**Софийски Университет „Св. Климент Охридски“**

**Факултет по математика и информатика**

**Кристиян Стоянов Стоянов**

**ФН 45559**

**09.02.2018г**

**Резлизирани функционалности спрямо project summary-то на проекта:**

* Home page с информация за платформата.
* Регистрация на потребител чрез имейл и парола.
* Feed екран, където се избира от настоени от потребителя feed presets, които съдържат набор от сайтове като източник на новини.
* Конфигурация на Feed ресурс. Позволява на потребителя да създава, променя и изтрива news feeds.
* Kонфигурация на News sources. Позволява на потребителя да създава, променя и изтрива източници на новини.
* Login в системата след въвеждане на имейл и парола на вече създаден потребител.

**Нереализирани функционалности спрямо началното описание на проекта:**

* OAuth регистрация/логин
* User roles – администратор и премиум потребител
* Филтри във feed страницата за търсене и сортиране на статии
* Ticketing system
* Admin panel

**Използвани технологии и библиотечни модули:**

* Spring Web MVC <https://spring.io/>
* Hibernate <https://hibernate.org/>
* JSoup <https://jsoup.org/> (DOM parser, used in the crawler to get the article information)
* Angular 8 <https://angular.io/>
* Bootstrap <https://getbootstrap.com/>
* MySQL <https://www.mysql.com/>

**Архитектура на проекта:**

Проекта е изграден от 3 основни модула. Backend API, Crawler и Frontend.

* Backend API – изграден със Spring Framework, отговаря за менажиране на основните ресурси в приложението, които са User, News Source, Feed, Article. Служи за автентикация на потребителя и свързва Frontend частта на приложението с MySQL база данни.
  + За всеки от ресурсите News Source, Feed и Article имаме съответно Repository Interface (който разширява CrudRepository и ни дава връката ни с базата), Service Interface и имплементация, която съдържа бизнес логиката и извършва операциите с базата чрез инжектиран Repository; и съответно контролер, който служи за получаване и обработване на http заявките идващи към api-то.
  + По интересен е случая при User ресурса, там също имаме Repository, Service и Controller, но service-a съдържа и бинес логика за хеширане на паролата на потребителя, генериране на токен (идентификатор) и автентикация на потребителя по имейл и парола.
* Crawler – изграден със Spring Framework, извършва работата на service, който чете въведените от потребителите News Sources и по тяхната конфигурация scrape-ва сайтовете за новинарски статии, които запазва в базата и чрез Backend API-то информацията достига до Frontend модула на приложението.
  + Има достъп до Article и News Source модела и техните съответни Repositories, за да извършва операциите за взимаме и записване на информация в базата данни.
  + Има CrawlerService, който стартира процес (след конструирането си), който взима всички News Sources от базата данни, според конфигурацията им намира url-ите на статиите и взима техния html, след това ги записва в Article модел и следователно в базата данни, за да бъдат достижими за потребителите на приложенито. Запазват се линковете на обходените статии, за да се избегнат дупликации и ненужно srape-ване на информация.
* Frontend – изграден с Angular 8.3.24, зарежда информацията от базата, чрез backend api-то и я показва на потребителя под формата на Single Page Application.
  + Съдържа следните компоненти:
    - Article Component – зарежда информацията на статия по id от backend-a и я визуализира за потребителя.
    - Feed Component - главният компонент на приложението, зарежда feed-овете на влезлия потребител и спрямо избора на feed от страна на потребителя зарежда статиите от източниците свързани с този feed.
    - Feed Edit Component – изплозва се за да се промени съществуващ вече feed. Данните за feed-a се зареждат от backend частта и след направените промени се изпращат отново в backend-а, за да се запишат в базата.
    - Create Feed Component – използва се за създаване на нов Feed от потребителя, освен това в дясната страна визуализира списък със съществуващите feed-ове на потребителя и дава хипервръзка насочваща ги към Feed Edit Component-a за даден feed.
    - News Source Edit Component – подобно на Feed Edit Component използва се за редакция на същесвуващ вече News Source.
    - Create Source Component – подобно на Create Feed Component, използва се за създаване на нов News Source и дава възможност за навигация към компонента за редактиране на News Source.
    - Login Component – форма за влизане в системата с имейл и потребителско име.
    - Register Component – форма за създаване на нов потребител, изплозвайки email и парола.
    - Logout Component – премахва userToken-а изплозван за автентикация от localStorage-a на браузера, превръщайки потребителя в анонимен.

**API Specification:**

* **POST /api/users/new**
  + Body: { name:String, email:String, password:String}
  + Creates a User if the body is valid and returns the generated token for the user with the name and email to the user, bundled in a JSON.
* **GET /api/users/{userToken}**
  + Returns the userToken, name and email of the user, whose token was sent as a path parameter.
* **POST /api/users/auth**
  + Body: { email:String, password:String}
  + If the credentials match a user returns their token, name and email bundled in a JSON.
* **DELETE /api/users/deleteUser?token={userToken}**
  + Deletes the user identified by the provided token(as query parameter), intended for the admin panel.
* **POST /api/feeds/new**
  + Body: { name:String, userToken:String, ids:Long[] }
  + Creates a new feed that belongs to the user identified by the userToken and containing the News Sources with ids in the Long array}
  + Returns the newly created Feed.
* **GET /api/feeds/get?token={userToken}**
  + Returns a list of the Feeds that belong to the user identified by the userToken provided as a query parameter.
* **GET /api/feeds/get/{feedId}**
  + Returns a single feed by a given id as a path parameter.
* **PUT /api/feeds/edit**
  + Body: { id:Long, name: String, userToken: String, ids: Long[] }
  + Updates the feed identified with the id parameter in the body of the request with the values of the other fields in the JSON. Ids is the ids of the news sources associated with the feed.
* **DELETE /api/feeds/delete/{feedId}**
  + Deletes the feed identified with the feedId provided as a path parameter.
  + Returns the resource which was deleted.
* **GET /api/sources/get/{sourceId}**
  + Returns the News Source identified by the sourceId provided as a path parameter.
* **POST /api/sources/new**
  + Body: { link: String, linkSelector: String, titleSelector: String, contentSelector: String, imageSelector: String, userToken: String}
  + Creates a new Source belonging to the user identified with the userToken provided in the body of the request.
* **PUT /api/sources/edit**
  + Body: { id: Long, link: String, linkSelector: String, titleSelector: String, contentSelector: String, imageSelector: String}
  + Updates the News Source identified by the id in the body of the request with the other values in the JSON body.
* **GET /api/sources/index**
  + Returns a list of all the News Sources in the database.
* **DELETE /api/sources/delete?id={sourceId}**
  + Deletes the resource identified by the sourceId given as a query parameter.
  + Returns the deleted News Source.
* **GET /api/articles/{articleId}**
  + Returns the article identified by the articleId given as a path paramether. The response body looks as follows:
    - {id: Long, title: String, content: String, image: String, link:String, newsSource:Object}
* **POST /api/articles/new**
  + - Body: { title: String, content: String, image: String, link:String }
    - Creates and returns the new article with a generated id
* **GET /api/articles/bySource/{newsSourceId}**
  + Returns a JSONArray of articles that were found from the News Source with the specified id as a path parameter.
* **PUT /api/articles/update**
  + Body: {id: Long, title: String, content: String, image: String, link:String, newsSource:Object}
  + Updates the record with the same id in the databse with the new values sent in the JSON body
* **DELETE /delete/{articleId}**
  + Deletes the article with the specified id as a path parameter
  + Returns the deleted database entry

**Начин на инсталация:**

\*Prerequisites: MySQL server, JRE, Maven, NPM, Angular CLI

1. Създаване на jar файл за backend и crawler-а. И двата проекта използват Maven, така че просто изпълнение на командата *mvn clean install* в директорията където се намира pom.xml съответно за news-aggregator и crawler проектите.
2. Стартирайте първо news-aggregator проекта, като в application.yml файла се конфигурира връзката с mysql базата.
3. News-aggregator ще създаде схемата на базата и след това може да се стартира и crawler-a, който само валидира схемата.
4. След като backend-a и crawler-a са стартиране, навигирайте в папката frontend и използвайки командата ng serve ще build/compile angular приложението и ще се вдигне локален сървър (по подразбиране на порт 4200) и там можете да заредите frontend частта на приложението от вашия браузър.
5. Добавил съм .sql файл с няколко готово конфигурирани news sources, за да се спести време при тестване.

**Кратка потребителска информация за ползване на продукта:**

Интерфейсът на приложението е сравнително семпъл и лесен за използване.

Имаме хоризонтално меню с основните view-та. Първо ще разгледаме configure sources. Там имаме форма за създаване на news source или за редакция на вече съществуващите такива. Един източник на новини има следните полета:

* Link (homepage-a на страницата от която ще взимаме новини)
* Link Selector (css selector, който взима всичките <a> тагове, които водят към статии)
* Title Selector (css selector, който трябва да вземе заглавието на статия от html-a на статията)
* Content Selector (css selector, който трябва да вземе съдържанието (най-често <p> таговете) на статия от html-a на статията)
* Image Selector (css selector, който взима <img> таг, който съдържа снимката на статията (thumbnail))

Препоръчително е да се изберат възможно най-точни css selector-и, за да се избегне погрешно селектиране на елемнти.

След като се запознахме с източниците на новини и как да създадем такива, можем да погледнем менюто configure feeds. Тук можем да си създадем feed, който практически групира news sources под определено име (оригиналната идея беше да се запазват и филтри към всеки фийд). Формата е self-explanatory.

След като сме създали feed можем да отидем във feed menu-то, и там от падащия списък избирам кой feed да бъде визуализиран.

Ако сме избрали feed и източниците в този фийд са конфигурирани правилно и crawler-а е включен трябва да получаваме статии от избраните от нас източници. При кликане върху заглавието на статия или хипервръзката MORE се отваря страница която визуализира статията със заглавие, снимка и съдържание.

Идеята на приложението е да се конфигурират голям на брой сайтове и да бъдат групирани в отделни feed-ове базирани на дадена тематика (ex. Business, Bulgaria News, Politics, etc.) и да не се налага на потребителя да обикаля всичките сайтове поотделно, за да събира информация на дадена тема.

**Заключение**

Системата има доста насоки в който да се развива. Поради ограничения във времето на създаване (заради липса на организационни умения от моя страна), доста от функционалностите не са завършени. Като например филтри и сортиране в отделните feeds, user roles, ticketing system (за помощ с конфигурация на нови източници на новини) и admin panel. Security issues, има липса на добра автентикация, API-то е достъпно дори без използването на userToken, token-a не се генерира всеки път при логин и няма expiration. Понякога се връща User обект(съдържащ в себе си сетивна информация) съдържащ се в обект от тип feed/source. Други проблеми, които мога да установя са неконсистентност в interface-a на api-то (използване на query и path parameters, без точна конвенция).

Проблеми при crawler модула, са твърде симплистичен алгоритъм за взимане на статии, не ефикасен, не се запазват обходените urls в таблица в базата и при изключване на приложението се губи тази информация. Самото конфигуриране на източниците изисква познания по css, което не може да се очаква от обикновенния потребител.

Проблеми при frontend частта, включват: повтаряемост на код, не използване на пълните възможности на angular, навигация с хиперлинкове вместо с router. От гледна точка на изглед, UI-a не изглежда добре, твърде минималистичен.

Въпреки това, ядрото на приложението и най-основните му функционалности работят. Архитектурата е остановена. Установява се връзка между отделните модули на приложението коректно, връзката с базата се осъщестява коректно. Backend частта обработва заявките и менижира ресурсите правилно. Освен това има добро разделение на бизнес логика от операции с Data Layer, и от приемане и обработване на заявки. Crawler модула, въпреки че е доста базов, извършва правилно задачите си и предоставя статиите на потребителите.

Няколко от срещнатите и преодолени трудности са:

* Дизайн и построяване на основните модели и връзките между тях
* Приемането на заявки с чужд произход (Cross-Origin)
* Хеширане на паролата на потребителя
* Динамично променяне на състоянието на компоненти във frontend частта, спрямо промени в localStorage-a на браузера. Преодоляно чрез използване на Observable и изнасяне на функционалността в отделен service.

Трудности, които не бяха преудолени(за времето на писане на този проект):

* Автентикация на потребителя, чрез Spring Security
* Създаване на по-пригледен UI
* „По-умно“ обхождане на сайтове и parse-ване на информация

**Използвани материали:**

* База на css за grid-view на статиите във Feed менюто: <https://w3layouts.com/gridzilla-corporate-multipurpose-flat-bootstrap-responsive-web-template/>
* Документацията на Angular: <https://angular.io/docs>
* Изплозвани материали: <https://spring.io/docs>
* RxJS: <https://rxjs-dev.firebaseapp.com/guide/subject>