои от 6 таблици:  
User – потребител, който се състои от:

Id – уникален идентификатор;  
 email – имейл на потребителя;

firebaseId – уникален идентификатор на потребителя от Firebase, който се използва при автентикация;  
 profileImagePath – пътят към профилната снимка на потребителя, съхранена локално, в сървъра

Relationship – свързваща таблица, за да се осъществи функционалността за следване на други потребители:

Id – уникален идентификатор на записа

followerId – уникален идентификатор на следващия потребител;

followedId – уникален идентификатор на потребителя, който е бил последван;

Category – съответната категория, към която принадлежи съответен елемент или колекция:

Id – уникален идентификатор;

Name – име на категорията;

Element – елемент, който да е част от колекциите:

Id – уникален идентификатор;

Name – име на елемента;

categoryId – уникален идентификатор на категорията, от която е елементът;

Collection – самата колекция, принадлежаща на потребителя

Id – уникален идентификатор;

name – име на колекцията;

userId – уникален идентификатор на потребителя, чиято е тази колекция

categoryId - уникален идентификатор на категорията, от която е колекцията;

CollectionElement – свързващата таблица за колекции и елементи

collectionId – уникален идентификатор на колекцията, в която е добавен елементът;

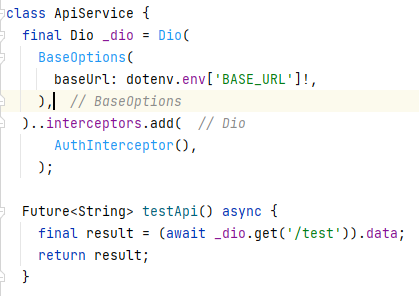
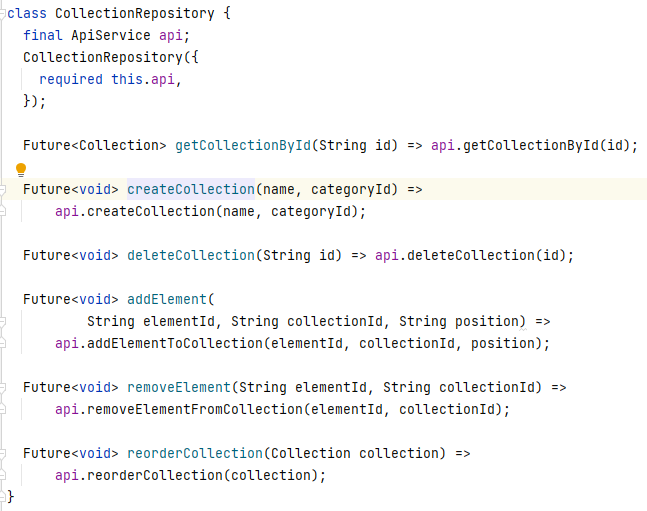
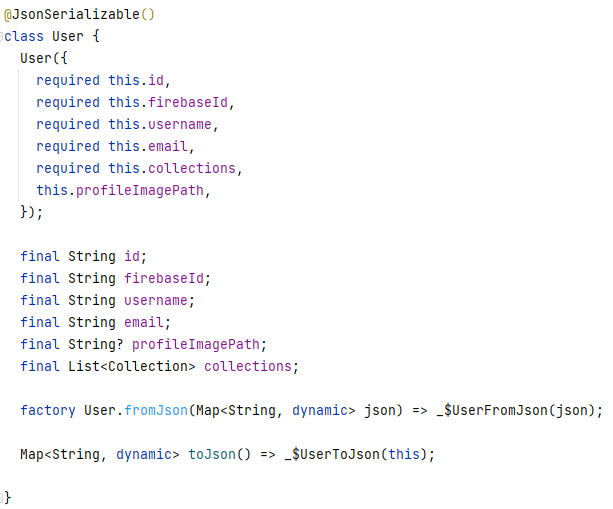
elementId – уникален идентификатор на самия елемент;

position – позицията на този елемент в колекцията

# ТРЕТА ГЛАВА

3.1 Frontend

Проектът има следната структура:

* Data – файловете нужни за достъпване на информацията от сървъра (backend-а)
  + Api – файловете за работа с външни ресурси
    - api\_service – предоставя функциите за интеракция със сървъра. На фиг.? се вижда създаването на инстанцията на Dio (библиотеката, за извършване на HTTP заявки, която използвам), като още при създаването ѝ закачам auth\_interceptor. Вижда се също така testApi() функцията, която използва тази инстанция и прави GET заявка към “/test” и връща резултата.
    - auth\_interceptor – прикачва auth token, за достъп до сървъра, към всяка заявка
  + repositories – съдържа „хранилище“ за всеки тип данни, което действа като посредник между източниците на данни за приложението и останалата му част. Целта на хранилището е да абстрахира подробностите за това как данните се извличат и съхраняват, като предоставя чист API, с който останалата част от приложението да взаимодейства. Пример на фиг.?
  + Models – “моделите”се използват за определяне на структурата и поведението на данните, които приложението манипулира.

Както се вижда на фиг.?, е обикновен клас, но за улеснение на работата със сървъра, правя превръщането на обекта от и в JSON посредством функции генерирани чрез json\_serializable библиотеката.

* Presentation – изобразяването и бизнес логиката за изобразяването на приложението
  + TODO

3.2 Backend

Проектът има следната структура:

* Един главен файл, който стартира приложението (фиг.?).
* Папката “routes”, която съдържа „маршрутизаторите“, които определят какъв код се изпълнява при достъпване на даден адрес. За пример давам маршрутизаторът за колекция, показан на фиг.?. Останалите са аналогични.  
    
  

За да се създаде маршрутизатор в Express, се използва метода Router(). Този метод връща инстанция на маршрутизатор, която след това може да се използва, за да се дефинират маршрутите на приложението за различните HTTP методи (GET, POST, PUT, DELETE и т.н.), като се използват методите на маршрутизатора .get(), .post(), .put() .delete() и т.н.

След като се създаде маршрутизатора, върху инстанцията на приложението се използва .use(), за да се прикачи към конкретен път във приложението. Това казва на Express да използва рутера за всички заявки, които съответстват на посочения път. В случая, този маршрутизатор се изпълнява при заявка към пътя „/collections“. Аналогично са направени и останалите 4 маршрутизатора.

* Папката “controllers”, съдържаща т. нар. контролери, които управляват бизнес логиката на приложениет. С други думи, контролерите определят как приложението ще отговаря на различни HTTP заявки въз основа на логиката и правилата.

На фиг.? е показан контролерът за колекция.  
Този код дефинира набор от функции, които обработват HTTP заявки, свързани с колекции в приложение Node.js/Express.js. Кодът импортира обектите Request и Response от библиотеката Express.js, както и други функции и типове от други файлове в проекта.

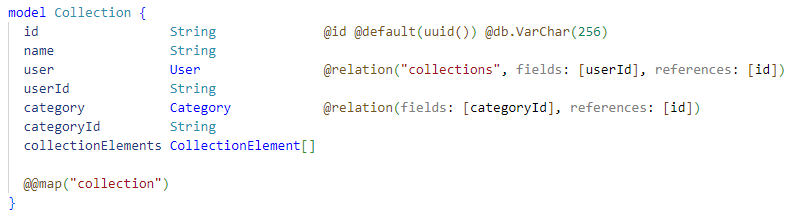
Функциите, дефинирани в контролера:

* + createCollection: Създава нова колекция чрез извикване на функцията collectionService.createCollection с данни от тялото на HTTP заявката. След това изпраща обратно HTTP отговор с новосъздадените данни за събиране.
  + getCollectionById: Намира колекция по нейния идентификатор чрез извикване на функцията collectionService.getCollectionById. След това изпраща обратно HTTP отговор с извлечените данни за колекцията.
  + addElementToCollection: Добавя елемент към колекция чрез извикване на функцията collectionService.addElementToCollection с данни от тялото на HTTP заявката. След това изпраща обратно HTTP отговор с резултата от операцията.
  + removeElementFromCollection: Тази функция премахва елемент от колекция чрез извикване на функцията collectionService.removeElementFromCollection с данни от параметрите на HTTP заявката. След това изпраща обратно HTTP отговор, показващ успех или неуспех.
  + deleteCollection: Тази функция изтрива колекция чрез извикване на функцията collectionService.deleteCollection с данни от параметрите на HTTP заявката.
  + reorderCollection: Тази функция пренарежда елементите в колекция чрез извикване на функцията collectionService.reorderCollection с данни от тялото на HTTP заявката.

Всяка функция съдържа код за обработка на грешки, който улавя грешки и връща съответния резултат.

* Папката „services“, която съдържа кода, управляващ всички случаи на интеракция с базата данни. В случая е разделен на файлове за различните ресурси.

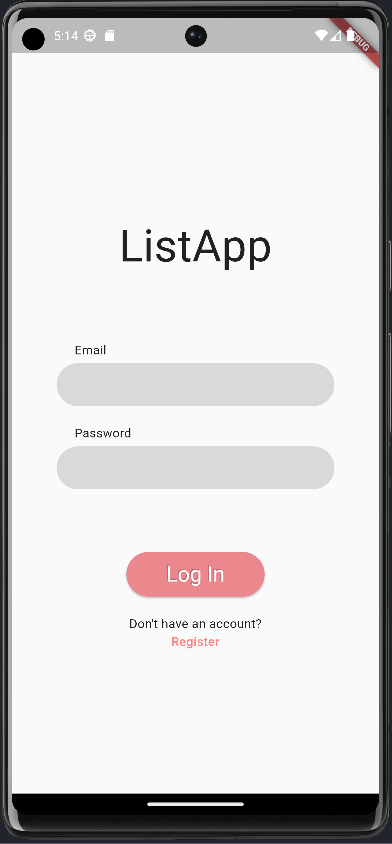
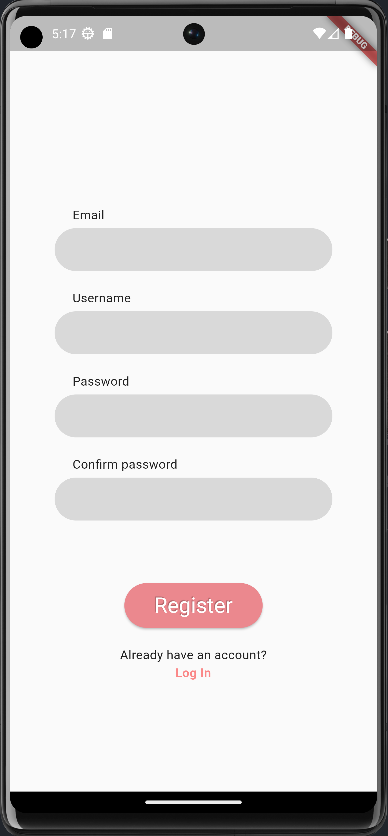
На фиг.? се вижда service файлът за колекция. Съдържа следните функции:

* + createCollection: създава нова колекция с посоченото име, принадлежаща на потребителя с посочения userId и свързана с категорията с посочения categoryId. Ако вече съществува колекция със същото име и потребителски идентификатор, тя извежда персонализирана грешка.
  + getCollectionById: извлича колекция с посочения идентификатор от базата данни. Той също така включва всички елементи, свързани с колекцията.
  + reorderCollection: актуализира позицията на елементите в колекция, за да съответства на реда, зададен от параметъра reorderedCollection. Той извършва актуализациите в транзакция, за да осигури последователност на данните.
  + addElementToCollection: добавя нов елемент с посочения elementId и позиция към колекция с посочения collectionId.
  + removeElementFromCollection: премахва всички елементи с посочения elementId от колекция с посочения collectionId.
  + deleteCollection: изтрива колекция с посочения идентификатор. Ако колекцията не съществува, тя извежда персонализирана грешка.
* Папката „prisma“, която съдържа папка с всички миграции на базата данни и schema.prisma файла, който дефинира схемата на базата данни, връзките между таблиците, типовете данни и други настройки, свързани с базата данни. На фиг.? се вижда как е дефиниран моделът на “колекция” за базата данни.
* Папка „Middleware“

# ЧЕТВЪРТА ГЛАВА РЪКОВОДСТВО НА ПОТРЕБИТЕЛЯ

4.1 Инсталация и конфигурация

* Клонирайте гитхъб хранилището
* Настройване на сървъра:
  + Свалете и инсталирайте postgresql и създайте база данни с име по избор
  + В “backend“ директорията на проекта създайте “.env” файл. Той трябва да съдържа “DATABASE\_URL” с формат postgresql://USER:PASSWORD@HOST:PORT/DATABASE  
    и “GOOGLE\_APPLICATION\_CREDENTIALS” – firebase admin sdk service account key, взет от firebase конзолата на проекта
  + Свалете и инсталирайте Node.js
  + В “backend“ директорията изпълнете командата “npm install”
* Настройване на самото приложение:
  + Инсталирайте и настройте flutter (може да следвате инструкциите на [docs.flutter.dev](https://docs.flutter.dev/get-started/install))
  + Във ”frontend/list\_app” директорията създайте .env файл.
    - Ако използвате вградения емулатор направете ”BASE\_URL” променлива със стойност „10.0.2.2:8080“
    - Ако не използвате вградения емулатор, а външно устройство, ще трябва устройството да е на същата мрежа като сървъра и да промените “BASE\_URL” в “.env” файла в директорията “/frontend/list\_app” на IP-то на сървъра.
* Стартиране на приложението
  + В “backend” директорията на проекта изпълнете командата “npm start” за да пуснете сървъра
  + Стартирайте самото приложение

4.2 Ползване на приложението:  
При стартиране на приложението се открива екранът от фиг.?. Ако нямате вече създаден акаунт натиснете “Register”, за да се прехвърлите на екрана от фиг.?, където можете да попълните полетата и да създадете нов акаунт.  
След успешен вход се открива главния екран, който е разделен на 2: профилна страница и страница за търсене. На страницата за търсене могат да се видят вече последваните или следващите вас потребители, както и да се потърси потребител и да се види техния профил (фиг.?). При отваряне на чужд профил имаме опцията за последване или отпоследване съответно според това дали вече следваме този потребител.  
На профилната страница на текущия потребител (фиг.?) се вижда бутон за смяна на профилната снимка, бутон за излизане от текущия акаунт, както и главната функционалност на приложението – списъците му и бутон за създаването на нови.  
При натискане на бутона за създаване на списък, потребителя се отвежда на друг екран, където трябва да кръсти новия списък и да избере категорията, към която принадлежи. Ако такава категория не съществува към момента, той може да я създаде (фиг.?).  
След създаване, потребителя може да отвори списъка и да добавя елементи в него, като добавянето на елемент към списъка работи подобно на избирането на категория за списъка.  
Когато има повече от един елемент в списъка потребителя може да го пренарежда, чрез продължително натискане на даден елемент и преместването му на желаната позиция.

# ЗАКЛЮЧЕНИЕ

В настоящата дипломна работа са спазени всички изисквания и е разработено приложението за създаване и споделяне на лични списъци от различни категории.

Може да се развие по много различни начини, като някои от възможностите са:

* Добавяне на екран в приложението, на който да се вижда активност на хората, последваните от нас хора (когато, добавят нов списък, добавят елемент към списък или пренаредят някой от списъците си)
* Получаване на нотификации при тази активност
* Възможност за добавяне на множество елементи към списък наведнъж
* Система за модерация на създадените от потребителите елементи, категории и колекции.

# Използвана литература

1. Статистика мобилни/настолни устройства - <https://gs.statcounter.com/platform-market-share/desktop-mobile/worldwide>
2. Статистика за времето в приложения/браузъри на мобилни устройства - <https://www.insiderintelligence.com/content/the-majority-of-americans-mobile-time-spent-takes-place-in-apps>

Съдържание

Титулна страница1

Задание2

Становище на дипломния ръководител3

Увод4

Първа глава5

Втора глава3

Въведете заглавие на главата (ниво 3)3

**Въведете заглавие на главата (ниво 1)4**

Въведете заглавие на главата (ниво 2)5

Въведете заглавие на главата (ниво 3)6