

## DOKUMENTACIJA SISTEMA PREPORUKE za aplikaciju SeriLovers

Predmet: **Razvoj softvera II**  
Fakultet informacijskih tehnologija

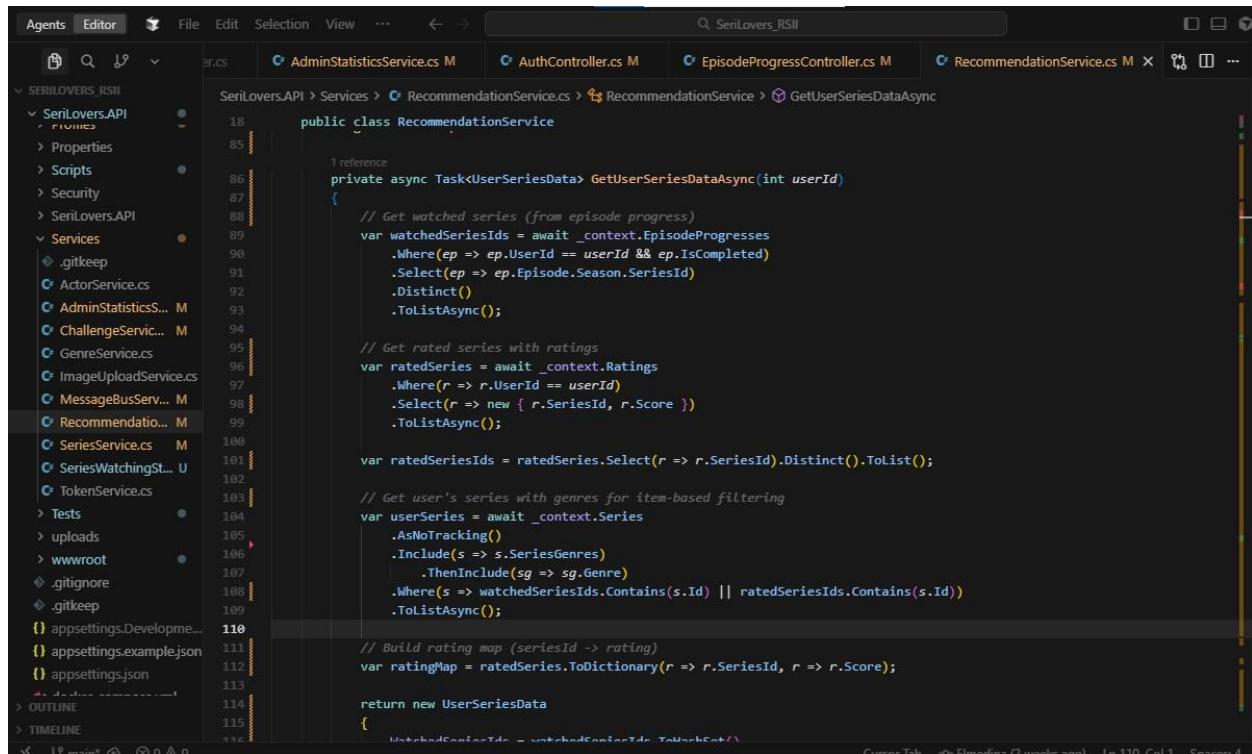
Studentica: **Elmedina Marić**  
Akademska godina: **2025/2026**

### 1. Uvod

Ovaj dokument predstavlja detaljan opis implementacije sistema preporuke u aplikaciji **SeriLovers**. Glavni cilj sistema preporuke je unaprijediti korisničko iskustvo kroz personalizovane prijedloge TV serija, prilagođene interesima i prethodnom ponašanju korisnika.

Sistem preporuke omogućava korisnicima da brže i lakše pronađu relevantan sadržaj, bez potrebe za dugotrajnim i ručnim pretraživanjem baze podataka. Time se povećava zadovoljstvo korisnika i ukupna vrijednost aplikacije.

Implementirani sistem preporuke je **hibridni recommender sistem**, koji kombinuje sadržajno filtriranje (*item-based filtering*) i kolaborativno filtriranje zasnovano na korisnicima (*user-based collaborative filtering*). Kombinovanjem ova dva pristupa ostvaruje se veća preciznost, relevantnost i pouzdanost preporuka u odnosu na korištenje samo jednog modela.



The screenshot shows the Visual Studio code editor with the file `RecommendationService.cs` open. The code implements a hybrid recommendation system using item-based filtering and user-based collaborative filtering. It includes methods for getting watched series, rated series, and user's series with genres. The code uses Entity Framework Core to interact with a database context named `_context`.

```
public class RecommendationService
{
    private readonly SeriLoversContext _context;

    public RecommendationService(SeriLoversContext context)
    {
        _context = context;
    }

    // Get watched series (from episode progress)
    private async Task<UserSeriesData> GetUserSeriesDataAsync(int userId)
    {
        var watchedSeriesIds = await _context.EpisodeProgresses
            .Where(ep => ep.UserId == userId && ep.IsCompleted)
            .Select(ep => ep.Episode.Season.SeriesId)
            .Distinct()
            .ToListAsync();

        // Get rated series with ratings
        var ratedSeries = await _context.Ratings
            .Where(r => r.UserId == userId)
            .Select(r => new { r.SeriesId, r.Score })
            .ToListAsync();

        var ratedSeriesIds = ratedSeries.Select(r => r.SeriesId).Distinct().ToList();

        // Get user's series with genres for item-based filtering
        var userSeries = await _context.Series
            .AsNoTracking()
            .Include(s => s.SeriesGenres)
            .ThenInclude(sg => sg.Genre)
            .Where(s => watchedSeriesIds.Contains(s.Id) || ratedSeriesIds.Contains(s.Id))
            .ToListAsync();

        // Build rating map (seriesId -> rating)
        var ratingMap = ratedSeries.ToDictionary(r => r.SeriesId, r => r.Score);

        return new UserSeriesData
        {
            WatchedContentIds = watchedSeriesIds,
            UnwatchedContentIds = userSeries.Where(s => !watchedSeriesIds.Contains(s.Id)).Select(s => s.Id),
            RatingMap = ratingMap
        };
    }
}
```

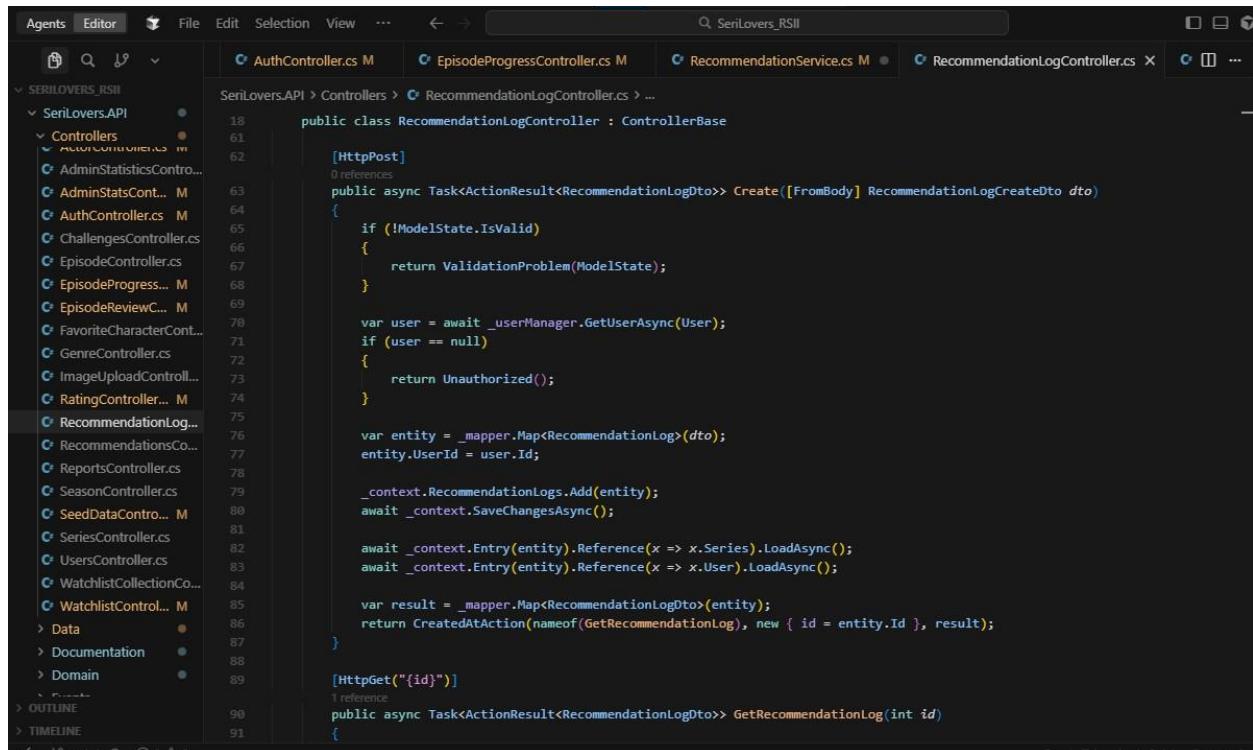
Slika 1 -SeriLovers.API/Services/RecommendationService.cs

## 2. Arhitektura sistema preporuke

Sistem preporuke je implementiran u backend dijelu aplikacije koristeći **ASP.NET Core Web API** arhitekturu. Centralni dio sistema predstavlja servis **RecommendationService**, koji sadrži kompletну poslovnu logiku za generisanje preporuka.

Kontroler **RecommendationsController** služi kao posrednik između frontend aplikacije i sistema preporuke, izlažući REST API rute koje frontend koristi za dohvati preporuka. Ovakva arhitektura omogućava jasnu separaciju odgovornosti, lakše održavanje sistema i jednostavno proširenje funkcionalnosti u budućnosti.

Frontend dio aplikacije, razvijen u **Flutter** tehnologiji, komunicira sa backendom putem HTTP zahtjeva i prikazuje rezultate sistema preporuke krajnjim korisnicima.



The screenshot shows a code editor interface with multiple tabs open at the top. The active tab is 'RecommendationLogController.cs' under the 'SeriLovers.API' project. The left sidebar shows a tree view of the project structure, including 'Controllers' and various controller files like 'AdminController.cs', 'AuthController.cs', etc. The main editor area displays the C# code for the 'RecommendationLogController'. The code includes methods for creating a new recommendation log entry and getting a specific log by ID. The code uses annotations such as [HttpPost] and [HttpGet("{id}")] to define the API endpoints. The code editor has a dark theme with syntax highlighting for C#.

```
public class RecommendationLogController : ControllerBase
{
    [HttpPost]
    public async Task<ActionResult<RecommendationLogDto>> Create([FromBody] RecommendationLogCreateDto dto)
    {
        if (!ModelState.IsValid)
        {
            return ValidationProblem(ModelState);
        }

        var user = await _userManager.GetUserAsync(User);
        if (user == null)
        {
            return Unauthorized();
        }

        var entity = _mapper.Map<RecommendationLog>(dto);
        entity.UserId = user.Id;

        _context.RecommendationLogs.Add(entity);
        await _context.SaveChangesAsync();

        await _context.Entry(entity).Reference(x => x.Series).LoadAsync();
        await _context.Entry(entity).Reference(x => x.User).LoadAsync();

        var result = _mapper.Map<RecommendationLogDto>(entity);
        return CreatedAtAction(nameof(GetRecommendationLog), new { id = entity.Id }, result);
    }

    [HttpGet("{id}")]
    public async Task<ActionResult<RecommendationLogDto>> GetRecommendationLog(int id)
    {
    }
}
```

Slika 2 - SeriLovers.API/Controllers/RecommendationsController.cs

### **3. Korišteni podaci za generisanje preporuka**

Sistem preporuke koristi više izvora podataka kako bi formirao potpun korisnički profil. Analiziraju se serije koje je korisnik gledao, serije koje je ocijenio, kao i ocjene i ponašanje drugih korisnika unutar sistema.

Posebna pažnja posvećena je žanrovima serija, jer oni predstavljaju važan indikator korisničkih preferencija. Također se uzima u obzir popularnost serija, broj ocjena i prosječna ocjena, što omogućava sistemu da doneše kvalitetnije odluke prilikom preporučivanja sadržaja.

Kombinovanjem eksplisitnih podataka (ocjene) i implicitnih podataka (gledanje sadržaja), sistem dobija realističniji uvid u interesovanja korisnika.

### **4. Item-based filtering (sadržajno filtriranje)**

Item-based filtering se zasniva na pretpostavci da će se korisniku svidjeti serije koje su slične onima koje je već gledao ili visoko ocijenio. U ovom sistemu sličnost između serija se prvenstveno određuje na osnovu žanrova.

Za svaku seriju koju je korisnik gledao ili ocijenio, izdvajaju se pripadajući žanrovi, koji se zatim ponderišu prema korisničkoj ocjeni. Više ocijenjene serije imaju veći uticaj na formiranje korisničkog profila.

Sličnost između korisnikovog profila i kandidatskih serija računa se pomoću **ponderisane Jaccard sličnosti**, čime se dobija numerički skor koji označava stepen podudaranja sadržaja. Ovaj pristup omogućava da preporuke budu tematski uskladjene sa interesima korisnika.

```
public class RecommendationService
{
    public async Task<List<SeriesRecommendationDto>> GetRecommendationsAsync(int userId, int maxResults = 10)
    {
        // Get candidate series (not watched/rated by user)
        var candidateSeries = await GetCandidateSeriesAsync(userId, userSeriesData.AllSeriesIds);

        if (candidateSeries.Count == 0)
        {
            _logger.LogInformation("No candidate series found for user {UserId}", userId);
            return await GetFallbackRecommendationsAsync(maxResults, userSeriesData.AllSeriesIds);
        }

        // Calculate item-based scores (genre similarity)
        var itemBasedScores = CalculateItemBasedScores(userSeriesData, candidateSeries);

        // Calculate user-based scores (similar users)
        var userBasedScores = await CalculateUserBasedScoresAsync(userId, userSeriesData, candidateSeries);

        // Combine scores
        var combinedRecommendations = await CombineRecommendationsAsync(
            candidateSeries,
            itemBasedScores,
            userBasedScores,
            maxResults);

        logger.LogInformation(
            "Generated {Count} recommendations for user {UserId} (Item-based: {ItemCount}, User-based: {UserCount})",
            combinedRecommendations.Count,
            userId,
            itemBasedScores.Count(s => s.Value > 0),
            userBasedScores.Count(s => s.Value > 0));

        return combinedRecommendations;
    }
}
```

Slika 3 -CalculateItemBasedScores

## 5. User-based collaborative filtering

User-based collaborative filtering se oslanja na ideju da korisnici sa sličnim ukusom u prošlosti imaju tendenciju da se slažu i u budućnosti. U ovom sistemu sličnost između korisnika se računa na osnovu njihovih ocjena i gledanog sadržaja.

Za svakog korisnika se formira vektor ocjena, pri čemu se eksplicitne ocjene normalizuju, a gledane serije tretiraju kao implicitne pozitivne interakcije. Na osnovu tih vektora računa se sličnost između korisnika koristeći **kosinusnu sličnost**.

Korisnici sa najvećim stepenom sličnosti smatraju se relevantnim, a njihov ukus se koristi za preporuku novih serija. Na ovaj način sistem koristi kolektivno znanje zajednice korisnika.

The screenshot shows the Visual Studio IDE interface. The title bar says "Agents Editor File Edit Selection View ... Q SeriLovers\_RSII". The left sidebar shows a file tree for "SERILOVERS\_RSII" with "SeriLovers.API" selected, containing "Properties", "Scripts", "Security", "SeriLovers.API", "Services", ".gitkeep", "ActorService.cs", "AdminStatisticsS... M", "ChallengeServic... M", "GenreService.cs", "ImageUploadService.cs", "MessageBusServ... M", "Recommendatio... M", "SeriesService.cs M", "SeriesWatchingSt... U", "TokenService.cs", "Tests", "uploads", "wwwroot", ".gitignore", ".gitkeep", "appsettings.Developme...", "appsettings.example.json", and "appsettings.json". The right pane displays the "RecommendationService.cs" code. The code is a C# file with several imports at the top. The main method shown is "FindSimilarUsersAsync". It starts by creating a dictionary "otherUserVector" and populating it with user ratings. It then iterates through "otherUserWatched" series and adds them to the vector if they don't already exist. The similarity is calculated using the cosine formula. If the similarity is greater than 0.1, a new "SimilarUser" object is created with the user ID, similarity score, and the other user's rating vector. Finally, the users are sorted by similarity and the top 20 are returned. A note at the bottom indicates that the results are sorted by similarity and take the top 20 most similar users.

```
public class RecommendationService
{
    private async Task<List<SimilarUser>> FindSimilarUsersAsync()
    {
        .Distinct()
        .ToListAsync();

        var otherUserVector = new Dictionary<int, double>();
        foreach (var rating in otherUserRatings)
        {
            otherUserVector[rating.SeriesId] = (rating.Score - 1) / 9.0;
        }
        foreach (var seriesId in otherUserWatched)
        {
            if (!otherUserVector.ContainsKey(seriesId))
            {
                otherUserVector[seriesId] = 0.5;
            }
        }

        // Calculate cosine similarity
        var similarity = CalculateCosineSimilarity(currentUserVector, otherUserVector);

        if (similarity > 0.1)
        {
            similarUsers.Add(new SimilarUser
            {
                UserId = otherUserId,
                Similarity = similarity,
                RatingVector = otherUserVector
            });
        }
    }

    // Sort by similarity and take top 20 most similar users
    return similarUsers;
}
```

Slika 4 - CalculateCosineSimilarity

## 6. Kombinovanje rezultata (hibridni pristup)

Kako bi se postigla optimalna ravnoteža između sadržajne sličnosti i ponašanja drugih korisnika, rezultati item-based i user-based filtriranja se kombinuju u jedinstveni skor.

U implementaciji je primijenjen ponderisani pristup, gdje item-based filtering ima težinu od **60%**, dok user-based filtering učestvuje sa **40%** u konačnom rezultatu. Ovakva raspodjela omogućava da se korisničke lične preferencije blago favorizuju u odnosu na kolektivne preporuke.

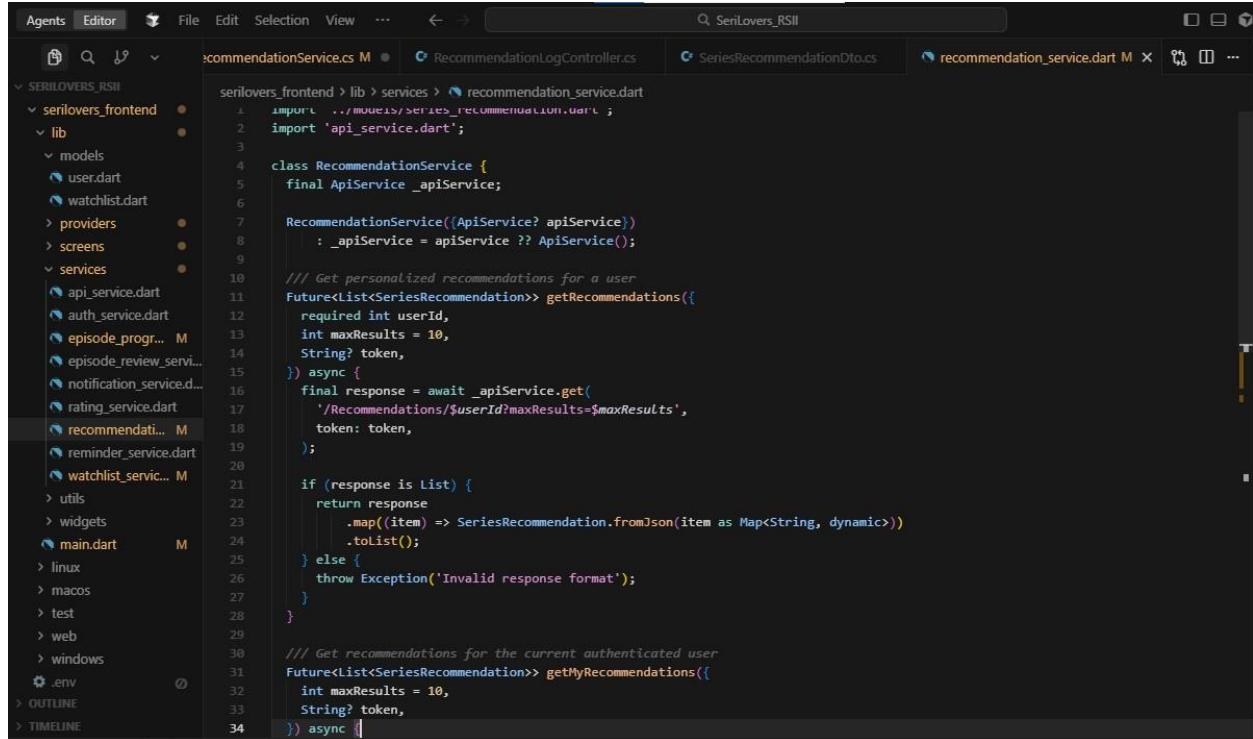
Finalni skor se koristi za rangiranje serija, nakon čega se korisniku prikazuju najrelevantnije preporuke.

## 7. Prikaz preporuka u aplikaciji

Preporuke se prikazuju u frontend dijelu aplikacije putem posebnog servisa RecommendationService. Frontend šalje zahtjev prema API ruti /api/Recommendations/me, nakon čega backend vraća listu preporučenih serija.

Svaka preporuka sadrži dodatne informacije, uključujući naziv serije, listu žanrova, prosječnu ocjenu i razlog preporuke. Prikaz razloga preporuke povećava transparentnost sistema i pomaže korisniku da razumije zašto je određena serija predložena.

Na ovaj način sistem preporuke ne djeluje kao „crna kutija“, već pruža objašnjenja za svoje odluke.



```
Agents Editor File Edit Selection View ... ← → Q SeriLovers_RSII
recommendationService.cs M RecommendationLogController.cs SeriesRecommendationDto.cs recommendation_service.dart M ...
SERILOVERS_RSII
serilovers_frontend lib
models user.dart watchlist.dart
providers screens
services api_service.dart auth_service.dart episode_program... episode_review_servi... notification_service.d... rating_service.dart recommendation... reminder_service.dart watchlist_servi... utils widgets main.dart
linux macos test web windows .env
OUTLINE TIMELINE
serilovers_frontend > lib > services > recommendation_service.dart
1 import '../../models/series_recommendation.dart';
2 import 'api_service.dart';
3
4 class RecommendationService {
5   final ApiService _apiService;
6
7   RecommendationService({ApiService? apiService})
8     : _apiService = apiService ?? ApiService();
9
10  /// Get personalized recommendations for a user
11  Future<List<SeriesRecommendation>> getRecommendations({
12    required int userId,
13    int maxResults = 10,
14    String? token,
15  }) async {
16    final response = await _apiService.get(
17      '/Recommendations/$userId?maxResults=$maxResults',
18      token: token,
19    );
20
21    if (response is List) {
22      return response
23        .map((item) => SeriesRecommendation.fromJson(item as Map<String, dynamic>))
24        .toList();
25    } else {
26      throw Exception('Invalid response format');
27    }
28  }
29
30  /// Get recommendations for the current authenticated user
31  Future<List<SeriesRecommendation>> getMyRecommendations({
32    int maxResults = 10,
33    String? token,
34  }) async {
```

*Slika 5 - lib/services/recommendation\_service.dart*

## 8. Zaključak

Implementirani sistem preporuke u aplikaciji SeriLovers predstavlja robustno i fleksibilno rješenje za personalizaciju sadržaja. Korištenjem hibridnog pristupa postiže se veća tačnost preporuka i bolje korisničko iskustvo.

Sistem je dizajniran tako da se može lako proširivati dodavanjem novih faktora, kao što su glumci, režiseri ili vremenski trendovi, što ga čini pogodnim za dalji razvoj i unapređenje aplikacije.