

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

**Dist – Aplicație care sugerează articole pe baza
intereselor utilizatorului**

propusă de

Cătălin-Constantin Manole

Sesiunea: *Iulie, 2018*

Coordonator științific

Drd. Colab. Florin Olariu

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI
FACULTATEA DE INFORMATICĂ

Dist – Aplicație care sugerează articole pe baza intereselor utilizatorului

Cătălin-Constantin Manole

Sesiunea: *Iulie, 2018*

Coordonator științific

Drd. Colab. Florin Olariu

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele _____

Data _____

Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemnatul(a)

domiciliul în

născut(ă) la data de, identificat prin CNP,
absolvent(a) al(a) Universității „Alexandru Ioan Cuza” din Iași, Facultatea de
..... specializarea, promoția
....., declar pe propria răspundere, cunoscând consecințele falsului în
declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr.
1/2011 art.143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul:

.....elaborată sub îndrumarea dl. / d-na
....., pe care urmează să o susțină în fața
comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin
orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la
introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări
științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei
lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie
răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am
întreprins-o.

Data azi,

Semnătură student

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „Dist – Aplicație care sugerează articole pe baza intereselor utilizatorului”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *data*

Absolvent *Cătălin-Constantin Manole*

(semnătura în original)

Cuprins

Introducere.....	5
Contribuții.....	6
Capitolul 1: Implementarea componentei server a aplicației	7
Tehnologii utilizate	7
Arhitectura componentei server	7
Extensii utilizate	8
Api expus	9
Exemple de implementare.....	10
Capitolul 2: Obținerea de interese din care să aleagă utilizatorul	13
Abordări anterioare	13
Soluție aleasă	14
Capitolul 3: Implementarea componentei client a aplicației	15
Mediul de dezvoltare	15
Limbajele folosite	15
Descrierea framework-ului	16
Arhitectura componentei client.....	21
Capitolul 4: Obținerea de posturi pe post de mostră pentru interesele găsite.....	22
Capitolul 5: Utilizarea aplicației.....	23
Concluzie	27
Bibliografie.....	29

Introducere

Tema aleasă conține o aplicație ce are rolul de a afișa conținut utilizatorilor ei sub formă de link-uri externe către articole care i-ar putea interesa pe aceștia. Sugestiile sunt generate pe baza intereselor utilizatorului. Cu cât acesta face cunoscute mai multe interese cu atât i se vor putea sugera mai multe articole.

Utilizatorul are posibilitatea de a genera la rândul său conținut prin adăugarea de articole.

Unul din motivele alegerii temei este acela că site-urile web deja existente care promovează conținut extern către utilizatori nu granulează suficient de bine articolele sugerate iar rezultatul este acela că articolele nu sunt grupate suficient de specific. Astfel, un utilizator interesat de un domeniu foarte specific va putea vizualiza un procent foarte mic de articole care chiar să-l intereseze ajungând ca în restul timpului acesta să trebuiască să treacă prin multe titluri care nu-i sunt de interes acestuia. Din acest motiv aplicația pe care am realizat-o își propune să rezolve această problemă și să le confere celor care o utilizează un mijloc mai eficient de a vizualiza acele articole care prezintă un interes pentru utilizator.

În cadrul acestei aplicații utilizatorul va dispune de posibilitatea de a-și alege singur conținutul pe care dorește să-l vizioneze. Acesta va avea posibilitatea să caute în interesele ce îi sunt puse la dispoziție de către aplicație și să aleagă despre care din acestea dorește să vizioneze articole. De asemenea aplicația dispune și de felurite mijloace de filtrare a articolelor în funcție de criteriile (gradul de noutate, număr de aprecieri de la alți utilizatori, articole favorite) puse la dispoziție utilizatorului spre citire.

Contribuții

Tema lucrării a fost propusă și aleasă de către mine după discuțiile cu coordonatorul meu (Drd. Colab. Florin Olariu). În urma feedback-ului pozitiv oferit de către coordonatorul meu am conchis că voi susține tema curentă ca și lucrare de licență. Colaborarea mea cu domnul Olariu a constat în întâlniri săptămânale în care prezentam progresul. Întâlnirile erau organizate în grupuri de câte aproximativ opt persoane astfel încât să putem schimba idei și cu alți colegi și eventual să găsim posibile aspecte ce ar putea fi îmbunătățite pentru fiecare dintre cei prezenți. De asemenea a fost încurajată prezentarea de demo-uri și ținerea de prezentări în fața colegilor pentru a putea simula prezentarea finală ce va avea loc în fața comisiei de licență.

Un alt factor benefic l-a reprezentat utilizarea platformelor de socializare prin intermediul cărora domnul colaborator ne-a putut oferi tuturor acces facil către diverse resurse, ne-a oferit actualizări constante, ne-a motivat și ne-a îndrumat astfel încât să ducem la bun sfârșit lucrarea de licență.

Demonstrarea faptului că lucrarea și munca depusă îmi aparține se poate face observând caracterul evolutiv al *repository-ului* meu de pe **Github** destinat lucrării de licență ce este public. De asemenea acesta oferă și transparență putând fi observate evoluția codului și etapele intermediare prin care s-a ajuns la rezultatul final.

Pentru realizarea lucrării am utilizat cunoștințele și aptitudinile acumulate în cadrul celor trei ani de facultate. Am utilizat cunoștințele de OOP pentru a face codul scris de mine modular, flexibil și reutilizabil în posibile viitoare proiecte. Pe partea de tehnologii utilizate cunoștințele acumulate cât și proiectele realizate la disciplina Introducere în .NET m-au ajutat la realizarea *backend-ului* aplicației. Un element de noutate îl constituie *frontend-ul* unde am folosit un framework popular la momentul curent (Angular 5/6). Pentru realizarea lui am fost nevoit să studiez modul lui de utilizare iar cunoștințele fundamentale dobândite la Tehnologii Web au constituit un punct de plecare pentru mine. Cunoștințele de bază despre HTML / CSS și JavaScript au fost necesare pentru a putea înțelege și utiliza un framework pentru partea de frontend a aplicației.

Capitolul 1: Implementarea componentei server a aplicației

Pentru realizarea aplicației a fost necesar să implementez o component de tip server care să proceseze cererile pe care le fac clienții. Arhitectura de tip client-server este bine cunoscută așa că nu voi mai menționa la ce se referă. Este necesar ca pentru a implementa o aplicație web ce utilizează și oferă servicii să existe o astfel de component ce servește drept punct central de rezolvare a cererilor făcute de către client.

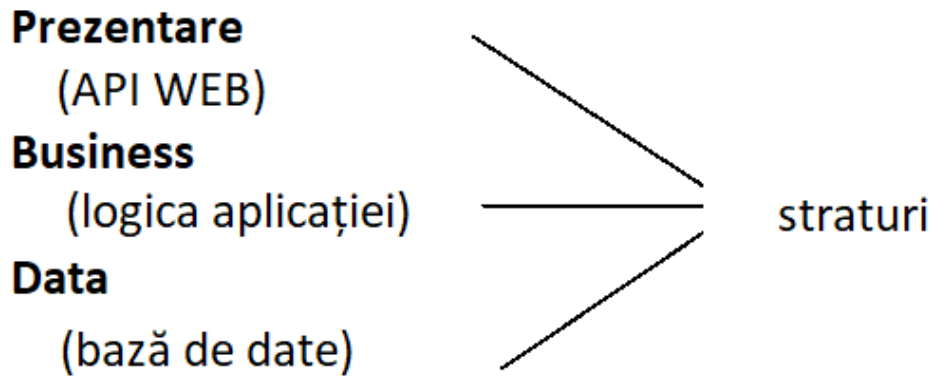
Componenta server va fi responsabilă să stocheze și să pună la dispoziție date fiecărei instanțe a componentei client. Datele sunt constituite din conturile utilizatorilor, articolele pe care le vizionează utilizatorul, articole pe care le-a creat acesta, articolele favorite ale acestuia cât și articolele care i-au plăcut. De asemenea vor trebui reținute interesele utilizatorilor pentru a le putea furniza articole stocate în baza de date pentru care aceștia au afinitate.

Tehnologii utilizate

Am optat să utilizez pe partea de server framework-ul ASPNET Core (1) deoarece acesta este modern, independent de platformă, cu sursă deschisă, performant și este gândit să ofere suportul dezvoltării de aplicații conectate prin internet.

Arhitectura componentei server

Serverul este structurat pe nivele și straturi (*tiers and layers*) (2). În cazul de față vorbim despre o aplicație cu 3 straturi (prezentare, logică, date). Această arhitectură aduce felurite avantaje printre care merită menționate: flexibilitatea de a putea înlocui un anumit strat din punct de vedere tehnologic fără a le afecta pe celelalte, scalabilitate, independență între straturi, mentenanță mai ușoară a codului etc.



Figură 1 Structura componentei server

Data

- Core – conține clase de domeniu, clase de configurare pentru FluentApi, componente legate de autentificare, interfețe pentru repository-uri (3)
- Persistence – DbService, o clasă ce oferă suportul pentru a realiza diferite operații pe baza de date. De asemenea tot aici sunt stocate migrările

Business

- Business – conține logica construită peste entitățile cu care operează aplicația încapsulată la nivel de Repository
- AuthService – o componentă ce are rolul de a genera un token pentru autentificarea prin Jwt (4) (JSON web token)

Prezentare

- WebApi – API-ul REST expus de aplicația server

Extensii utilizate

Am utilizat diverse extensii pentru a ușura procesul de dezvoltare și pentru a nu fi nevoit să implementez de la 0 anumite aspecte

- *Microsoft.AspNetCore.Authentication.JwtBearer* - middleware care permite unei aplicații să primească un token OpenID Connect Bearer.
- *Microsoft.AspNetCore.Identity* – oferă suport pentru gestionarea datelor de conectare și datelor utilizatorului, permite adăugarea unor funcții de conectare la aplicație și ușurează personalizarea datelor despre utilizatorul conectat.

- *AutoMapper.Extensions.Microsoft.DependencyInjection* – extensie ce oferă suport de DI (injectarea dependențelor) pentru AutoMapper. AutoMapper facilitează maparea între obiecte astfel încât sporește productivitatea (nu mai e nevoie să scriem codul ce ar fi făcut maparea) și menține codul mai curat (mai puține linii de cod).
- *FluentValidation.AspNetCore* – ajută la construirea unor reguli de validare *strongly-typed* printr-o interfață de tipul fluent
- *Swashbuckle.AspNetCore* – utilitare Swagger pentru documentarea și exemplificarea de API-uri

Api expus

Pentru a putea face administrarea resurselor prezente în aplicație serverul expune următoarele metode:

POST /api/Auth/register

POST /api/Auth/login

POST /api/Favorites

DELETE /api/Favorites/{id}

POST /api/Likes

DELETE /api/Likes/{id}

GET /api/Posts

PUT /api/Posts

POST /api/Posts

DELETE /api/Posts/{id}

...

Api-ul de tip REST care folosește în componența sa verbe cât și utilizarea resurselor în rută face ca acesta să fie autodocumentat deoarece se poate înțelege fără detalii suplimentare intenția fiecărei metode în parte și care vor fi rezultate în urma apelării lor.

Exemple de implementare

În stratul Data sunt definite toate entitățile utilizate în aplicație. Un exemplu de astfel de entitate se poate observa în figura de mai jos.

```
public class Post : ISoftDeletable
{
    public Guid Id { get; set; }
    public string Title { get; set; }
    public string SourceUrl { get; set; }
    public string Description { get; set; }
    public DateTime CreatedAt { get; set; }
    public Guid UserId { get; set; }
    virtual public User User { get; set; }
    virtual public ICollection<PostInterest> PostInterests { get; set; }
    virtual public ICollection<Favorite> Favorites { get; set; }
    virtual public ICollection<Like> Likes { get; set; }
}
```

Figură 2 Exemplu de cod al unei entități din partea de server

Ca și particularitate se poate observa faptul că entitatea extinde interfața ISoftDeletable ce se comportă ca o metadată pentru context. În acest fel clasei îi este atribuit mecanismul de ștergere aparentă. Obiectul în sine nu este șters ci este doar marcat să nu mai fie obținut atunci când se obțin intrări din baza de date.

```
public class PostConfiguration : IEntityTypeConfiguration<Post>
{
    public void Configure(EntityTypeBuilder<Post> builder)
    {
        builder.HasKey(p => p.Id);
        builder.Property(p =>
p.SourceUrl).HasMaxLength(2048).IsRequired();
        builder.Property(p => p.Title).HasMaxLength(256).IsRequired();
        builder.Property(p =>
p.Description).HasMaxLength(500).IsRequired();
        builder.Property(p => p.CreatedAt).IsRequired();

        builder
            .HasOne(pt => pt.User)
            .WithMany(t => t.Posts)
            .HasForeignKey(pt => pt.UserId)
            .IsRequired();

        builder.HasMany(x => x.Favorites)
            .WithOne(x => x.Post)
            .HasForeignKey(x => x.PostId);
    }
}
```

Figură 3 Exemplu de configurare a relațiilor entității Post

În figura 2 se poate observa faptul că entitatea este legată de alte entități prin relații. Când acele proprietăți ce referențiază alte entități sunt accesate acestea vor fi aduse pe moment din baza de date datorită mecanismului de încărcare leneșă (lazy loading).

Relațiile dintre entități și specificațiile proprietăților sunt descrise separat într-o clasă destinată acestui aspect.

```
public class Repository<TEntity> : IRepository<TEntity> where TEntity : class
{
    protected readonly DbContext Context;
    protected readonly DbSet<TEntity> _entities;

    public Repository(DbContext context)...
    virtual public TEntity Get(Guid id)...
    virtual public IEnumerable<TEntity> Get(int pageIndex, int pageSize,
    Ordering<TEntity> ordering = null)...
    public IEnumerable<TEntity> Find(Expression<Func<TEntity, bool>> predicate)...
    public TEntity SingleOrDefault(Expression<Func<TEntity, bool>> predicate)...
    public virtual void Add(TEntity entity)...
    public void AddRange(IEnumerable<TEntity> entities)...
    public void Remove(TEntity entity)...
    public void RemoveRange(IEnumerable<TEntity> entities)...
}
```

Figură 4 Implementarea generică a unui repository

La stratul de logică de business am definit un repository generic care oferă în mare funcționalitățile de bază pe care ar trebui să-l expună orice repository. La nevoie unele repository-uri expun și funcționalități suplimentare.

```
public class LikeRepository : Repository<Like>, ILikeRepository
{
    public LikeRepository(DbContext context) : base(context)
    {
    }

    public Guid? GetLikeIdOfPostForUser(Guid postId, Guid userId)
    {
        var like = _entities.FirstOrDefault(l => l.PostId == postId && l.UserId ==
        userId);
        return like?.Id;
    }
}
```

Figură 5 Exemplu de repository ce oferă o funcționalitate extinsă

```

[Authorize(Policy = "ApiUser")]
[Route("api/[controller]")]
[ApiController]
public class PostsController : ControllerBase
{
    private readonly IUnitOfWork _unitOfWork;
    private readonly IMapper _mapper;
    public PostsController(IUnitOfWork unitOfWork, IMapper mapper)...

    [HttpGet]
    public ActionResult<List<ReadPostModel>> Get(Guid userId, bool selfPosts, int?
pageIndex, int? pageSize, string orderBy)...

    [HttpPost]
    public ActionResult<Post> Create([FromBody] CreatePostModel post)...
    [HttpPut]
    public ActionResult<Post> Edit([FromBody] EditPostModel post)...

    [HttpDelete("{id}")]
    public IActionResult Delete(Guid id)...
}

```

Figură 6 Exemplu de controller

Aspecte care merită menționate sunt din nou metadatele prezente pe clasă. Acestea oferă informația cum ar fi că clasa definită în imaginile de mai sus este un controller, ruta pentru care acesta expune metode și faptul că acesta refuză să rezolve cereri dacă cel care le face nu este autentificat.

Autentificarea este realizată prin JSON Web Token, standardul industrial pentru a reprezenta în mod sigur cererile între două părți.

Ori de câte ori utilizatorul dorește să acceseze o rută sau o resursă protejată, cel care face accesul trebuie să trimită token-ul, de obicei în antetul de autorizare folosind schema Bearer. Acesta este un mecanism de autentificare lipsită de stare deoarece starea utilizatorului nu este salvat niciodată în memoria serverului. Rutele protejate ale serverului vor verifica existența unui token valid în antetul "Authorization" iar dacă acesta este prezent, utilizatorul va avea acces la resurse protejate. Deoarece token-urile JWT sunt autonome toate informațiile necesare sunt încorporate în ele, reducând necesitatea de a se realiza interogări pe baza de date de mai multe ori.

Capitolul 2: Obținerea de interese din care să aleagă utilizatorul

Pentru ca un utilizator să poată să își găsească articole pe baza intereselor lui este necesar ca în baza de date să avem un număr considerabil de intrări astfel încât să fie rare cazurile când un utilizator nu va găsi vreun articol pe baza intereselor sale.

Abordări anterioare

Pentru a realiza o astfel de baza de date am recurs inițial la a utiliza un extras al bazei de cu toate titlurile articolelor postate pe Wikipedia. Ca rezultat, am obținut un fișier cu aproape 14 milioane de intrări. Acestea din urmă având nevoie de o preprocesare înainte de a putea fi folosite în aplicație deoarece conțin caractere non-alfanumerice precum (|, _ etc).

Sanitarizarea datelor s-a făcut incremental. Inițial am eliminat liniile care nu respectau un format anume folosindu-mă de ajutorul expresiilor regulate. Limbajul de programare utilizat pentru a realiza curățarea intereselor a fost *Python* (5) datorită faptului că este un limbaj de scriptare ușor de utilizat și este destinat unor astfel de treburi.

```
import re

regex_match = re.compile('^[a-zA-Z0-9_]*$')
regex_norm = re.compile('[^a-zA-Z0-9]')
counter = 0

with open("output_db_v3", "w") as outf:
    with open("input_db_v3", "r", encoding="utf-8") as inf:
        for line in inf.readlines():
            counter += 1
            if counter % 1000000 == 0:
                print(counter)

            if not len(line):
                continue

            if not line[0].isalpha():
                continue

            if not regex_match.match(line):
                continue

            regex_norm_str = regex_norm.sub(' ', line)
            if len(regex_norm_str):
                outf.write(regex_norm_str)
                outf.write('\n' )
            else:
                outf.write(line)
```

Figură 7 Script care elimina liniile ce nu respecta un format

Din păcate această abordare nu a fost una practică pentru mine deoarece necesita o filtrare prea complexă. Cantitatea de informație era mult prea abundentă și nu ar fi fost facil să încerc să triez datele fiind probabil nevoie de o metodă foarte sofisticată și ingenioasă astfel încât să obțin o sinteză a datelor ce ar avea caracter relevant într-un final.

Soluție aleasă

Am ales într-un final să utilizez o listă foarte generală ce conține diverse activități pe care le-ar putea realiza un om ca și hobby. Cu toate că această listă nu este una comprehensivă cu ce ar putea interesa pe o persoană ea reprezintă un punct de start iar orice îmbunătățire ar putea fi binevenită astfel încât utilizatorul să își poată regăsi cât mai multe din interese în cadrul aplicației.

Capitolul 3: Implementarea componentei client a aplicației

Aplicația are nevoie de o componentă client ce va constitui interfața pe care o folosește utilizatorul. Acesta trebuie să fie capabil să se poată înregistra, autentifica, să își adauge interese, să vizualizeze articole.

Mediul de dezvoltare

Am folosit ca și mediu de dezvoltare al aplicației client IDE-ul *Visual Studio Code* dezvoltat de către Microsoft. Acest program este un editor de cod sursă ce este capabil să ruleze pe platforme multiple (Windows, macOS, Linux) include suport pentru depănare, utilizarea Git-ului este incorporată în acesta.

De asemenea IDE-ul prezintă niște funcționalități pe care orice mediu de dezvoltare competent ar trebui să le ofere precum: evidențierea sintaxei, autocompletarea codului, fragmente de cod (code snippets) și suport de refactorizare a codului (redenumire de funcții și variabile). Un alt aspect convenient este faptul că aplicația este personalizabilă, astfel încât utilizatorii pot schimba tema editorului, comenzile rapide de la tastatură și poate face diverse setări în funcție de preferințe. Acesta este gratuit și cu sursă deschisă.

Limbajele folosite

Limbajele folosite sunt în mare parte cele universale dezvoltării de pagini web. HTML ca limbaj de marcă, CSS pentru stilizarea elementelor HTML. TypeScript este un limbaj de programare cu sursa deschisă dezvoltat și întreținut de Microsoft. Este un superset al JavaScript-ului și oferă în mod opțional verificarea statică a tipului variabilelor. Utilizarea lui a survenit în urma alegerii framework-ului de dezvoltare a componentei client (Angular 5).

Descrierea framework-ului

Pentru realizarea aplicației client am recurs la a folosi framework-ul Angular versiunea 5. Angular este un framework cu sursă deschisă (open source) dezvoltat și menținut de Google. În ultimul timp a devenit probabil cel mai folosit. Trăsătura lui caracteristică o constituie faptul că extinde HTML-ul cu etichete și proprietăți noi. De asemenea acesta implementează conceptul de directive ce conferă abilitatea de a crea elemente HTML noi.

Acest framework este destinat dezvoltării de aplicații pe o singură pagină (Single Page Application, SPA) (6). Acestea sunt diferite de o aplicație web standard prin faptul că navigarea de la o pagină la alta nu mai necesită generarea completă a paginii.

Într-o SPA site-ul este încărcat doar o singură dată prima oară când este accesat. După aceasta oricât s-ar naviga pagina nu mai este niciodată încărcată complet (nu va mai avea loc nici o reîncărcare de pagină). Evident că site-ul va fi format din mai multe pagini dar acestea sunt încărcate sub formă de fragmente HTML, fără a se reîncărca complet pagina.

Un prim beneficiu al unui SPA este faptul că este mult mai rapidă decât o aplicație standard deoarece utilizatorul nu trebuie să aștepte ca o pagină să fie complet încărcată ca acesta să o poată fi utilizată. Un alt beneficiu îl constituie faptul că o bună parte din funcționalitatea site-ului nu mai este rulată pe partea de server ci acum este rulată de client (în browser prin intermediul JavaScript-ului) ducând la timpi de răspuns de la server mult mai scăzuți.

Modulele (7) reprezintă o modalitate excelentă de a organiza o aplicație și de a o extinde cu capacități din bibliotecile externe.

În Angular bibliotecile exista sub forma de NgModules (exemple: FormsModule, HttpClientModule, și RouterModule). Multe biblioteci terțe sunt disponibile ca NgModule, cum ar fi Material Design, Ionic și AngularFire2.

NgModules consolidează componentele, directivele și conductele (pipes) în blocuri de funcționalitate coezive, fiecare concentrându-se pe o zonă de caracteristici, domeniu de aplicații, flux de lucru sau o colecție comună de utilități.

Modulele pot adăuga, de asemenea, servicii pentru o aplicație. Astfel de servicii ar putea fi dezvoltate de către programator sau să vină din surse externe(cum ar fi routerul Angular și clientul HTTP).

Modulele pot fi încărcate la inițializare (primul acces) când aplicația pornește sau la cerere (lazy) în momentul în care modulul este încărcat asincron de către router.

Metadatele NgModule fac următoarele:

- declară ce componente, directive și țevi (pipe-uri) aparțin modulului.
- face unele dintre aceste componente, directive și țevi publice astfel încât șabloanele de componente ale altor module să le poată utiliza.
- importa alte module cu componentele, directivele și conductele pe care le au componentele din modulul curent.
- oferă servicii pe care le pot utiliza celelalte componente ale aplicației.
- fiecare aplicație Angular are cel puțin un modul, modulul rădăcină (*root*). Modulul respectiv este pornit pentru a lansa aplicația.

```

import { CoreModule } from './core/core.module';
import { AppRoutingModuleModule } from './app-routing.module';

import { AppComponent } from './app.component';
import { DropdownDirective } from './shared/dropdown.directive';
import { HeaderComponent } from './core/header/header.component';
import { NgbModule } from '@ng-bootstrap/ng-bootstrap';
import { PageNotFoundModule } from './pagenotfound/pagenotfound.module';
import { AuthRoutingModule } from './auth/auth-routing.module';
import { PostsRoutingModule } from './posts/posts-routing.module';
import { PostsModule } from './posts/posts.module';
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    NgbModule.forRoot(),
    CoreModule,
    HttpClientModule,
    AuthModule,
    AppRoutingModuleModule,
    AuthRoutingModule,
    PostsModule,
    PageNotFoundModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}

```

Figură 8 Modulul rădăcină al aplicației Dist

Una din tipurile particulare de directive este *componenta*.

Componentele reprezintă un element fundamental al aplicațiilor Angular. Acestea se pot compune construind componente ce conțin la rândul lor alte componente. O aplicație Angular este, prin urmare, doar un arbore al unor astfel de componente, atunci când fiecare componentă este redată (*rendered*) acest lucru se întâmplă în mod recursiv componentelor copii.

Decoratorul de componente (`@Component`) permite marcarea unei clase ca fiind o componentă și furnizarea de metadate suplimentare care determină modul în care componenta trebuie procesată, instanțiată și utilizată în timpul rulării programului.

Componentele reprezintă cel mai elementar element de construcție al unei interfețe de utilizator într-o aplicație construită în Angular. O aplicație Angular este în esență un arbore de componente.

Atunci când fiecare componentă este redată (*rendered*) are loc redarea în mod recursiv și a componentelor copii. La rădăcina acestui arbore este o componentă inițială numită componenta rădăcină (*root*).

Atunci când are loc pornirea unei aplicații Angular, îi spunem browser-ului să redea componenta rădăcină ce declanșează redarea componentelor copil, mecanism prezentat mai sus.

O componentă este un subset al unei directive. Spre deosebire de directivă, componentele au întotdeauna un șablon și numai o componentă poate fi instanțiată pe un element dintr-un șablon.

O componentă trebuie să aparțină unui NgModule pentru a fi utilizată de o altă componentă sau aplicație. Pentru a specifica faptul că o componentă este membră a unui NgModule, ar trebui menționată în câmpul de declarații al acelui NgModule.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Dist';
}
```

Figură 9 Componenta rădăcină a aplicației client

Un aspect ce merită menționat sunt metadatele și valorile acestora prezente în decoratorul clasei. *Selector* se referă la numele elementului ce trebuie scris pentru a instanția componenta în cadrul unui șablon. *templateUrl* se referă la calea către fișierul unde este scris șablonul componentei iar *styleUrls* se referă la calea către fișierul ce conține stilurile css ce se vor aplica pe această componentă.

```

<app-header></app-header>
<div class="container">
  <div class="row">
    <div class="col-md-12">
      <div class="container" infiniteScroll (scrolled)="onScroll()">
        <app-post-card class="row my-2 justify-content-md-center"
[post]="post" *ngFor="let post of posts; let i = index">
          </app-post-card>
        </div>
      </div>
    </div>
  </div>
</div>

```

Figură 10 Exemplu de șablon al unei componente

În figura de mai sus se poate observa apelul componentei *Header* prin elementul *app-header*.

Serviciile reprezintă o modalitate excelentă de a face schimb de informații între clase care nu se cunosc reciproc. Acestea sunt injectate la nivel de componentă prin intermediul mecanismului de injectare al dependențelor oferit de către Angular.

```

import { Injectable } from '@angular/core';
import { Post, CreatePostModel } from '../posts/post.model';
import { Interest } from '../interests/interest.model';
import { InterestsModule } from '../interests/interests.module';
import { HttpClient } from '@angular/common/http';
import { environment } from 'src/environments/environment';

@Injectable({
  providedIn: 'root'
})
export class DataStorageService {
  linkPreviewApiKey = 'placeholderApiKey';

  availableInterests: Interest[] = [];

  activeInterests: Interest[] = [];

  constructor(private httpClient: HttpClient) {
    ...
  }

  getPosts(page?: number, perPage?: number) {
    ...
  }
}

```

Figură 11 Schema unui serviciu în Angular

Se poate observa în figura de mai sus utilizarea instrucțiunii *@Injectable* chiar înainte de definiția clasei *DataStorageService* cât și faptul că metadata *providedIn* are valoarea *"root"*.

Când serviciul este oferit (*provided*) la nivelul rădăcinii (*root*) Angular creează o singură instanță partajată a clasei `DataService` și o injectează în orice clasă care o solicită. Înregistrarea furnizorului în metadatele `@Injectable` permite, de asemenea, Angular-ului să optimizeze o aplicație prin eliminarea unui serviciu în cazul în care acesta nu este folosit nicăieri.

```
import { Component, OnInit, Input } from '@angular/core';
import { NgbActiveModal } from '@ng-bootstrap/ng-bootstrap';
import { Observable, Subject } from 'rxjs';
import { debounceTime, distinctUntilChanged, map } from 'rxjs/operators';
import { DataService } from '../../shared/data-storage.service';
import { Interest } from '../../interest.model';

@Component({
  selector: 'app-display-interests',
  templateUrl: './display-interests.component.html',
  styleUrls: ['./display-interests.component.css']
})
export class DisplayInterestsComponent implements OnInit {
  @Input() name;

  public searchedInterest: Interest;
  public imgThumbnailSelected = '';
  public displayInterestName = '';
  public activeInterests: Interest[] = [];

  constructor(public activeModal: NgbActiveModal, private dataStorageService:
    DataService) {

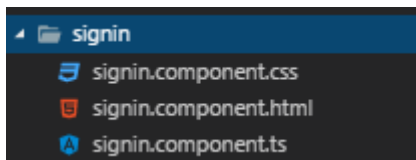
    this.activeInterests = dataStorageService.getActiveInterests(1, 5);
  }

  ngOnInit() {}
}
```

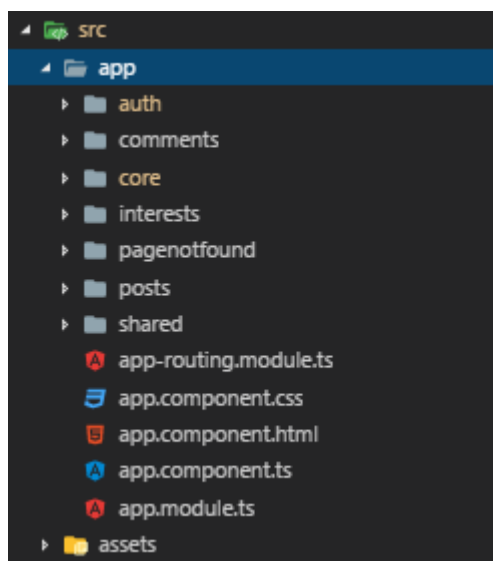
Figură 12 Exemplu de injectare a serviciului `DataService` într-o componentă

Arhitectura componentei client

Clientul este construit sub forma de module în cadrul cărora sunt prezente mai multe componente fiecare cu un rol bine definit. În general o componentă este structurată în 3 fișiere. Unul pentru șablonul componentei, unul pentru stiluri și unul pentru codul clasei componentei.



Figură 13 Structura de fișiere al unei componente



Figură 14 Structura de directoare al aplicației client

Capitolul 4: Obținerea de posturi pe post de mostră pentru interesele găsite

Este necesar ca în lipsa unei comunități de utilizatori care să folosească aplicația să oferim noi niște articole demonstrative pentru a putea vizualiza o utilizare a aplicației în acțiune și pentru a sugera sentimentul de veridicitate al informațiilor prezentate.

Am ales ca soluție să utilizez un API Web gratuit pentru un timp limitat oferit de către motorul de căutare Bing. Motivul alegerii îl constituie numărul rezonabil de cereri HTTP ce se pot realiza în comparative cu competitorul mult mai popular Google care oferă doar 100 de apeluri de API pe zi.

```
import requests
import json

subscription_key = "73c8ac9d13094d6480c2d48fd6e2bd0e"
search_url = "https://api.cognitive.microsoft.com/bing/v7.0/search"

posts = {"posts": []}
print("Started")

i = 0

with open("posts.txt", "w", encoding="utf-8") as outf:
    with open("interests.txt", "r", encoding="utf-8") as inf:
        for line in inf.readlines():
```

```

        i = i + 1
        if i == 4:
            break

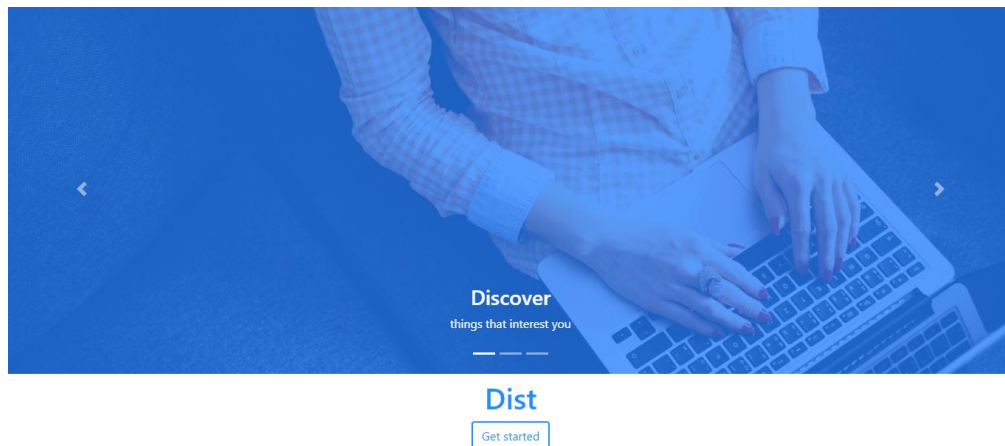
        search_term = line

        headers = {"Ocp-Apim-Subscription-Key" : subscription_key}
        params = {"q": search_term} # "textDecorations":True,
        "textFormat":"HTML"
        response = requests.get(search_url, headers=headers,
        params=params)
        response.raise_for_status()
        search_results = response.json()
        #print(search_results)
        for value in search_results["webPages"]["value"]:
            print(value["name"])
            post = {"title": "", "sourceUrl": "", "description": ""}
            post["title"] = value["name"]
            post["sourceUrl"] = value["url"]
            post["description"] = value["snippet"]
            post["interests"] = [line[:-1]]
            posts["posts"].append(post)
        outf.write(json.dumps(posts))
    print("Done")

```

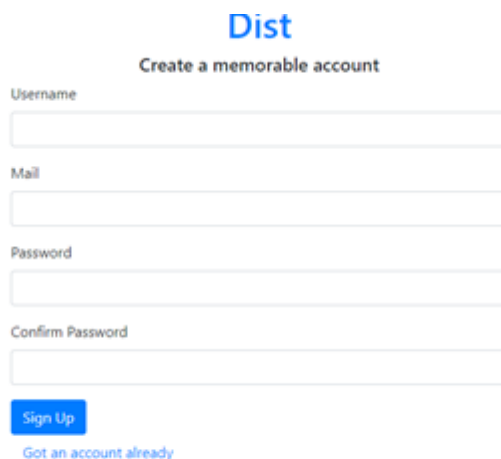
Figură 15 Cod pentru obținerea de posturi prin intermediul api-ului Bing

Capitolul 5: Utilizarea aplicației



Figură 16 Pagina ce întâmpină utilizatorul

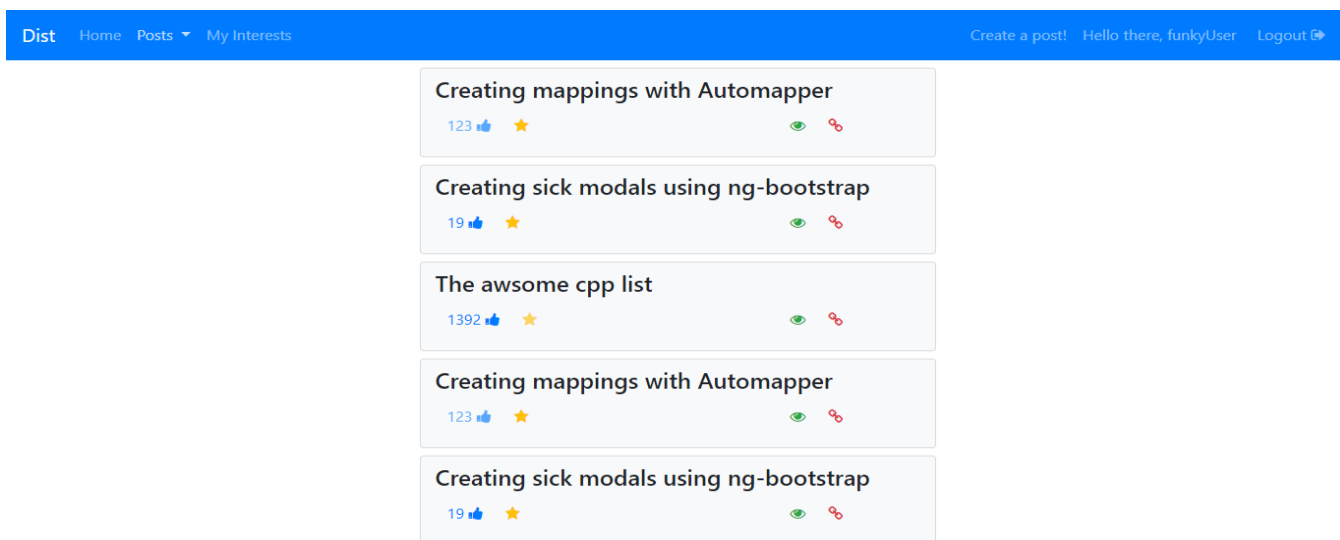
Când utilizatorul accesează prima dată aplicația este întâmpinat de o pagină simplă ce încearcă să îl ajute pe utilizator să își formeze o idee despre aplicație în sine prin afișarea unui carusel ce prezintă în cuvinte puține dar bine alese esența aplicației.



The image shows a registration form for a website called "Dist". The form is titled "Create a memorable account" and includes fields for "Username", "Mail", "Password", and "Confirm Password". Below the fields is a blue "Sign Up" button and a link that says "Got an account already?".

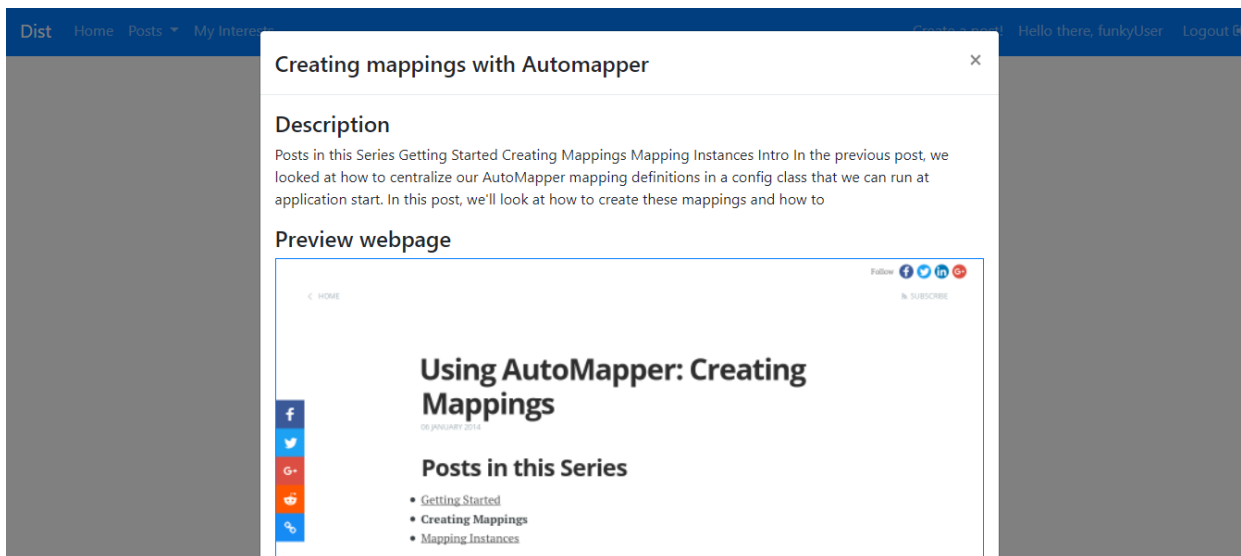
Figură 17 Pagina de înregistrare a unui utilizatorului

Utilizatorul se autentifică cu credențialele sau dacă nu are se înregistrează apoi poate să vizualizeze aplicația în sine.



Figură 18 Vizualizarea articolelor

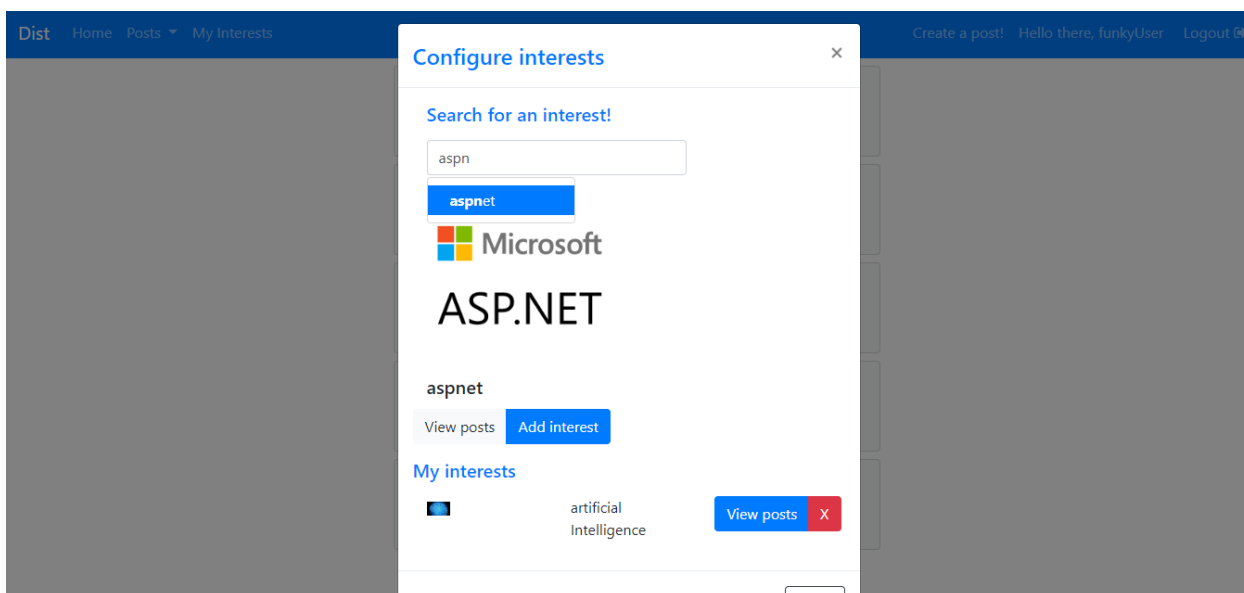
După ce s-a autentificat, utilizatorul poate să vizualizeze articolele ce-i sunt sugerate. Acestea sunt încărcate pe măsură ce utilizatorul dă scroll pe pagină. I se oferă posibilitatea să contribuie la popularitatea unui articol dând un like, să “salveze” postările care-i sunt pe plac adăugându-le la postări favorite. Evident să vizualizeze pagina pe care se află articolul spre al citi sau să vadă o previzualizare a paginii ce deschide totodată o fereastră modală ce-i prezintă mai multe informații despre articol.



Figură 19 Fereastră modală de previzualizare a unui articol web

Previzualizarea este realizată cu un api web sofisticat numit *thum.io* (8) care simulează în spate un browser real pentru a simula cât mai în detaliu condițiile necesare pentru ca pagina să arate cât mai aproape de felul în care este redată când ar accesa-o un utilizator de pe un browser veritabil.

În afară de opțiunea de previzualizare restul opțiunilor sunt în continuare disponibile în josul ferestrei modale.



Figură 20 Fereastra de configurare a intereselor unui utilizator

Dând click pe butonul “*My Interests*” utilizatorului i se va afișa o fereastră modală în care are opțiunea de a căuta noi interese ce nu sunt deja adăugate în lista lui curentă de interese.

Mecanismul de *typeahead* vine în ajutorul utilizatorului astfel încât el poate să exploreze lista de interese și chiar să afle despre interese noi care ar putea să-i stârnească interesul. Acesta se referă la faptul că atunci când utilizatorul introduce caractere în zona de text se caută interese în baza de date și îi sunt afișate utilizatorului sub forma unei liste dropdown.

Acesta poate vizualiza în mod specific posturile pentru acel interes sau să-l adauge ca interes propriu. Sub toate acestea se găsesc lista de interese deja existente.

Aceeași opțiune de vizualizare de posturi este disponibilă pentru fiecare din interesele prezente iar dacă utilizatorul se răzgândește cu privire la un anumit interes el poate opta să îl șteargă din interesele sale.

Concluzie

În concluzie, putem spune că *Dist* este o aplicație de distribuire și vizualizare a articolelor sub formă de pagini Web care încearcă pe cât posibil să ofere utilizatorilor ei articole ce prezintă interes pentru aceștia. Acest fapt se realizează în mod activ bazându-ne pe faptul că utilizatorul va ști să selecteze acele interese ce îl interesează folosindu-se de interfața aplicației client ce îi pune la dispoziție posibilitatea să caute interese pe baza mecanismului de *typeahead* (scriere înainte). Să poată viziona articole în funcție de popularitatea acestora, în funcție de gradul lor de noutate sau să poată pur și simplu să vadă articolele sale favorite. Un utilizator poate contribui la rândul său cu articole și poate să facă administrarea acestora (ștergere / editare).

Această aplicație își propune să demonstreze rezolvarea problemei vizualizării de conținut neinteresant care afectează orice aplicație Web populară bazată pe articole din zilele noastre. O parte din aceste aplicații depun eforturi să filtreze în diverse metode conținutul pus la dispoziție utilizatorului (de ex filtrarea de conținut destinat adulților), dar de multe ori comportamentul utilizatorilor este unul destul de complex de aceea este o adevărată provocare pentru filtrarea bazată pe învățarea automată să îndeplinească cu exactitate această sarcină. De multe ori metoda cea mai efektivă de a-i oferi unui utilizator conținut pe care acesta și-l dorește este să-i pui la dispoziție posibilitatea să filtreze chiar el conținutul pe care acesta dorește să-l vadă.

Mi-am propus ca prin aplicația *Dist* să ofer control deplin utilizatorului în a alege conținutul pe care dorește să-l poată vedea sub formă de interese. Articolele postate sunt legate de interesele cu care acestea au legătura la momentul publicării. Un utilizator va vizualiza articole care îl interesează deoarece îi vor fi afișate doar acele articole care au măcar un interes comun cu cele ale respectivului utilizator. De asemenea, utilizatorul este liber oricând să se răzgândească în privința a ce tipuri de articole dorește să-i se pună la dispoziție.

Consider că această abordare este una foarte efektivă și de impact dar un posibil inconvenient este acela că un utilizator își poate extinde sfera de interese doar în mod conștient și voit. Trăim într-o lume foarte dinamică în care lucrurile evoluează, se schimbă repede. Astfel de schimbări trebuie făcute în mod manual de către utilizator pentru a păstra conținutul sugerat ca

fiind la curent cu zilele noastre ceea ce s-ar putea considera un aspect destul de problematic pentru utilizator pentru ca el ar trebui să știe deja ce își dorește să vizualizeze.

Un aspect care ar putea fi îmbunătățit este acela că la momentul postării linkul către articolul ce este introdus spre a fi postat ar putea fi scanat de către o componentă care să sugereze interesele pe care utilizatorul să le aleagă ca fiind caracteristice articolului crescând astfel calitatea etichetării postărilor. Desigur utilizatorul nu ar fi obligat să folosească doar interesele rezultate în urma sugestiei ci ar avea în continuare posibilitatea să le introducă în mod manual.

Aplicația *Dist* și-a propus să fie un punct de plecare pentru abordarea propusă dar aceasta ar putea fi extinsă în foarte multe direcții. Pe de-o parte s-ar putea integra o sumedenie de componente terțe de la diverse alte aplicații (*facebook*, *twitter*, *google*, etc). De asemenea s-ar putea oferi un suport mult mai bun pentru filtrarea articolelor.

O posibilă funcționalitate suplimentară o constituie afișarea unei liste cu interese înrudite de cele pe care le are utilizatorul. De asemenea i s-ar putea sugera interese populare în rândul utilizatorilor aplicației.

Există multe provocări ce trebuie rezolvate pentru a putea ajunge la performanța ca o persoană să poată viziona ce își dorește. Fie că i se sugerează conținut sau este pus să aleagă sau își configurează dinainte niște preferințe în privința conținutului putem să observăm că nici o metodă de rezolvare nu este perfectă în sinea ei și că ar fi de preferat să încercăm să folosim metoda corespunzătoare sarcinii cu care ne confruntăm.

Bibliografie

1. ASP.NET Core <https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.1>.
2. Arhitectura n-Tier <https://stackoverflow.com/questions/312187/what-is-n-tier-architecture>.
3. Autentificare JWT cu ASP.NET Core <https://fullstackmark.com/post/13/jwt-authentication-with-aspnet-core-2-web-api-angular-5-net-core-identity-and-facebook-login>.
4. JWT <https://jwt.io/>.
5. Python <https://docs.python.org/3/index.html>.
6. Single Page Application (SPA) <https://ro-cod.appspot.com/articole/5439632586047488/aplicatiile-pe-o-singura-pagina-aka-single-page-applications>.
7. Angular <https://angular.io/docs>.
8. API Web thum.io <https://www.thum.io/documentation/api/url>.
9. EFcore <https://docs.microsoft.com/en-us/ef/core/>.
10. API Căutare Bing <https://azure.microsoft.com/en-us/services/cognitive-services/bing-web-search-api/>.
11. API Căutare Google <https://developers.google.com/custom-search/docs/overview>.