Dokumentacija Recommender Sistema - iCinema

1. Opis Implementacije

iCinema koristi hibridni recommender sistem koji kombinira **Content-Based Filtering** i **Collaborative Filtering** pristupe za preporuke filmova korisnicima.

1.1. Tipovi Preporuka

Sistem implementira dva tipa preporuka:

1. **Korisničke Preporuke (User Recommendations)**
   - Personalizovane preporuke bazirane na korisnikovoj istoriji
   - Kombinuje ocjene (ratings) i rezervacije korisnika
   - Uzima u obzir popularnost, svježinu i dostupnost filmova
2. **Slični Filmovi (Similar Movies)**
   - Content-based preporuke bazirane na sličnosti filmova
   - Koristi žanrove, glumce i režisere za izračunavanje sličnosti

1.2. Algoritam

Content-Based Filtering

Svaki film se predstavlja kao **vektor karakteristika** (item vector) koji uključuje: - **Žanrove** (težina: 1.0) - **Glumce** (težina: 0.5) - **Režisera** (težina: 0.8)

Vektori se normalizuju koristeći L2 normu.

User Profile

Korisnički profil se gradi na osnovu: - **Ocjena (Ratings)**: Težina se računa kao `max(0.2, min(1.0, rating/5.0))` - **Rezervacija**: Težina se računa kao `0.6 * exp(-days/90)` gdje se primjenjuje eksponencijalni decay

Scoring Funkcija

Finalni score za preporuku se računa kao:

```
score = 1.0 * cosine_similarity + popularity_boost + freshness_boost + availa
bility_boost
```

Gdje: - **cosine_similarity**: Kosinusna sličnost između korisničkog profila i film vektora - **popularity_boost**: `0.1 * tanh(popularity/10)` - bazirano na rezervacijama u posljednjih 60 dana - **freshness_boost**: `0.05 * (1 - age/365)` - bonus za nove filmove (mlađe od godinu dana) - **availability_boost**: `0.05 + 0.03 * projection_count` - bonus za filmove dostupne u preferiranom kinu

Similar Movies

Za slične filmove, sistem koristi kosinusnu sličnost između item vektora:

```
similarity = cosine_similarity(movie1_vector, movie2_vector)
```

---

2. Putanja Glavne Logike

Glavna logika recommender sistema se nalazi u:

Backend: - `iCinema.Infrastructure/Persistence/Repositories/RecommendationRepository.cs` - Metoda `GetUserRecommendations()` - linije 18-101 - Metoda `GetSimilarMovies()` - linije 103-130 - Helper metode: - `BuildItemVector()` - linije 134-160 - `Cosine()` - linije 180-192 - `Normalize()` - linije 171-178

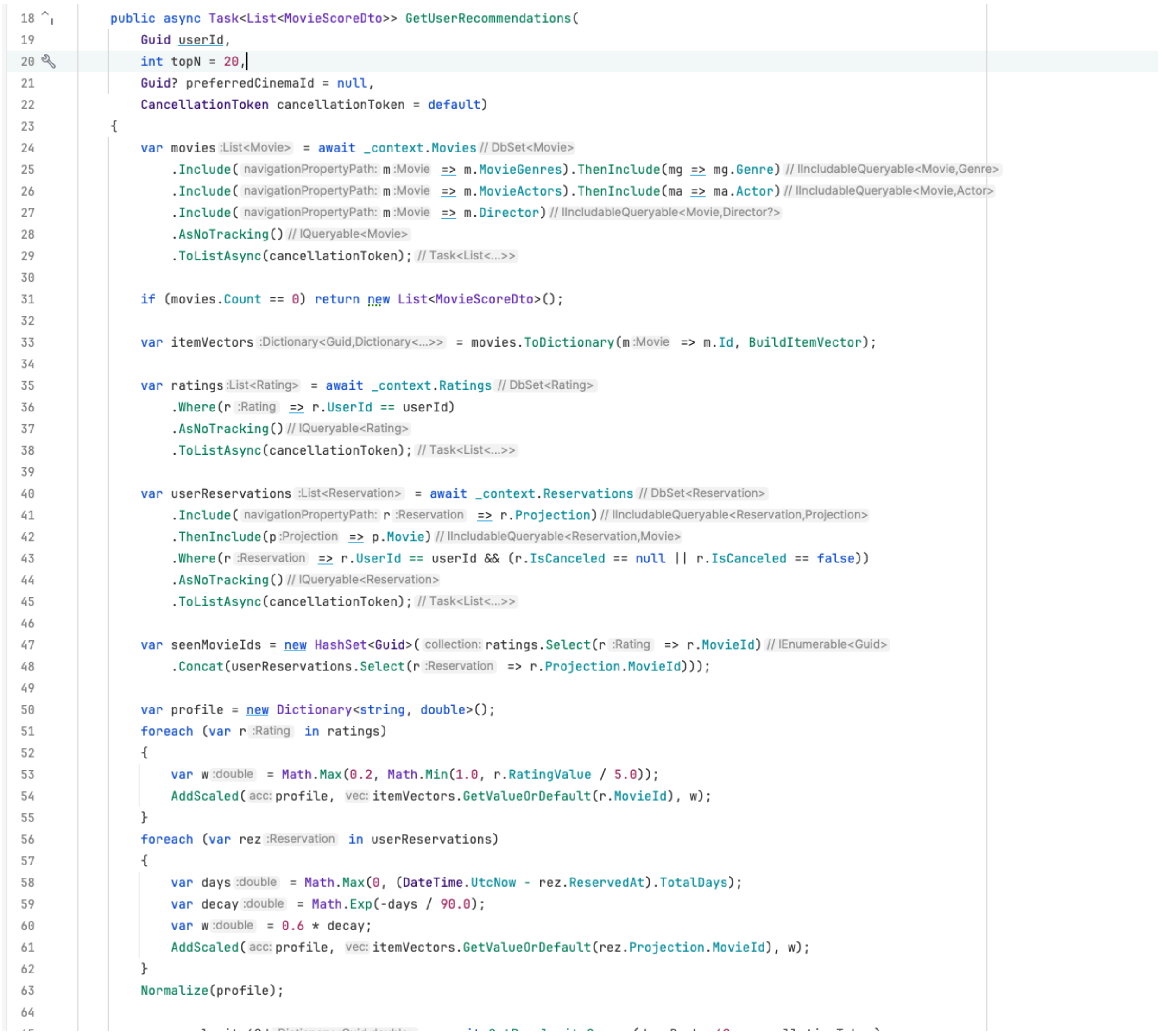
API Endpoint: - `iCinema.Api/Controllers/RecommendationsController.cs` - `GET /recommendations/my` - linije 14-25 - `GET /recommendations/similar/{movieId}` - linije 28-35

Frontend (Mobile): - `iCinema.UI/icinema_mobile_client/lib/features/home/presentation/pages/home_page.dart` - Metoda `_buildRecommendations()` - linija 516 - `iCinema.UI/icinema_mobile_client/lib/features/movies/presentation/pages/movie_details_page.dart` - Metoda `_buildSimilarMoviesSection()` - linija 212

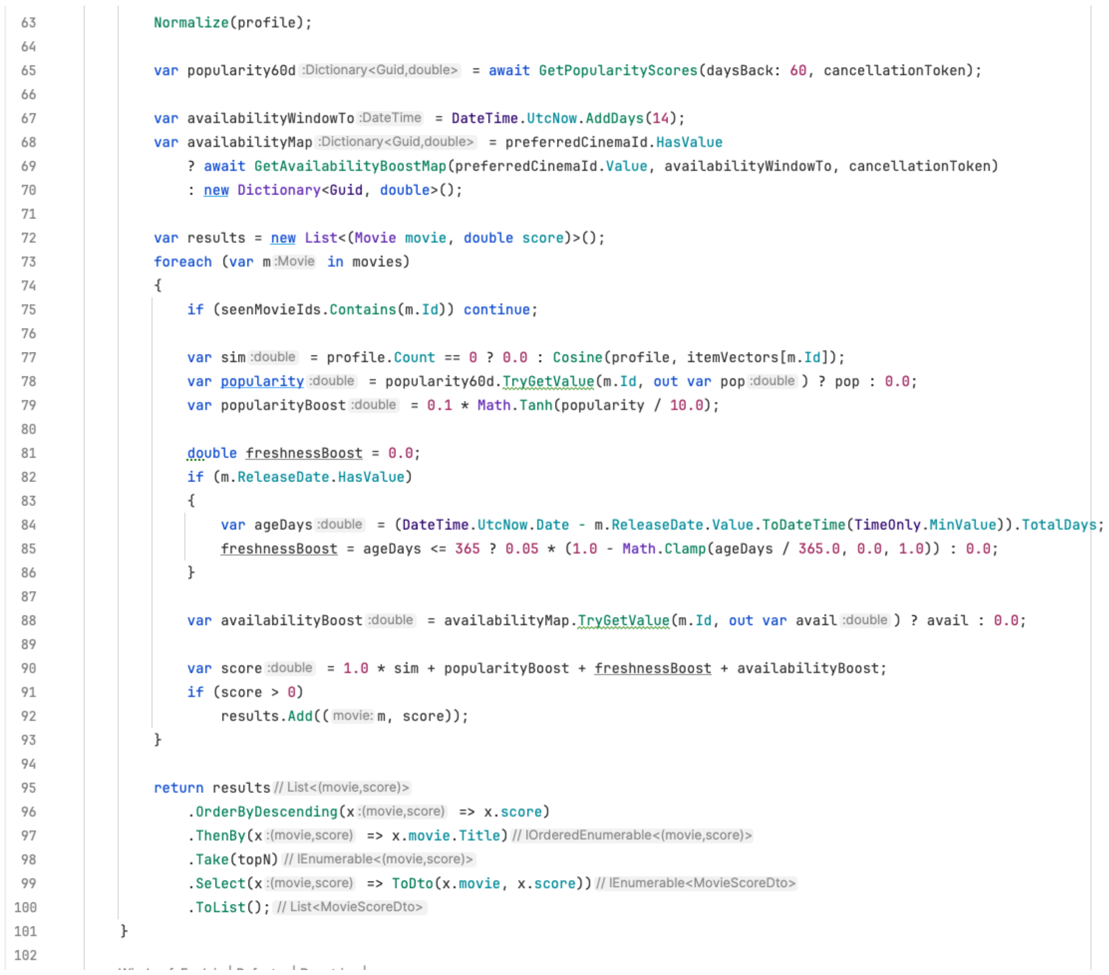
---

3. Screenshot Source Code-a

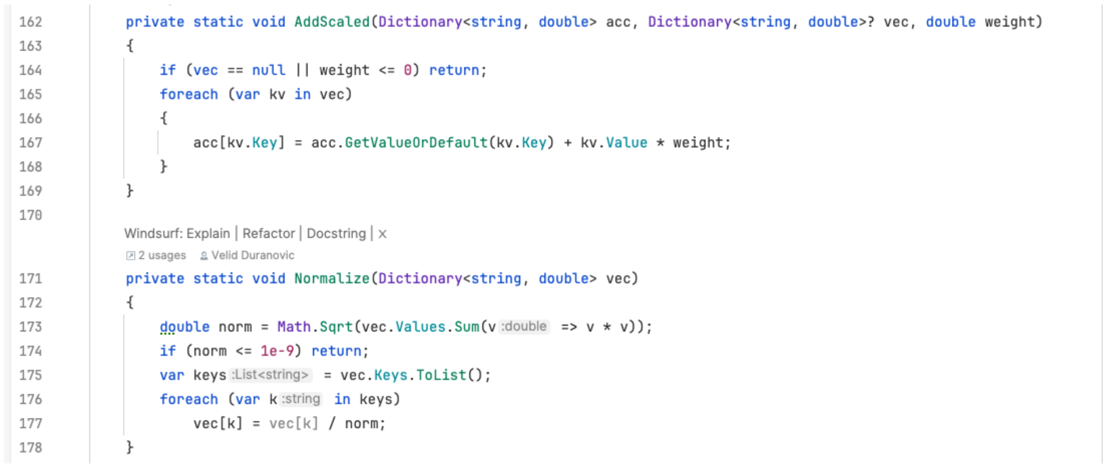
3.1. Glavna Metoda – GetUserRecommendations



```
public async Task<List<MovieScoreDto>> GetUserRecommendations(
    Guid userId,
    int topN = 20,
    Guid? preferredCinemaId = null,
    CancellationToken cancellationToken = default)
{
    var movies :List<Movie>  = await _context.Movies // DbSet<Movie>
        .Include( navigationPropertyPath: m :Movie  => m.MovieGenres).ThenInclude(mg => mg.Genre) // IIncludableQueryable<Movie,Genre>
        .Include( navigationPropertyPath: m :Movie  => m.MovieActors).ThenInclude(ma => ma.Actor) // IIncludableQueryable<Movie,Actor>
        .Include( navigationPropertyPath: m :Movie  => m.Director) // IIncludableQueryable<Movie,Director?>
        .AsNoTracking() // IQueryable<Movie>
        .ToListAsync(cancellationToken); // Task<List<...>>

    if (movies.Count == 0) return new List<MovieScoreDto>();

    var itemVectors :Dictionary<Guid,Dictionary<...>>  = movies.ToDictionary(m :Movie  => m.Id, BuildItemVector);

    var ratings :List<Rating>  = await _context.Ratings // DbSet<Rating>
        .Where(r :Rating  => r.UserId == userId)
        .AsNoTracking() // IQueryable<Rating>
        .ToListAsync(cancellationToken); // Task<List<...>>

    var userReservations :List<Reservation>  = await _context.Reservations // DbSet<Reservation>
        .Include( navigationPropertyPath: r :Reservation  => r.Projection) // IIncludableQueryable<Reservation,Projection>
        .ThenInclude(p :Projection  => p.Movie) // IIncludableQueryable<Reservation,Movie>
        .Where(r :Reservation  => r.UserId == userId && (r.IsCanceled == null || r.IsCanceled == false))
        .AsNoTracking() // IQueryable<Reservation>
        .ToListAsync(cancellationToken); // Task<List<...>>

    var seenMovieIds = new HashSet<Guid>( collection: ratings.Select(r :Rating  => r.MovieId) // IEnumerable<Guid>
        .Concat(userReservations.Select(r :Reservation  => r.Projection.MovieId)));

    var profile = new Dictionary<string, double>();
    foreach (var r :Rating  in ratings)
    {
        var w :double  = Math.Max(0.2, Math.Min(1.0, r.RatingValue / 5.0));
        AddScaled( acc: profile,  vec: itemVectors.GetValueOrDefault(r.MovieId), w);
    }
    foreach (var rez :Reservation  in userReservations)
    {
        var days :double  = Math.Max(0, (DateTime.UtcNow - rez.ReservedAt).TotalDays);
        var decay :double  = Math.Exp(-days / 90.0);
        var w :double  = 0.6 * decay;
        AddScaled( acc: profile,  vec: itemVectors.GetValueOrDefault(rez.Projection.MovieId), w);
    }
    Normalize(profile);

```

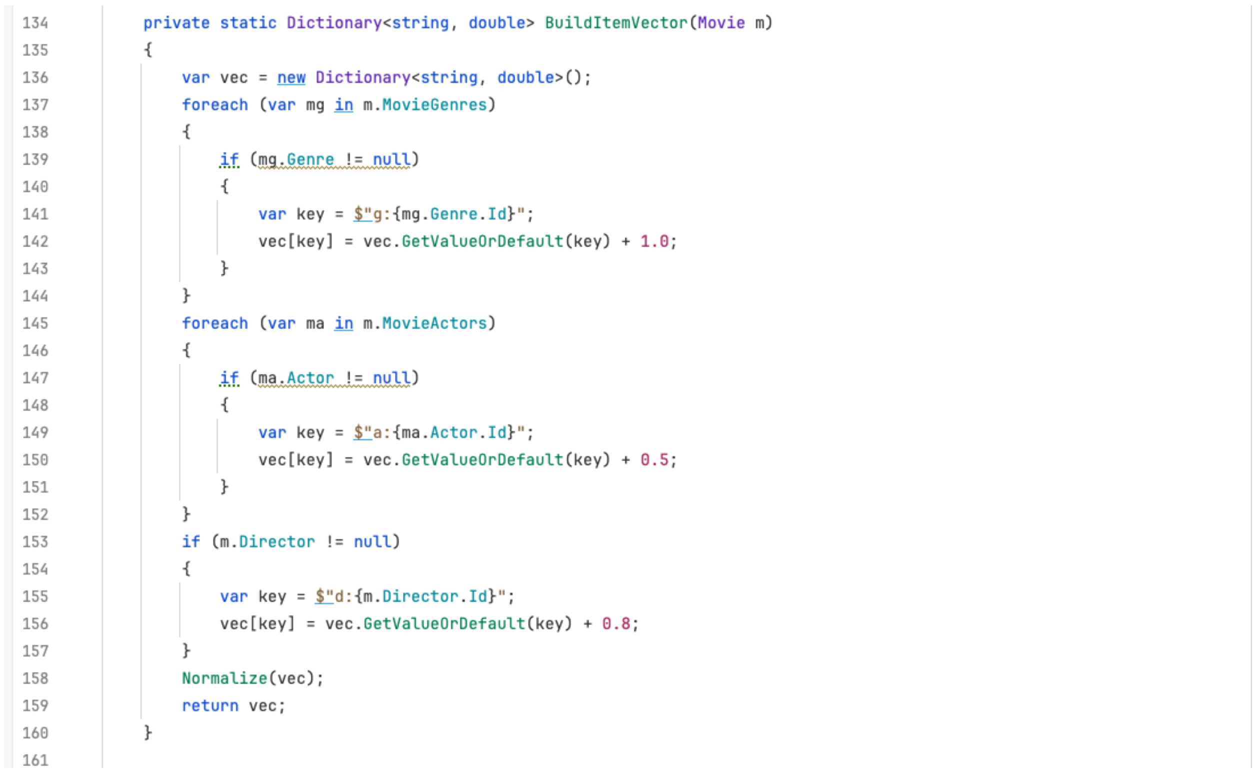


```
    Normalize(profile);

    var popularity60d :Dictionary<Guid,double> = await GetPopularityScores(daysBack: 60, cancellationToken);

    var availabilityWindowTo :DateTime = DateTime.UtcNow.AddDays(14);
    var availabilityMap :Dictionary<Guid,double> = preferredCinemaId.HasValue
        ? await GetAvailabilityBoostMap(preferredCinemaId.Value, availabilityWindowTo, cancellationToken)
        : new Dictionary<Guid, double>();

    var results = new List<(Movie movie, double score)>();
    foreach (var m :Movie in movies)
    {
        if (seenMovieIds.Contains(m.Id)) continue;

        var sim :double = profile.Count == 0 ? 0.0 : Cosine(profile, itemVectors[m.Id]);
        var popularity :double = popularity60d.TryGetValue(m.Id, out var pop :double ) ? pop : 0.0;
        var popularityBoost :double = 0.1 * Math.Tanh(popularity / 10.0);

        double freshnessBoost = 0.0;
        if (m.ReleaseDate.HasValue)
        {
            var ageDays :double = (DateTime.UtcNow.Date - m.ReleaseDate.Value.ToDateTime(TimeOnly.MinValue)).TotalDays;
            freshnessBoost = ageDays <= 365 ? 0.05 * (1.0 - Math.Clamp(ageDays / 365.0, 0.0, 1.0)) : 0.0;
        }

        var availabilityBoost :double = availabilityMap.TryGetValue(m.Id, out var avail :double ) ? avail : 0.0;

        var score :double = 1.0 * sim + popularityBoost + freshnessBoost + availabilityBoost;
        if (score > 0)
            results.Add(( movie: m, score));
    }

    return results // List<(movie,score)>
        .OrderByDescending(x :(movie,score) => x.score)
        .ThenBy(x :(movie,score) => x.movie.Title) // IOrderedEnumerable<(movie,score)>
        .Take(topN) // IEnumerable<(movie,score)>
        .Select(x :(movie,score) => ToDto(x.movie, x.score)) // IEnumerable<MovieScoreDto>
        .ToList(); // List<MovieScoreDto>
}

```



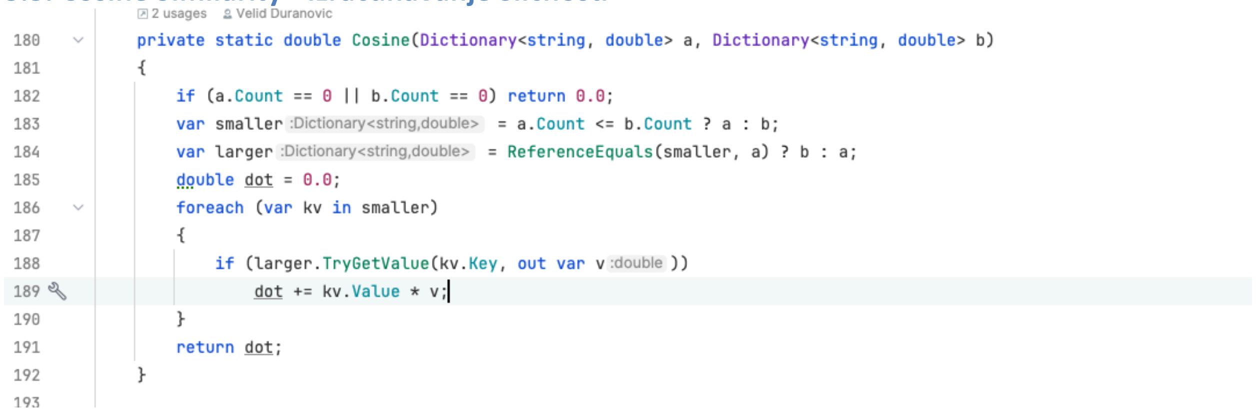
```
private static void AddScaled(Dictionary<string, double> acc, Dictionary<string, double>? vec, double weight)
{
    if (vec == null || weight <= 0) return;
    foreach (var kv in vec)
    {
        acc[kv.Key] = acc.GetValueOrDefault(kv.Key) + kv.Value * weight;
    }
}

Windsurf: Explain | Refactor | Docstring | ×
2 usages   Velid Duranovic
private static void Normalize(Dictionary<string, double> vec)
{
    double norm = Math.Sqrt(vec.Values.Sum(v :double => v * v));
    if (norm <= 1e-9) return;
    var keys :List<string> = vec.Keys.ToList();
    foreach (var k :string in keys)
        vec[k] = vec[k] / norm;
}
```

3.2. BuildItemVector - Kreiranje Vektora Karakteristika



```
private static Dictionary<string, double> BuildItemVector(Movie m)
{
    var vec = new Dictionary<string, double>();
    foreach (var mg in m.MovieGenres)
    {
        if (mg.Genre != null)
        {
            var key = $"g:{mg.Genre.Id}";
            vec[key] = vec.GetValueOrDefault(key) + 1.0;
        }
    }
    foreach (var ma in m.MovieActors)
    {
        if (ma.Actor != null)
        {
            var key = $"a:{ma.Actor.Id}";
            vec[key] = vec.GetValueOrDefault(key) + 0.5;
        }
    }
    if (m.Director != null)
    {
        var key = $"d:{m.Director.Id}";
        vec[key] = vec.GetValueOrDefault(key) + 0.8;
    }
    Normalize(vec);
    return vec;
}
```

3.3. Cosine Similarity - Izračunavanje Sličnosti



```
private static double Cosine(Dictionary<string, double> a, Dictionary<string, double> b)
{
    if (a.Count == 0 || b.Count == 0) return 0.0;
    var smaller :Dictionary<string,double> = a.Count <= b.Count ? a : b;
    var larger :Dictionary<string,double> = ReferenceEquals(smaller, a) ? b : a;
    double dot = 0.0;
    foreach (var kv in smaller)
    {
        if (larger.TryGetValue(kv.Key, out var v :double ))
            dot += kv.Value * v;
    }
    return dot;
}
```

---

4. Putanja i Screenshot iz Pokrenute Aplikacije

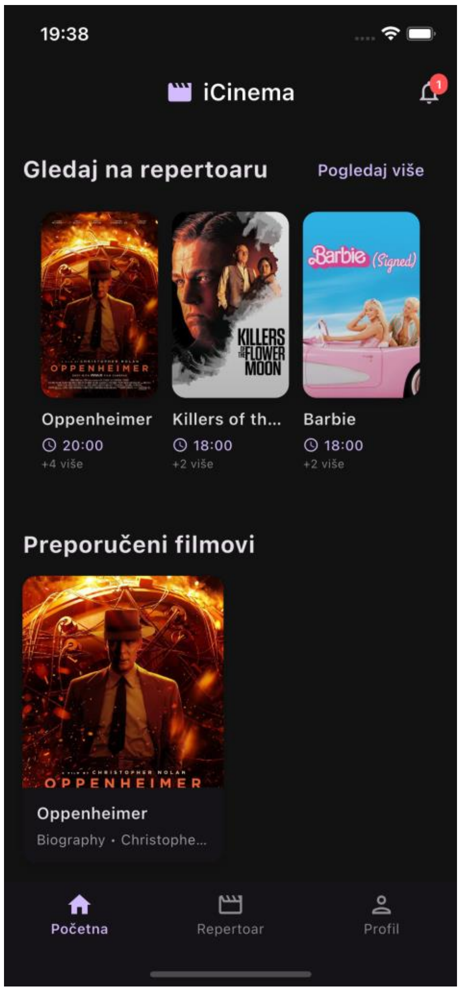
4.1. Preporučeni Filmovi (Home Page)

Putanja u aplikaciji: - Home Page → Sekcija “Preporučeni filmovi”

Putanja u kodu: - `iCinema.UI/icinema_mobile_client/lib/features/home/presentation/pages/home_page.dart` - Metoda `_buildRecommendations()` - linija 516

API Endpoint: - `GET /recommendations/my` - Zahtijeva autentifikaciju (Bearer token)

Screenshot:



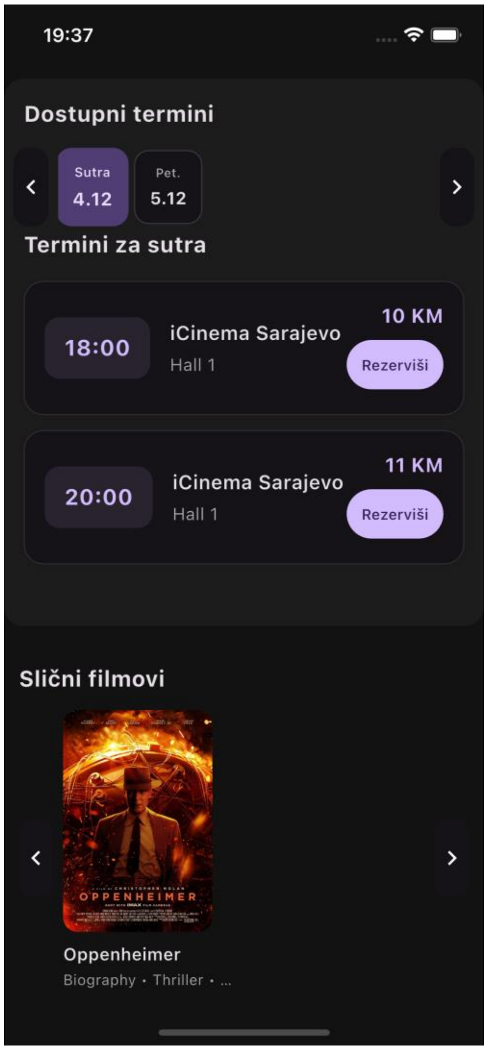
4.2. Slični Filmovi (Movie Details Page)

Putanja u aplikaciji: - Movie Details Page → Sekcija "Slični filmovi"

Putanja u kodu: - `iCinema.UI/icinema_mobile_client/lib/features/movies/presentation/pages/movie_details_page.dart` - Metoda `_buildSimilarMoviesSection()` - linija 212

API Endpoint: - `GET /recommendations/similar/{movieId}?top=20` - Ne zahtijeva autentifikaciju (AllowAnonymous)

Screenshot:



---

5. Tehnički Detalji

5.1. Performanse

- Sistem koristi **AsNoTracking()** za Entity Framework upite radi boljih performansi
- Item vektori se kreiraju jednom i koriste za sve filmove
- Popularnost se računa za posljednjih 60 dana
- Dostupnost se provjerava za sljedećih 14 dana

5.2. Filtri

- Filmovi koje je korisnik već ocjenio ili rezervisao se **isključuju** iz preporuka
- Samo filmovi sa score > 0 se vraćaju
- Rezultati se sortiraju po score-u (opadajuće), zatim po naslovu

5.3. Parametri

- **topN**: Broj preporuka (default: 20)
- **preferredCinemaId**: Opcioni ID preferiranog kina za availability boost
- **daysBack**: Period za izračunavanje popularnosti (default: 60 dana)

---

6. Zaključak

iCinema recommender sistem koristi hibridni pristup koji kombinuje: - **Content-Based Filtering** za slične filmove - **Collaborative Filtering** za personalizovane preporuke - **Popularity-based** i **Freshness-based** boost-ove za bolje rezultate - **Availability-based** boost za filmove dostupne u preferiranom kinu

Sistem je optimizovan za performanse i skalabilnost, koristeći efikasne algoritme i caching strategije.