SISTEM PREPORUKE – IB200100

U sklopu aplikacije **EventBa** razvijen je sistem preporuke koji korisnicima predlaže događaje koji bi im mogli biti zanimljivi, s ciljem unapređenja korisničkog iskustva i lakšeg pronalaska relevantnih sadržaja.

Sistem preporuke koristi hibridni pristup koji kombinuje **collaborative filtriranje**, **content-based filtriranje** i **Matrix Factorization** algoritam. Ova kombinacija omogućava preciznije i relevantnije predlaganje događaja na osnovu ponašanja i interesovanja korisnika.

Za potrebe generisanja preporuka koristi se biblioteka **ML.NET**, pri čemu se primjenjuje **Matrix Factorization** algoritam koji otkriva povezanosti između događaja na osnovu korisničkih interakcija.

Model se trenira koristeći podatke o:

- događajima koje je korisnik označio kao „sviđa mi se“,
- događajima kojima je korisnik prisustvovao (kupovina karata),
- kategorijama događaja koje je korisnik označio kao interesne.

Podaci za preporuke prikupljaju se iz više izvora i kombinuju se uz različite težine. Događaji kojima je korisnik prisustvovao i događaji koje je označio kao “sviđa mi se” imaju najveći uticaj na preporuke, dok događaji iz istih kategorija i iz korisnikovih interesnih kategorija imaju nešto manju težinu. Na osnovu ovih podataka trenira se model koji se koristi za predviđanje relevantnosti događaja.

Kada korisnik zatraži preporuke, sistem filtrira događaje kojima je korisnik već prisustvovao ili koje je već označio kao “sviđa mi se”, te za preostale događaje izračunava njihovu relevantnost. Kao rezultat, korisniku se prikazuje lista najrelevantnijih preporučenih događaja. U slučaju da ne postoji dovoljno podataka za korištenje modela, sistem koristi preporuke zasnovane isključivo na interesnim kategorijama korisnika. Generisane preporuke se čuvaju u bazi podataka radi bržeg prikazivanja prilikom narednih zahtjeva.

Glavna logika sistema preporuke implementirana je u servisu:
EventBa.Services/Services/RecommendedEventService.cs

```
1+2 usages  almedinaheric *
public void TrainModel()
{
    lock (_lock)
    {
        if (_ml != null && _model != null)
            return;

        _ml ??= new MLContext();

        var trainingData:List<EventEntry> = CollectTrainingData();
        if (trainingData.Count < 2)
        {
            Console.WriteLine("Insufficient training data. Skipping model training.");
            return;
        }

        var mappedData:List<EventEntry> = MapEventsToIndices(trainingData);
        if (mappedData.Count < 2)
            return;

        var dataView = _ml.Data.LoadFromEnumerable(mappedData);
        var trainer = _ml.Recommendation().Trainers.MatrixFactorization(CreateTrainerOptions());

        _model = trainer.Fit(dataView);

        Console.WriteLine($"Model trained with {mappedData.Count} entries.");
    }
}
```

```
public async Task<List<EventResponseDto>> GetRecommendedEventsForUser(Guid userId)
{
    try
    {
        var user = await LoadUserWithRelations(userId);
        if (user == null)
            return [];

        var cachedRecommendations:List<Event> = await LoadCachedRecommendations(userId);
        if (cachedRecommendations.Any())
            return _mapper.Map<List<EventResponseDto>>(cachedRecommendations);

        var recommendations:List<Event> = await GenerateRecommendations(user, userId);
        return _mapper.Map<List<EventResponseDto>>(recommendations);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error in GetRecommendedEventsForUser for user {userId}: {ex.Message}");

        var fallbackCached:List<Event> = await LoadCachedRecommendations(userId);
        if (fallbackCached.Any())
        {
            Console.WriteLine($"Returning cached recommendations as fallback for user {userId}");
            return _mapper.Map<List<EventResponseDto>>(fallbackCached);
        }

        return [];
    }
}

private async Task<List<Event>> GenerateRecommendations(User user, Guid userId)
{
    var excludedEventIds:List<Guid> = GetExcludedEventIds(user);
    var relevantCategoryIds:List<Guid> = GetRelevantCategoryIds(user);

    if (!relevantCategoryIds.Any())
        return [];

    var candidateEvents:List<Event> = await LoadCandidateEvents(userId, relevantCategoryIds, excludedEventIds);
    if (!candidateEvents.Any())
        return [];

    var recommendedEvents:List<Event> = IsModelAvailable()
        ? await RankEventsWithMl(user, candidateEvents, excludedEventIds)
        : candidateEvents.Take(MaxRecommendations).ToList();

    if (recommendedEvents.Any())
    {
        await CacheRecommendationsSafely(userId, recommendedEvents);
    }

    return recommendedEvents;
}
```

```
private async Task<List<Event>> RankEventsWithML(
    User user,
    List<Event> candidates,
    List<Guid> excludedIds)
{
    var seedEvents:List<Event> = GetSeedEvents(user);
    var categoryMultipliers = BuildCategoryMultipliers(user);

    var predictionEngine = _ml!.Model.CreatePredictionEngine<EventEntry, CoEventPrediction>(_model!);
    var scoredEvents = new List<(Event Event, float Score)>();

    foreach (var candidate:Event in candidates)
    {
        var mlScore:float = CalculateMlScore(seedEvents, candidate, predictionEngine);
        var contentMultiplier:float = GetContentMultiplier(candidate, categoryMultipliers);
        var finalScore:float = mlScore * contentMultiplier;

        scoredEvents.Add((Event:candidate, finalScore));
    }

    return scoredEvents// List<(Event,Score)>
        .OrderByDescending(x:(Event,Score) => x.Score)// IOrderedEnumerable<(Event,Score)>
        .Take(MaxRecommendations)// IEnumerable<(Event,Score)>
        .Select(x:(Event,Score) => x.Event)// IEnumerable<Event>
        .ToList();// List<Event>
}
```

Preporučeni događaji se prikazuju unutar aplikacije EventBa na korisničkom interfejsu u sekciji za preporuke.
eventba_mobile/lib/screens/home_screen.dart