Sistem preporuka u aplikaciji NovinskiPortal

U NovinskiPortal aplikaciji koristi se jednostavan content based sistem preporuke koji radi samo za prijavljene korisnike. Sistem prati koje kategorije korisnik najviše čita, koliko su članci popularni i koliko su svježi. Na osnovu toga računa listu preporučenih članaka.

1. Podaci koje sistem koristi

Sistem koristi sljedeće tabele:

- *UserCategoryPreference*
  Za svakog korisnika pamti koliko puta je čitao članke iz određene kategorije. Bitna polja su UserId, CategoryId, ViewCount i LastViewedAt. Više redova za isti par (UserId, CategoryId) se u algoritmu grupiše i sabira.
- *UserArticleView*
  Za svakog korisnika pamti koje je članke otvorio. Bitna polja su UserId, ArticleId i ViewedAt. Koristi se za prepoznavanje već pročitanih članaka.
- *Article* i *ArticleStatistics*
  Article sadrži CategoryId, SubcategoryId, PublishedAt i Active.
  ArticleStatistics sadrži TotalViews, ukupan broj pregleda članka.

2. Logika rada algoritma

Glavna metoda je **RecommendationService.GetPersonalizedAsync(int userId, int take)**.

Koraci:

1. **Učitavanje podataka**
   - Učitaju se aktivni članci (Article) sa Category, User i Statistics.
   - Iz UserCategoryPreference se učitaju sve preferencije za korisnika.
   - Ako korisnik nema preferencija, kao preporuke se vraćaju globalno najčitaniji i najnoviji članci.
2. **Težine po kategorijama**
   - Preferencije se grupišu po CategoryId, sabira se ViewCount.
   - Iz tih vrijednosti se računa težina kategorije za korisnika (ViewCount kategorije podijeljen sa sumom svih ViewCount vrijednosti).
   - Težina govori koliko korisnik voli tu kategoriju.
3. **Već pročitani članci**
   - Iz UserArticleView se učitaju svi ArticleId koje je korisnik otvorio.
   - Ti članci ne izlaze iz preporuka, ali im se kasnije smanjuje ocjena.
4. **Izračunavanje ocjene članka**
   Za svaki aktivni članak računa se score na osnovu:
   - contentScore: koliko korisnik voli kategoriju tog članka (težina kategorije)
   - popularityScore: relativna popularnost na osnovu TotalViews

- recencyScore: svježina na osnovu datuma objave
- viewedPenalty: ako je članak već čitan, score se množi penalom (npr. 0.4)
- mali random faktor, da preporuke ne budu svaki put potpuno iste

Score je kombinacija ova tri faktora sa različitim težinama, pomnožena penalom za već pročitane članke, uz dodatak malog nasumičnog dijela.

5. **Odabir i fallback**
   - Članci se sortiraju po score opadajuće.
   - Uzima se prvih take članaka (npr. 6).
   - Ako je rezultat manji od take, lista se dopuni najčitanijim aktivnim člancima koji još nisu u rezultatu, dok se ne dosegne traženi broj ili ne ponestane članaka.

Na ovaj način korisnik najčešće dobije tačno take preporuka, koje su usklađene sa njegovim interesima, ali uz uvažavanje popularnosti i svježine sadržaja.

3. Implementacija u kodu

Servis:

- Namespace: NovinskiPortal.Services.Services.RecommendationService
- Klasa: RecommendationService
- Metoda: Task<List<ArticleResponse>> GetPersonalizedAsync(int userId, int take = 6)

Entiteti:

- UserCategoryPreference, UserArticleView, Article, ArticleStatistics u namespaceu NovinskiPortal.Services.Database.Entities.

API:

- Namespace: NovinskiPortal.API.Controllers
- Klasa: RecommendationController
- Endpoint: GET /api/recommendation/personalized?take=6

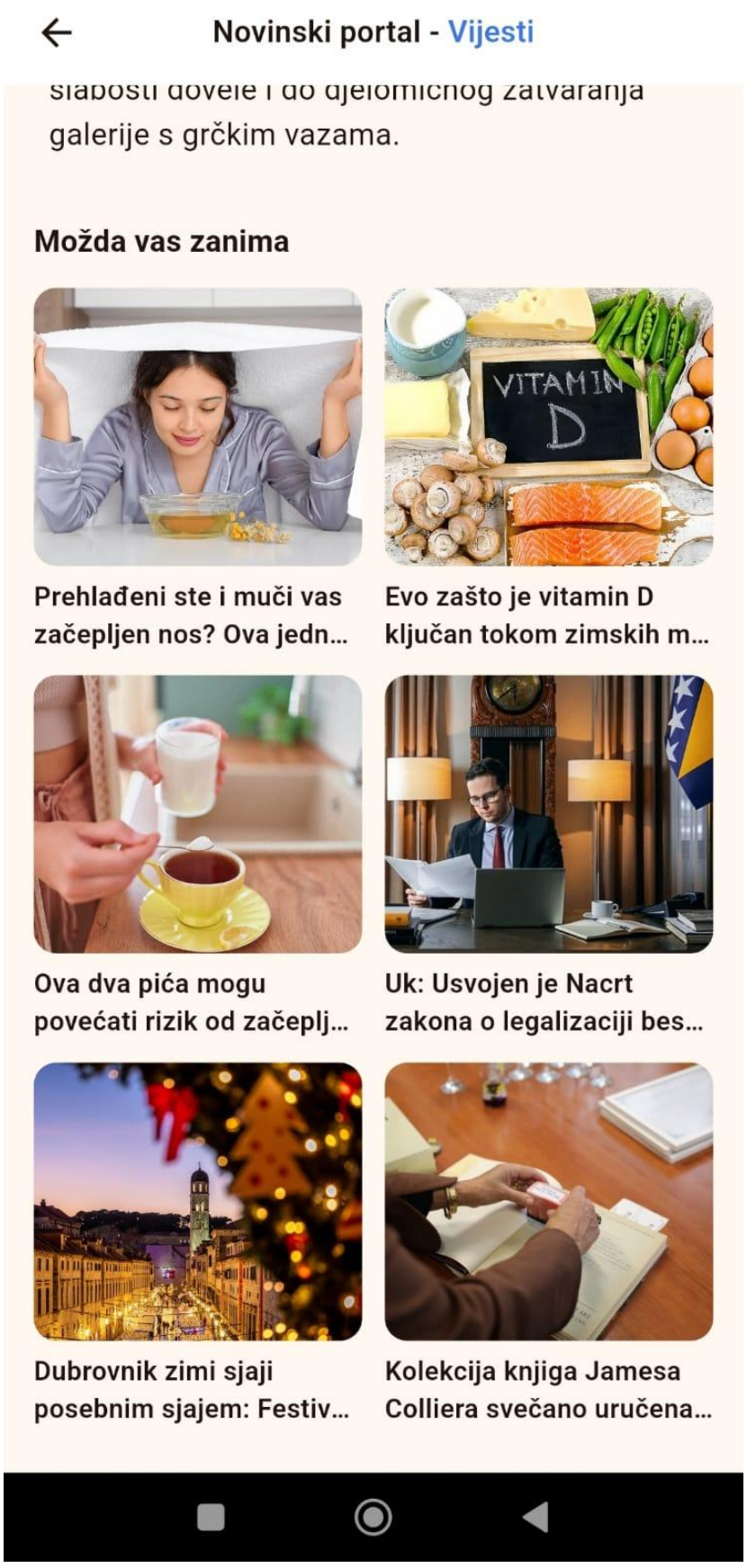
Kontroler čita userId iz JWT tokena, poziva IRecommendationService.GetPersonalizedAsync za tog korisnika i vraća listu preporučenih članaka kao List<ArticleResponse>.

```
public async Task<List<ArticleResponse>> GetPersonalizedAsync(int userId, int take = 6)
{
    var allArticles = await _context.Articles
        .Include(a => a.Category)
        .Include(a => a.Subcategory)
        .Include(a => a.User)
        .Include(a => a.Statistics)
        .Where(a => a.Active)
        .OrderByDescending(a => a.PublishedAt)
        .ThenByDescending(a => a.Statistics != null ? a.Statistics.TotalViews : 0)
        .Take(500)
        .ToListAsync();

    if (!allArticles.Any())
    {
        return new List<ArticleResponse>();
    }

    var prefsRaw = await _context.UserCategoryPreferences
        .Where(x => x.UserId == userId)
        .ToListAsync();

    if (!prefsRaw.Any())
    {
        var globalTop = allArticles
            .OrderByDescending(a => a.Statistics != null ? a.Statistics.TotalViews : 0)
            .ThenByDescending(a => a.PublishedAt)
            .Take(take)
            .ToList();

        return _mapper.Map<List<ArticleResponse>>(globalTop);
    }

    var prefs = prefsRaw
        .GroupBy(c => c.CategoryId)
        .Select(g => new
        {
            CategoryId = g.Key,
            ViewCount = g.Sum(x => x.ViewCount)
        })
        .ToList();

    var totalViews = prefs.Sum(x => x.ViewCount);
    if (totalViews <= 0)
    {
        totalViews = 1;
    }

    var categoryWeights = prefs.ToDictionary(
        x => x.CategoryId,
        x => (double)x.ViewCount / totalViews
    );
```

```
    var viewedIds = await _context.UserArticleViews
        .Where(v => v.UserId == userId)
        .Select(v => v.ArticleId)
        .Distinct()
        .ToListAsync();
    var viewedSet = new HashSet<int>(viewedIds);

    var now = DateTime.UtcNow;
    var maxViews = allArticles.Max(a => a.Statistics?.TotalViews ?? 0);
    if (maxViews <= 0)
    {
        maxViews = 1;
    }

    var rnd = new Random();

    var scoredArticles = allArticles
        .Select(a =>
        {
            categoryWeights.TryGetValue(a.CategoryId, out var categoryWeight);
            var contentScore = categoryWeight;

            var views = a.Statistics?.TotalViews ?? 0;
            var popularityScore = (double)views / maxViews;

            var daysAgo = (now - a.PublishedAt).TotalDays;
            if (daysAgo < 0)
            {
                daysAgo = 0;
            }
            var recencyScore = 1.0 / (1.0 + daysAgo / 7.0);

            var viewedPenalty = viewedSet.Contains(a.Id) ? 0.4 : 1.0;

            var noise = rnd.NextDouble() * 0.05;

            var finalScore =
                (0.6 * contentScore + 0.25 * popularityScore + 0.15 * recencyScore)
                * viewedPenalty
                + noise;

            return new
            {
                Article = a,
                Score = finalScore
            };
        })
        .OrderByDescending(x => x.Score)
        .Select(x => x.Article)
        .ToList();
```

```
var result = scoredArticles.Take(take).ToList();

if (result.Count < take)
{
    var existingIds = result.Select(a => a.Id).ToHashSet();

    var extra = allArticles
        .Where(a => !existingIds.Contains(a.Id))
        .OrderByDescending(a => a.Statistics != null ? a.Statistics.TotalViews : 0)
        .ThenByDescending(a => a.PublishedAt)
        .Take(take - result.Count)
        .ToList();

    result.AddRange(extra);
}

return _mapper.Map<List<ArticleResponse>>(result);
```

Glavna logika recommender sistema



Prikaz preporuka u aplikaciji