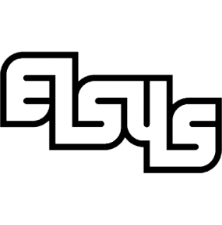
****

**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

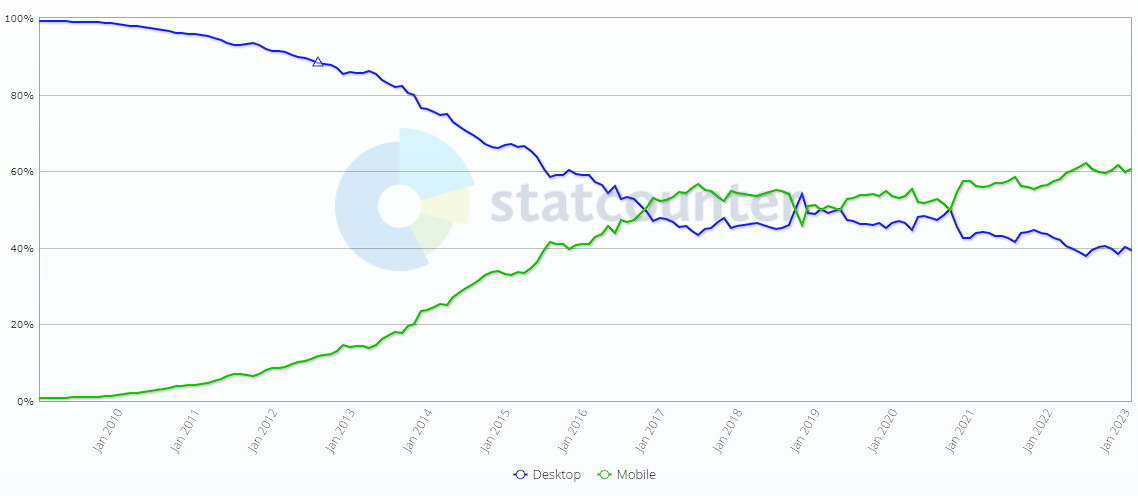
В днешната дигитална ера, социалните мрежи са съществена част от нашето ежедневие, предоставяйки начин да се свързваме с други хора, да споделяме нашите мисли, да откриваме нови интереси и др. С бързото нарастване на популярността на смартфоните и другите мобилни устройства, социалните мрежи също са 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 - насочената към потребителя част от софтуерно приложение
* Backend - частта от софтуерно приложение, която работи от страната на сървъра и отговаря за управлението на бизнес логиката, съхранението и обработката на данни.
* API (Application Programming Interface) – програмен интерфейс използван за връзка между отделни приложения
* Framework – софтуерна рамка, която предоставя готови компоненти или решения, които са персонализирани с цел ускоряване на разработката
* Widget – компонент във Flutter
* IDE (Integrated Development Environment) - софтуерно приложение, което предоставя изчерпателен набор от инструменти и функции за разработване, тестване и отстраняване на грешки в софтуер.

# ПЪРВА ГЛАВА

* 1. Възможни среди и технологии за реализация:

Фигура 1

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.

1.1.3 База данни

Firebase Realtime Database

Firebase Realtime Database е NoSQL база данни, хоствана в облака, която може да се използва като бекенд за мобилни приложения. Тя осигурява синхронизация в реално време и офлайн поддръжка, което я прави добър избор за мобилни приложения, които трябва да работят безпроблемно на различни устройства и мрежови условия. Firebase също така предоставя лесни за използване услуги за удостоверяване и хостинг, които могат да опростят процеса на разработка.

MongoDB

MongoDB е популярна NoSQL база данни, която се използва широко за разработка на уеб и мобилни приложения. Тя е известна със своята гъвкавост и мащабируемост и може да обработва големи количества данни. MongoDB също така предоставя богат набор от функции, включително, които могат да се използват за изграждане на сложни заявки.

SQLite

SQLite е лека и преносима релационна база данни. Включена е с операционни системи Android и iOS, което я прави лесен за използване, без да изисква допълнителна настройка. SQLite е идеален за малки до средни бази данни и е много подходящ за мобилни приложения, които трябва да работят офлайн.

PostgreSQL

PostgreSQL е мощна релационна база данни с отворен код. Тя предоставя богат набор от функции, включително съответствие с ACID[[6]](#_Използвана_литература), индексиране и сложни заявки, които могат да се използват за изграждане на надеждни и мащабируеми приложения.

1.1.4 IDE

Android Studio

Android Studio е официалната IDE за разработка на приложения за Android и е разработена от Google. Базиран е на платформата IntelliJ IDEA и предлага богат набор от инструменти и функции за разработване на приложения за Android. Android Studio включва Android емулатор за тестване на приложения на различни виртуални устройства, както и редактор на оформление за проектиране на потребителски интерфейси. Android Studio също поддържа Kotlin като език за програмиране.

Xcode

Xcode е официалната IDE за разработка на приложения за iOS и е разработена от Apple. Той включва широк набор от инструменти и функции за разработване на приложения за iOS, macOS, watchOS и tvOS. Xcode включва визуален редактор за проектиране на потребителски интерфейси, както и симулатор за тестване на приложения на виртуални устройства. Xcode поддържа Swift като език за програмиране.

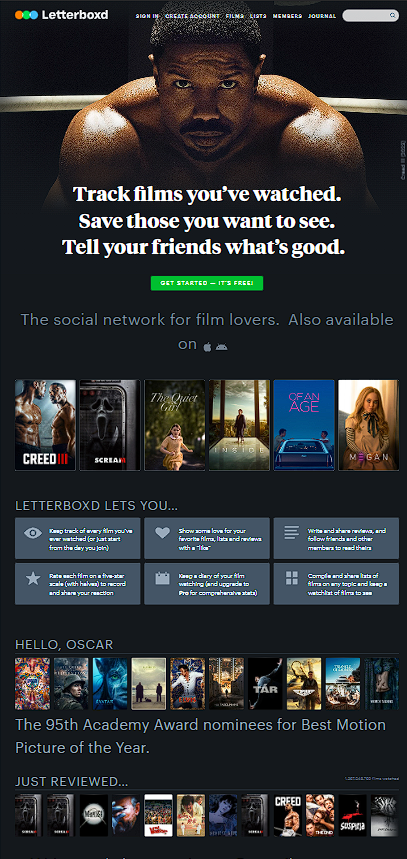
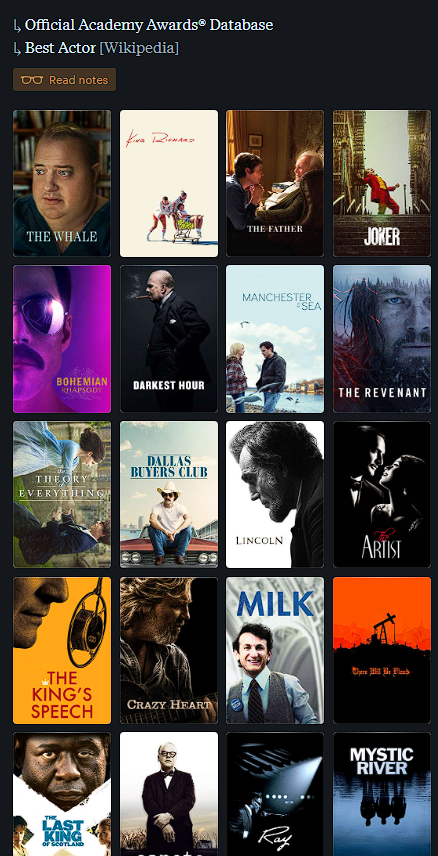
Visual Studio

Visual Studio е IDE, разработена от Microsoft и се използва широко за разработване на различни софтуерни приложения, включително мобилни приложения. Visual Studio включва набор от инструменти и функции за разработване на мобилни приложения за платформи iOS, Android и Windows. Той поддържа различни езици за програмиране, включително C#, който се използва за разработване на приложения на Xamarin.

Visual Studio Code

Visual Studio Code е лека и гъвкава IDE, разработена от Microsoft. Тя предлага набор от инструменти и функции за разработване на мобилни приложения за платформи Android и iOS, както и междуплатформени софтуерни рамки като Flutter и React Native. Visual Studio Code поддържа различни езици за програмиране и предлага богат набор от разширения за персонализиране на средата за разработка.

* 1. Конкурентни продукти

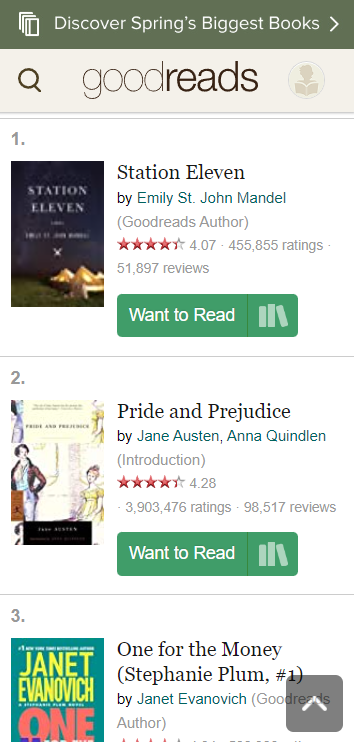
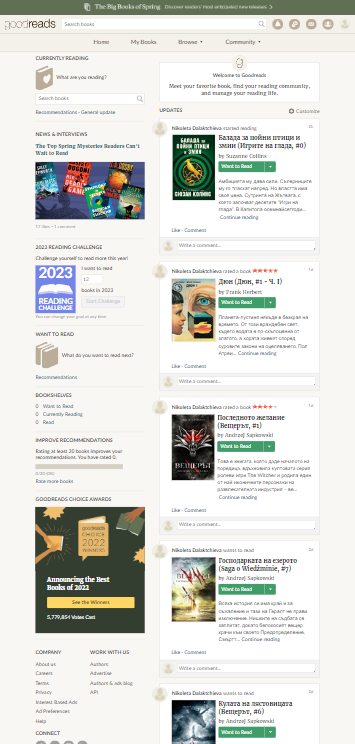
Letterboxd

Фигура 2.2

Фигура 2.1

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

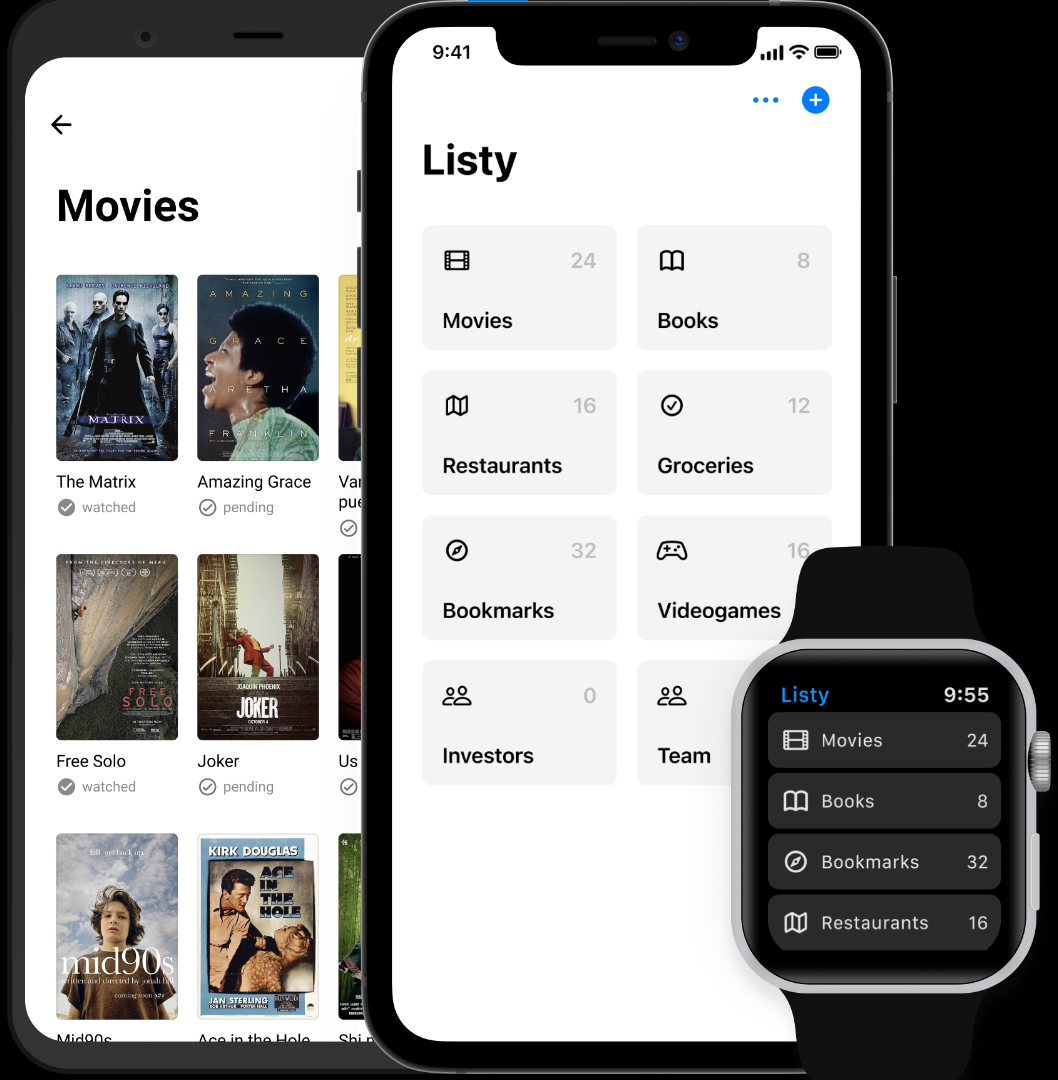
Goodreads

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.

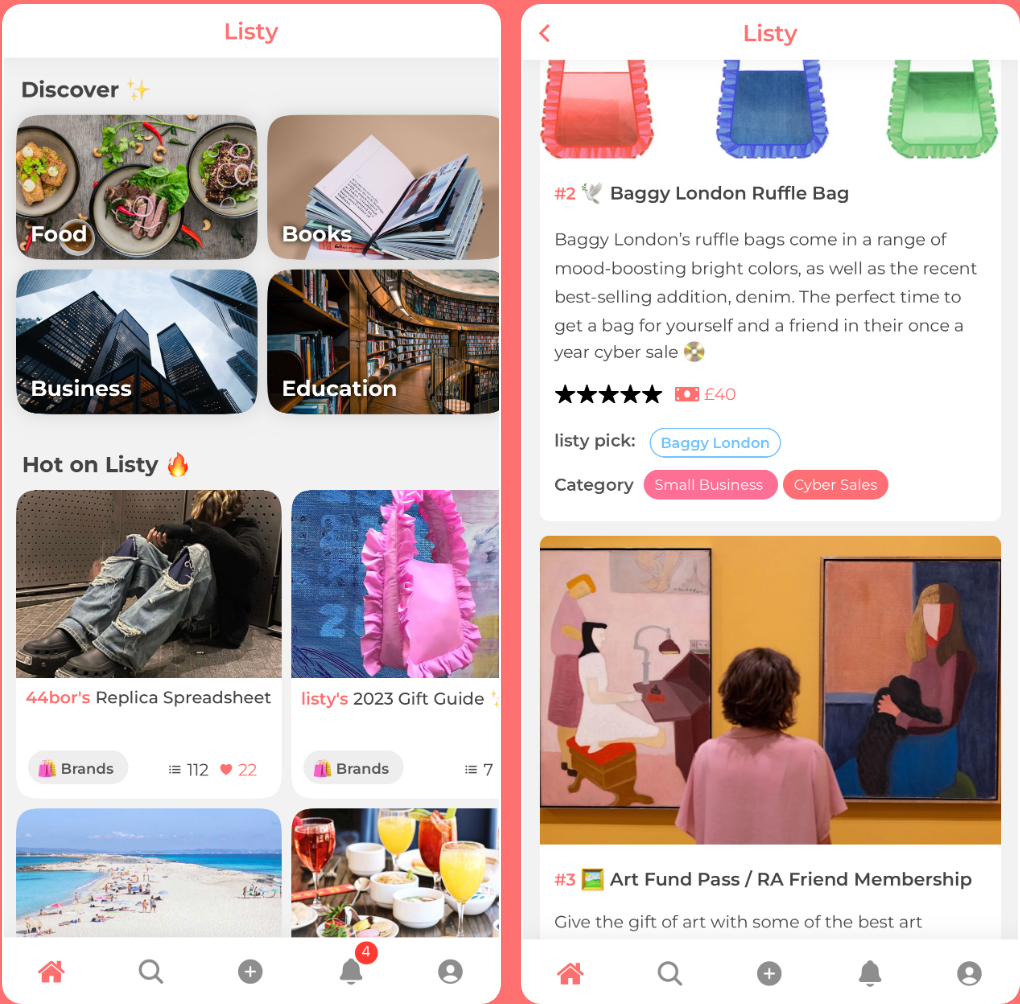
Фигура 3.2

Фигура 3.1

Listy – мобилно приложение за създаване на списъци, но на този етап без виждане на чужди списъци и само от 8 конкретни категории.



Фигура 4

Listy (2) – социална мрежа, с голям акцент върху създаването на списъци. Съдържа много голям брой допълнителни функционалности.

Фигура 5

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  
Приложението използва стандартните REST[[3]](#_Използвана_литература) принципи с клиент и сървър.

2.2.1 Frontend – Flutter[[4]](#_Използвана_литература)

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 софтуерна 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.

2.2.1.1 State management – Riverpod

Има множество библиотеки за менажиране на състояние във Flutter, като двете най-използвани са Provider и Riverpod и са направени от един и същи човек - Remi Rousselet. По негови думи Riverpod поправя много от недостатъците на Provider, като може би най-голямото му предимство е, че е изцяло compile-safe.

2.2.2 Backend – Nodejs[[5]](#_Използвана_литература) Express на Typescript

За бизнес логиката на сървъра използвам Express библиотеката за Nodejs на Typescript.

Едно от основните предимства на използването на Node.js с Express е неговата лекота на използване. Общността Node.js предоставя изобилие от документация, уроци и библиотеки, които могат да помогнат на разработчиците да се запознаят бързо с технологията. Интуитивният API на платформата я прави лесна за научаване и използване както от начинаещи, така и от опитни разработчици.

Друго предимство на използването на Node.js с Express е неговата богата екосистема. Екосистемата Node.js има огромна колекция от модули и библиотеки с отворен код, които могат лесно да бъдат интегрирани в приложения, което позволява на разработчиците да изграждат своите приложения по-бързо и по-ефективно.

Още едно голямо предимство е, че Firebase лесно се интегрира с технологията. Тя може да се използва с обикновен JavaScript или с TypeScript.

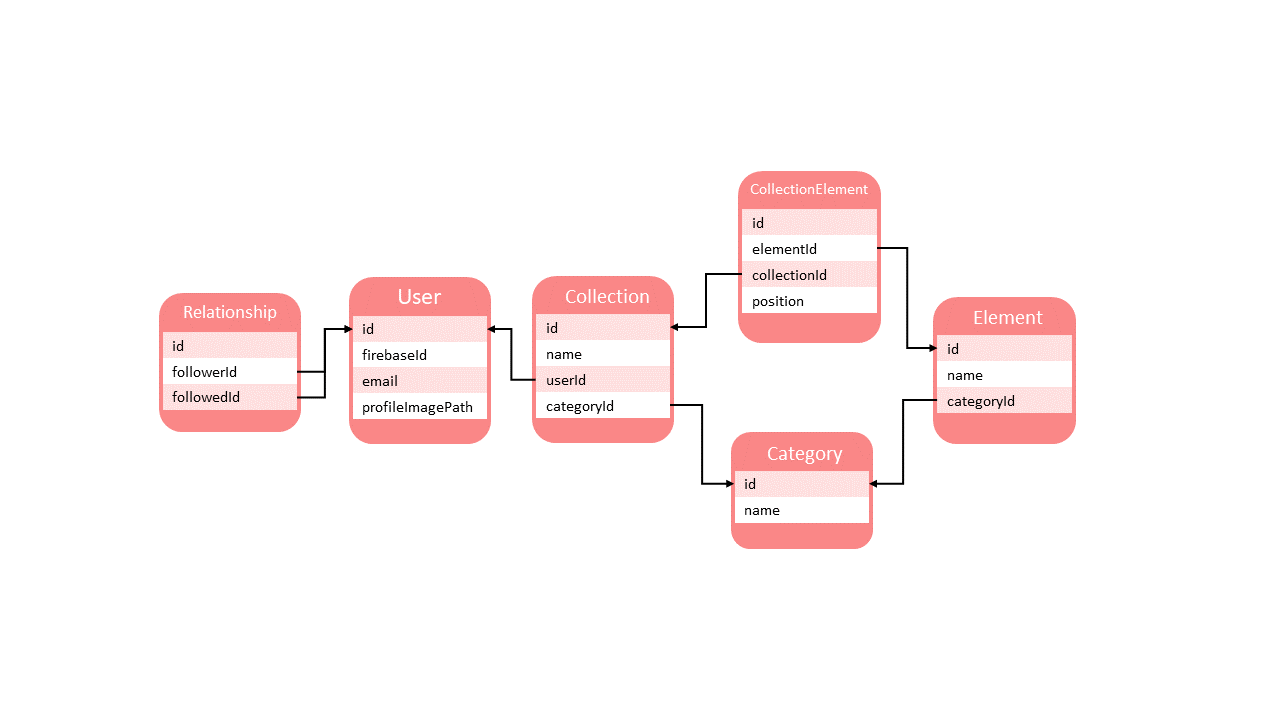
Ето някои от главните ползи от използването на express със typescript:

* Статично типизиране: TypeScript добавя статично типизиране към JavaScript, което помага за улавяне на грешки по време на компилиране, а не по време на изпълнение. Това улеснява поддържането на широкомащабни приложения.
* Подобрена продуктивност на разработчиците: Функциите за извеждане на типове и завършване на код на TypeScript помагат на разработчиците да пишат код по-бързо и с по-малко грешки.
* Подобрено качество на кода: Използването на TypeScript може да доведе до подобрено качество на кода, тъй като насърчава разработчиците да пишат по-стабилен и поддържаем код.

2.2.3 Authentication – Firebase

Firebase Authentication е услуга, предоставена от Google Firebase, която позволява на разработчиците лесно да добавят потребителско удостоверяване към своите приложения.  
Използването на Firebase Authentication за удостоверяване може да спести време и усилия на разработчиците при внедряването на сигурна и надеждна система за удостоверяване. Чрез предоставяне на множество методи за удостоверяване, управление на потребителите и интегриране с други услуги на Firebase, разработчиците могат лесно да създадат пълен бекенд за своето приложение. Услугата е лесна за използване и има щедър безплатен план, което я прави привлекателна опция за разработчици с всички нива на умения.

2.2.4 Структура на базата от данни

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

Фигура 6

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;

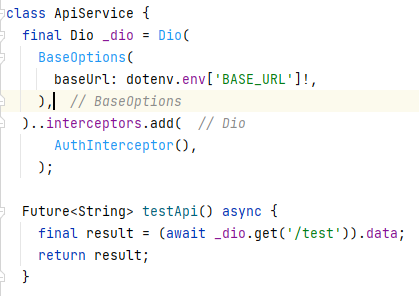
elementId – уникален идентификатор на самия елемент;

position – позицията на този елемент в колекцията

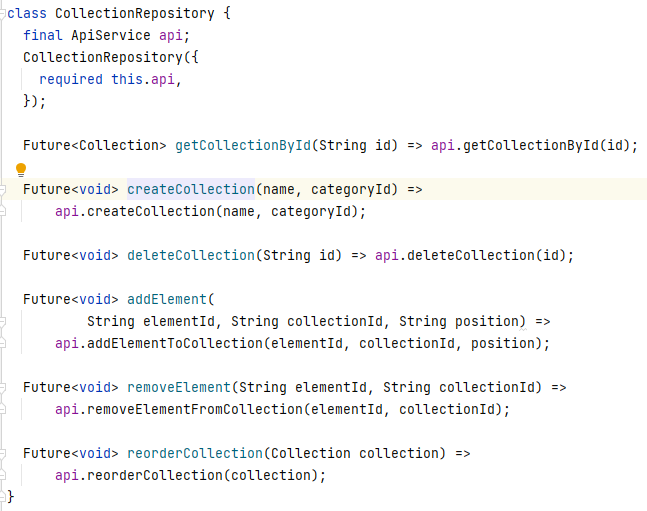
# ТРЕТА ГЛАВА

3.1 Frontend

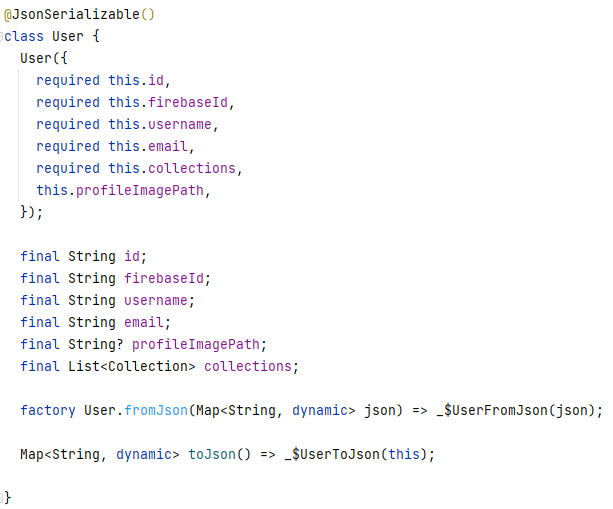
Проектът има следната структура:

* 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.7 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.

Фигура 7

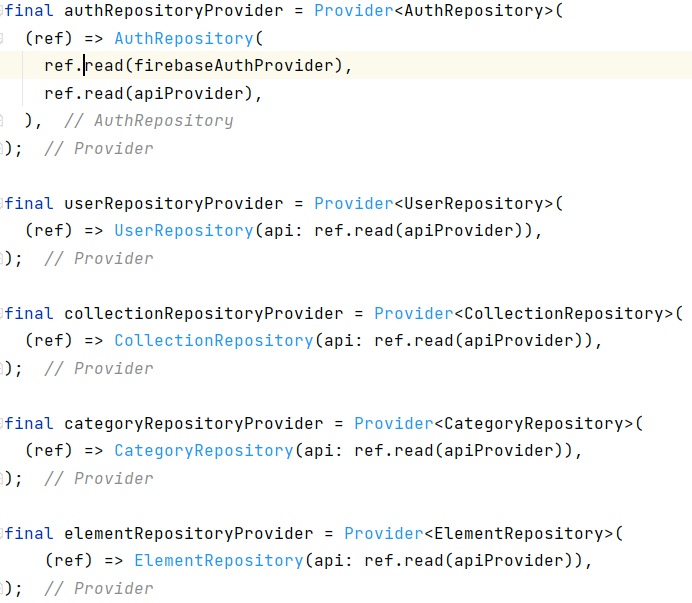
* + - auth\_interceptor – прикачва auth token, за достъп до сървъра, към всяка заявка
  + repositories – съдържа „хранилище“ за всеки тип данни, което действа като посредник между източниците на данни за приложението и останалата му част. Целта на хранилището е да абстрахира подробностите за това как данните се извличат и съхраняват, като предоставя чист API, с който останалата част от приложението да взаимодейства. На фиг. 8 се вижда хранилището за колекция

Фигура 8

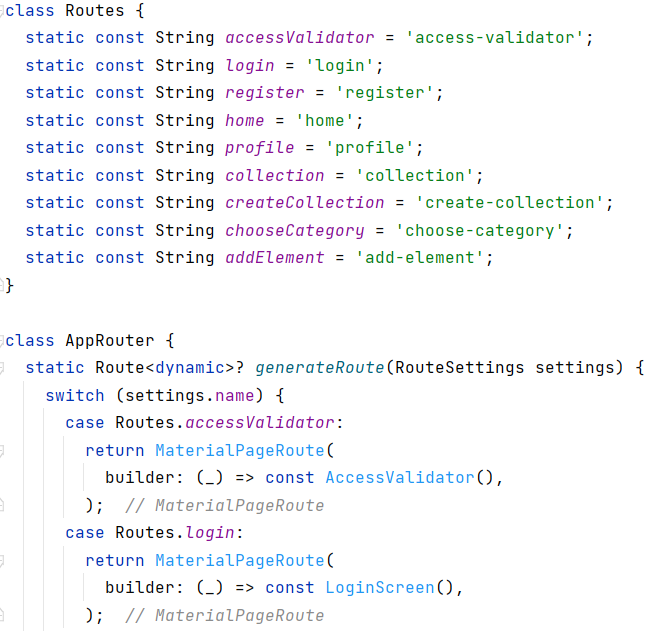
* + Models – “моделите”се използват за определяне на структурата и поведението на данните, които приложението манипулира.

Фигура 9

Както се вижда на фиг. 9, моделът е обикновен клас, но за улеснение на работата със сървъра, превръщането на обекта от и в JSON се прави посредством функции генерирани чрез json\_serializable библиотеката.

* Presentation – изобразяването и бизнес логиката за изобразяването на приложението
  + common\_providers – съдържа provider-и общи за повече от един екран. На фиг.10 се виждат provider-ите за repository класовете.

Фигура 10

* + Util – съдържа полезни класове за цялото приложение. Пример: route\_manager – навигацията за цялото приложение (фиг. 11)

Фигура 11

* + Views
    - Custom\_widgets – персонализирани widget-и общи за повече от един екран
    - Screens – различните екрани на приложението.

Всяка Директория за екран има:

* Директория widgets, съдържаща widget-ите специфични за този екран
* providers – provider-ите специфични за този екран
* controller – бизнез логиката специфична за този екран
* screen – файлът на самия екран   
  На фиг. 12 се вижда част от кода на логин екрана.

Фигура 12

3.2 Backend

Проектът има следната структура:

* Един главен файл, от който стартира приложението, инициализира Express и му казва да използва библиотеки функции на съответните адреси (фиг. 13).

Фигура 13

* Директорията “routes”, която съдържа „маршрутизаторите“, които определят какъв код се изпълнява при достъпване на даден адрес. За пример давам маршрутизаторът за колекция, показан на фиг. 14. Останалите са аналогични.



Фигура 14

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.

* Директорията “controllers”, съдържаща т. нар. контролери, които управляват бизнес логиката на приложението. С други думи, контролерите определят как приложението ще отговаря на различни HTTP заявки.

Фигура 15

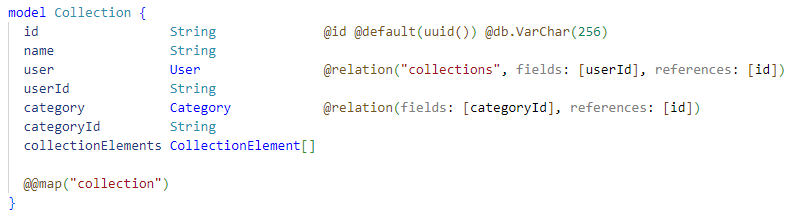
На фиг. 15 е показана частта от контролерът на колекция за създаването на нова колекция.

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

Фигура 16

На фиг. 16 се вижда частта от service файлът на колекция за създаването ѝ. Първо се прави проверка дали не съществува такава колекция за този потребител и връща грешка, ако съществува. Ако не върне грешка създава самата колекция, използвайки вградената create функция от Prisma. Тя приема обект със данните за новата колекция, като посредством “connect” атрибута се свързва със съответния потребител и категория.  
Този контролер 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.
* Директорията „prisma“, която съдържа директория 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. 17 се вижда как е дефиниран моделът на “колекция” за базата данни.

Фигура 17

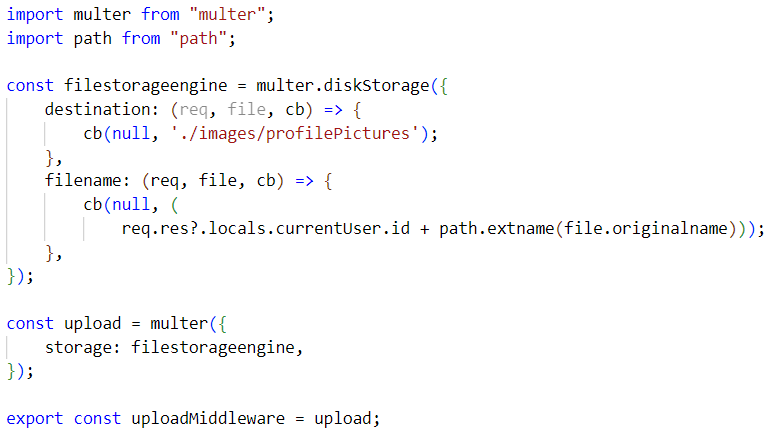
* Директория „Middleware“

Съдържа „посреднически“ функции, които могат да се използват в различни части на приложението. „Посредническите функции“ са функции, които могат да се използват за модифициране на входящи заявки или изходящи отговори, преди да бъдат обработени от функциите за обработка на маршрути.

В случая съдържа 2 файла:

* + auth\_middleware.ts, където се прави проверката за автентикационен жетон (фиг. 18)

Фигура 18

* + upload.ts, където се конфигурира библиотеката за качване на файлове – multer (фиг. 19).

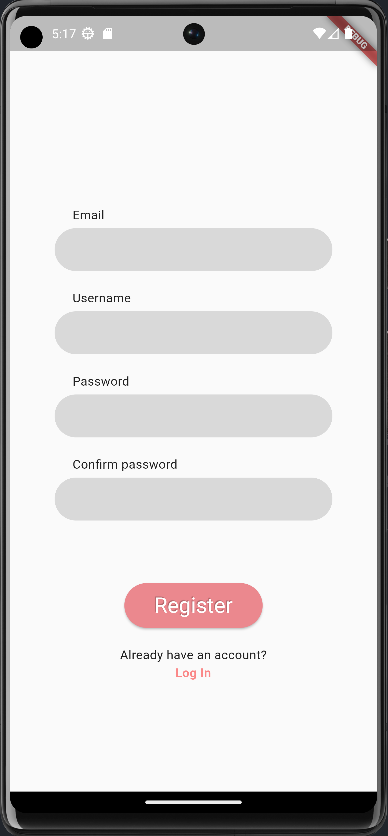
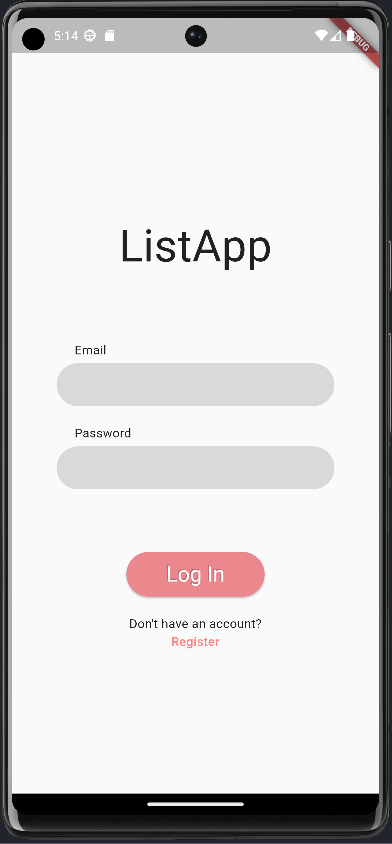
Фигура 19

# ЧЕТВЪРТА ГЛАВА РЪКОВОДСТВО НА ПОТРЕБИТЕЛЯ

4.1 Инсталация и конфигурация

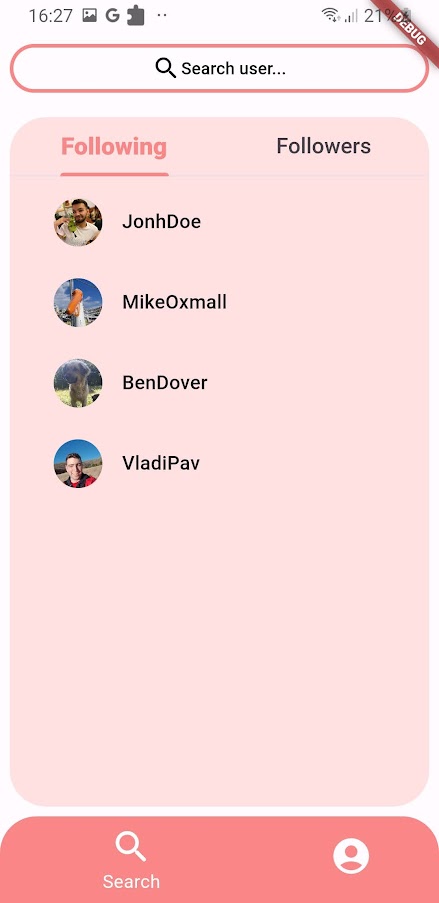
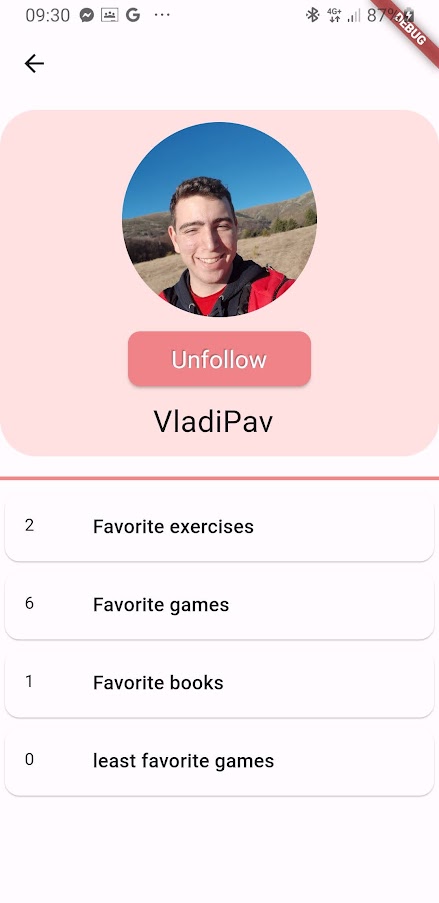
* Клонирайте гитхъб хранилището
* Настройване на сървъра:
  + Свалете и инсталирайте postgresql и създайте база данни с име по избор
  + В “backend“ директорията на проекта създайте “.env” файл. Той трябва да съдържа “DATABASE\_URL” с формат postgresql://USER:PASSWORD@HOST:PORT/DATABASE  
    и “GOOGLE\_APPLICATION\_CREDENTIALS” – 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. 20. Ако нямате вече създаден акаунт натиснете “Register”, за да се прехвърлите на екрана от фиг. 21, където можете да попълните полетата и да създадете нов акаунт.

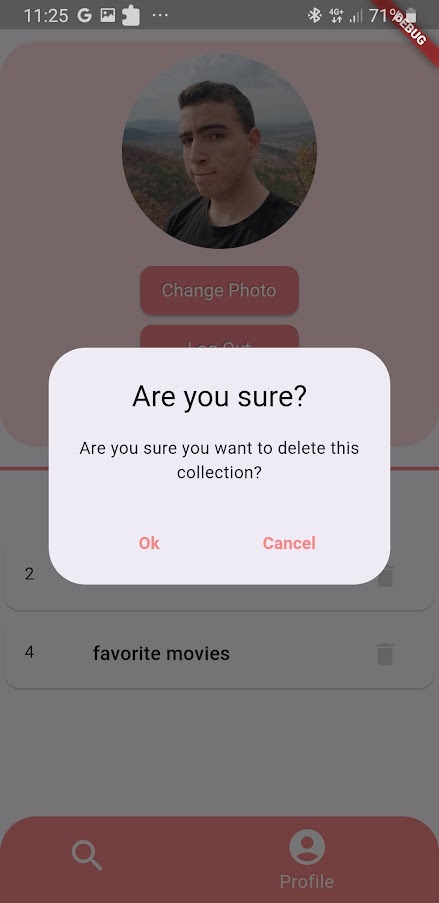
Фигура 20

Фигура 21

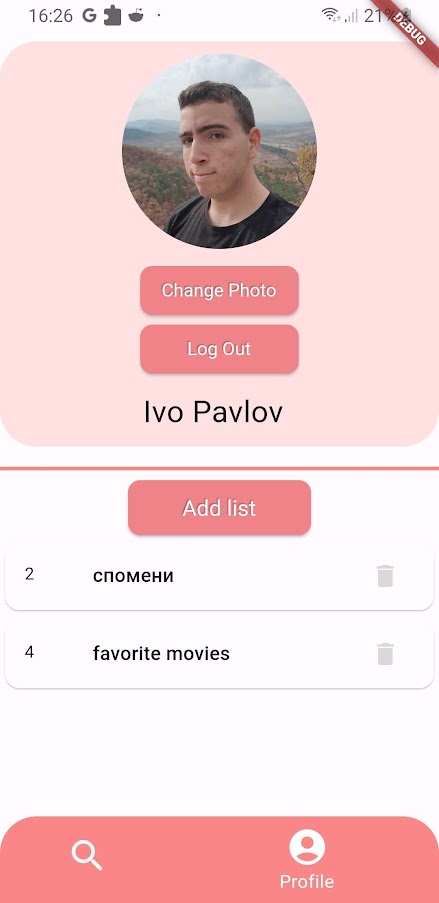
След успешен вход се открива главния екран, който е разделен на 2: профилна страница и страница за търсене. На страницата за търсене могат да се видят вече последваните или следващите вас потребители (фиг. 22), както и да се потърси потребител и да се види техния профил (фиг. 23). При отваряне на чужд профил имаме опцията за последване или отпоследване съответно според това дали вече следваме този потребител.

Фигура 23

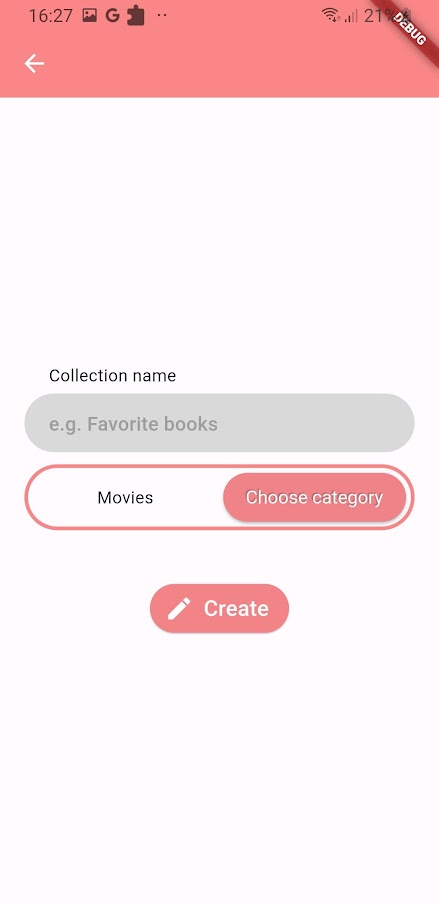
Фигура 22

На профилната страница на текущия потребител (фиг. 24) се вижда бутон за смяна на профилната снимка, бутон за излизане от текущия акаунт, както и главната функционалност на приложението – списъците му, като всеки има бутон за изтриване (фиг. 25) и бутон за създаването на нови.

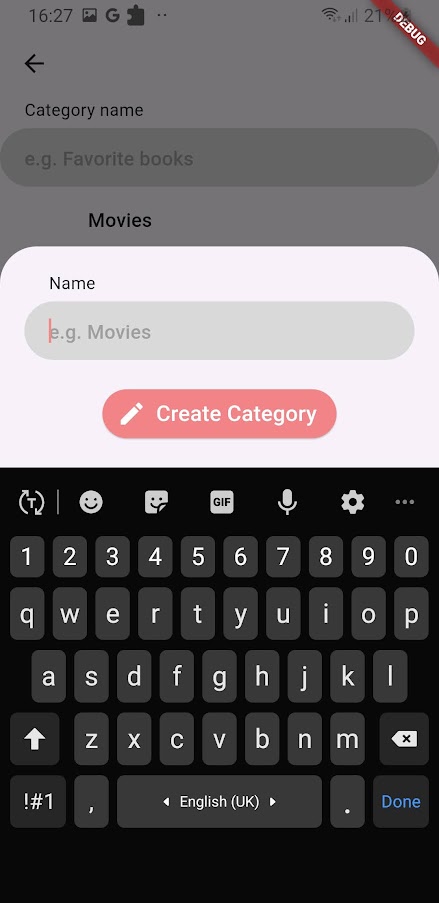
Фигура 25



Фигура 24

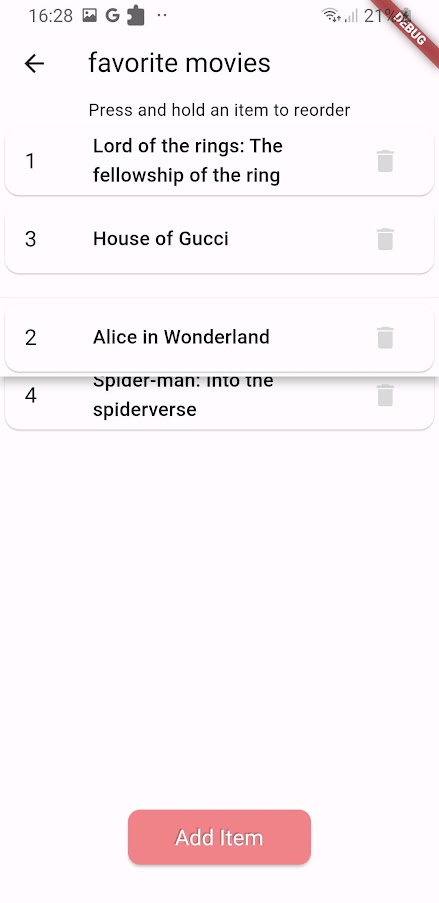
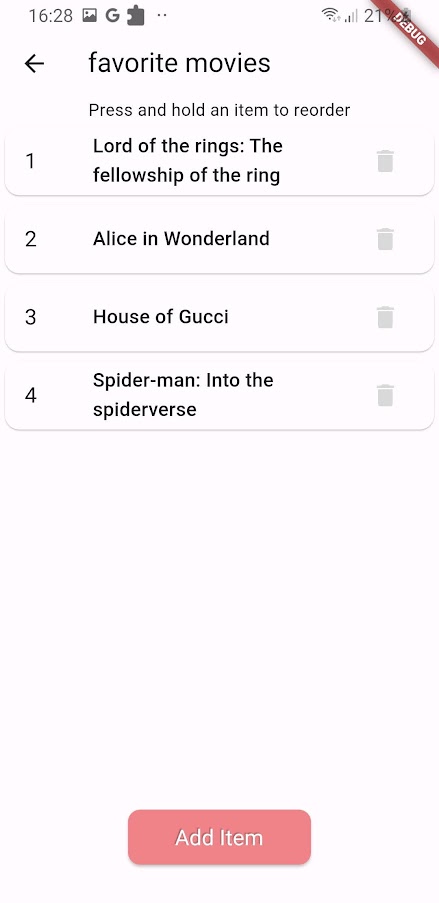
При натискане на бутона за създаване на списък, потребителя се отвежда на друг екран, където трябва да кръсти новия списък и да избере категорията, към която принадлежи (фиг. 26). Ако такава категория не съществува към момента, той може да я създаде (фиг.27).

Фигура 26



Фигура 27

След създаване, потребителя може да отвори списъка (фиг. 28) 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 (фиг. 29).

Фигура 28

Фигура 29

# 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>
3. REST - <https://www.codecademy.com/article/what-is-rest>
4. Flutter - [https://flutter.dev](https://flutter.dev/)
5. Nodejs - <https://nodejs.org/en/>
6. ACID - [https://www.ibm.com/docs/en/cics-ts/5.4?topic=processing-acid-properties-transactions](https://www.ibm.com/docs/en/cics-ts/5.4?topic=processing-acid-properties-transactions%20)

# **Съдържание**

[Титулна страница 1](file:///E:\School\diplomnaRabota\diplomna-rabota\Документация.docx#_Toc129859643)

[Увод 4](#_Toc129859648)

[Използвани термини и съкращения 5](#_Toc129859649)

[ПЪРВА ГЛАВА 6](#_Toc129859656)

[1.1 Възможни среди и технологии за реализация: 6](#_Toc129859657)

[1.1.1 Frontend 6](#_Toc129859661)

[1.1.2 Backend 8](#_Toc129859674)

[1.1.3 База данни 9](#_Toc129859685)

[1.1.4 IDE 10](#_Toc129859694)

[1.2 Конкурентни продукти 12](#_Toc129859703)

[ВТОРА ГЛАВА 16](#_Toc129859717)

[2.1 Изисквания 16](#_Toc129859718)

[2.2 Използвани технологии 16](#_Toc129859724)

[2.2.1 Frontend – Flutter 16](#_Toc129859725)

[2.2.2 Backend – Nodejs Express на Typescript 17](#_Toc129859733)

[2.2.3 Authentication – Firebase 19](#_Toc129859742)

[2.2.4 Структура на базата от данни 20](#_Toc129859744)

[ТРЕТА ГЛАВА 22](#_Toc129859769)

[3.1 Frontend 22](file:///E:\School\diplomnaRabota\diplomna-rabota\Документация.docx#_Toc129859781)

[3.2 Backend 28](file:///E:\School\diplomnaRabota\diplomna-rabota\Документация.docx#_Toc129859782)

[ЧЕТВЪРТА ГЛАВА РЪКОВОДСТВО НА ПОТРЕБИТЕЛЯ 36](#_Toc129859783)

[4.1 Инсталация и конфигурация 36](#_Toc129859784)

[4.2 Ползване на приложението: 37](#_Toc129859799)

[Използвана литература 43](#_Toc129859820)