## **TЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ**

**

**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 изместилои изместиха фокуса си към мобилни приложения, позволявайки на потребителите да имат достъп до своите акаунти и да се ангажират с други хора в движение. Това преминаване към мобилни устройства създаде необходимост от приложения за социални медии, които са предназначени специално за мобилни платформи и отговарят на променящите се нужди и предпочитания на потребителите.

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 приятелите си, както и да разглеждат колекциите на други. Приложението не е предназначено да бъде широкообхватна платформа като Facebook или Instagram, а по-скоро по-интимно пространство, където потребителите могат да разгледат интересите и мненията на своите приятели.

# Използвани термини

* Middleware – обработваща заявката функция

# ПЪРВА ГЛАВА

* 1. Възможни среди и технологии за реализация:
     1. Frontend
        1. React Native: React Native е популярна рамка за разработка на мобилни приложения, която позволява на разработчиците да създават високопроизводителни мобилни приложения както за iOS, така и за Android платформи, използвайки JavaScript и React. React Native предлага голяма общност от разработчици, библиотеки и инструменти на трети страни, както и гъвкав и ефективен процес на разработка.
        2. Flutter: Flutter е рамка за разработка на мобилни приложения, разработена от Google, която позволява на разработчиците да създават висококачествени, крос-платформени мобилни приложения за iOS и Android платформи, като използват една кодова база. Flutter използва език за програмиране Dart, който предлага функции като компилация Just-In-Time, Hot Reload и модерен синтаксис.
        3. Swift: Swift е език за програмиране, разработен от Apple, който се използва за разработване на собствени приложения за iOS. Swift е мощен и модерен език, който предлага безопасност, бързина и интерактивност при разработването на приложения. Swift е популярен избор за разработка на приложения за iOS и неговият синтаксис е лесен за научаване от разработчици, които вече са запознати с други езици за програмиране като JavaScript или Java.
        4. Kotlin: Kotlin е език за програмиране, разработен от JetBrains, който се използва за разработване на собствени приложения за Android. Kotlin предлага функции като нулева безопасност, функции за разширение и концепции за функционално програмиране, които го правят популярен избор за разработка на приложения за Android.
     2. Backend
        1. 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.
        2. 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.
        3. 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.
        4. 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 с него.
        5. 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.
  2. Конкурентни продукти
     1. Letterboxd  
        Letterboxd е приложение, основно за споделяне на мнения за филми, имащо огромен брой функционалности, една от които – създаване на списъци с филми.
     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.
     3. Listy (1)
     4. Listy (2)  
          
        Като цяло, въпреки че има многобройни конкуренти в контекста на социалните мрежи, Collections се отличава, като предлага по-малко, лично пространство за споделяне на интереси с приятели и същевременно не е специално за конкретна тема. Като се фокусира върху подредени списъци и използва най-новите мобилни технологии, Collections има потенциала да създаде уникална ниша в контекста на социалните медии.

# ВТОРА ГЛАВА

* 1. Изисквания
     1. Логин/регистрация
     2. Търсене на потребител
     3. Следване на потребител
     4. Създаване на класация/колекция
     5. Добавяне на елемент в класация
  2. Използвани технологии
     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 Collections, реших да използвам Flutter като технология за изграждане на frontend-а на приложението. Ето някои от предимствата на технологията:
        1. 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.
        2. 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.
        3. 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.
        4. 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. Backend – Nodejs
        1. Authentication – Firebase
  3. 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

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.? 

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 като .get(), .post(), .put() и .delete().

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.

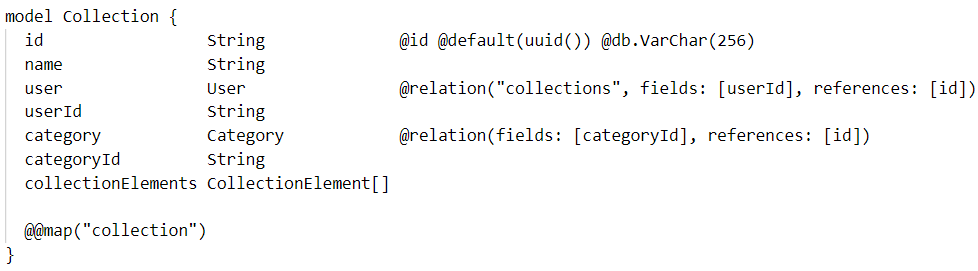
getCollectionById: извлича колекция с посочения идентификатор от базата данни. Той също така включва всички елементи, свързани с колекцията.

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.

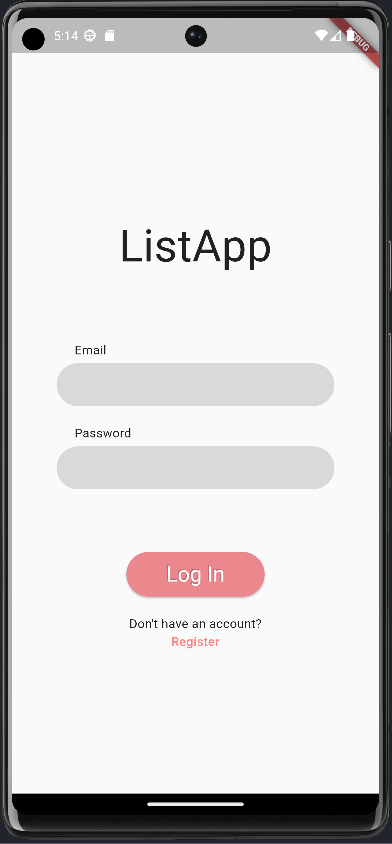
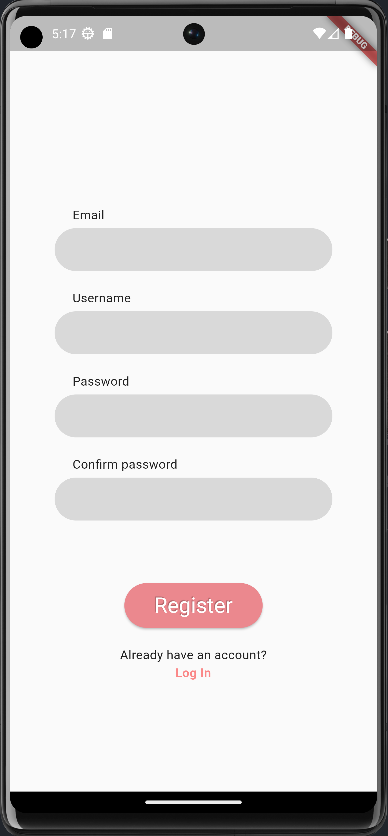
addElementToCollection: добавя нов елемент с посочения elementId и позиция към колекция с посочения collectionId.

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. Файлът е написан на езика на Prisma, schema, който се използва за описание на схемата на базата данни по кратък и интуитивен начин. На фиг.? се вижда как е дефиниран моделът на “колекция” за базата данни.
* Папка „Middleware“

# ЧЕТВЪРТА ГЛАВА РЪКОВОДСТВО НА ПОТРЕБИТЕЛЯ

* 1. Инсталация и конфигурация
     1. Клонирайте гитхъб хранилището
     2. Свалете и инсталирайте postgresql и създайте база данни с име по избор
     3. 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”
     4. Свалете и инсталирайте Node.js
     5. В backend директорията изпълнете командата “npm install”
     6. Инсталирайте и настройте flutter (може да следвате инструкциите на docs.flutter.dev)
     7. Ако не използвате вградения емулатор, а външно устройство, ще трябва устройството да е на същата мрежа и да промените “BASE\_URL” в “.env” файла директорията “/frontend/list\_app” на IP-то на сървъра.
  2. Стартиране на приложението
     1. В “backend” директорията на проекта изпълнете командата “npm start”
     2. Стартирайте самото приложение
  3.  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

# ЗАКЛЮЧЕНИЕ