Dokumentacija Sistema Preporuke za ITapply Platformu

Opis Implementacije Recommender Sistema

Recommender System na ITapply platformi je dizajniran da kandidatima pruži personalizirane preporuke za oglase za posao. Cilj sistema je povećati relevantnost prikazanih oglasa i poboljšati korisničko iskustvo tako što će predložiti poslove za koje postoji najveća vjerovatnoća da će odgovarati profilu i interesima kandidata.

Implementacija koristi **hibridni pristup**, kombinujući dvije tehnike preporučivanja:

1. **Collaborative Filtering:**
   Ova tehnika se bazira na analizi ponašanja korisnika kako bi otkrila skrivene veze između oglasa. Sistem koristi model faktorizacije matrice koji pregleda cjelokupnu historiju prijava svih kandidata. Na osnovu tih podataka, model uči koji oglasi se često pojavljuju zajedno u prijavama. Ako se kandidati koji su se prijavili na Oglas A također često prijavljuju i na Oglas B, sistem zaključuje da postoji jaka veza između ta dva oglasa. Zbog toga će kandidatima koji su se prijavili na Oglas A preporučiti i Oglas B, predviđajući da bi im mogao biti jednako relevantan.
2. **Content-Based Filtering:**
   Ova tehnika preporučuje oglase na osnovu njihovih specifičnih karakteristika. U ovom slučaju, sistem analizira vještine (skills) navedene u oglasima na koje se kandidat ranije prijavio. Zatim, za svaki novi oglas, sistem izračunava ocjenu sličnosti upoređujući skup vještina iz historije kandidata sa skupom vještina tog novog oglasa. Oglasi koji imaju najveći procenat podudaranja vještina dobijaju najvišu ocjenu i bivaju preporučeni kandidatu.

Konačni sistem kombinira ove dvije metode kako bi iskoristio prednosti obje:

- **Osnovna Ocjena (Content Score):** Filtriranje zasnovano na sadržaju (vještinama) služi kao osnova za preporuku. Oglasi koji nemaju nikakvo podudaranje vještina sa historijom kandidata se odmah odbacuju.
- **Bonus Ocjena (Collaborative Bonus):** Kolaborativno filtriranje se koristi kao "bonus". Ako model faktorizacije matrice predvidi visoku vjerovatnoću zajedničke prijave, ta preporuka dobija dodatne bodove. Ovo pomaže u otkrivanju neočekivanih, ali relevantnih preporuka koje se ne bi mogle pronaći samo analizom vještina.

Konačna ocjena za svaki oglas se računa po formuli:
finalScore = contentScore + (collaborativeScore * collaborativeBonusScale)

(collaborativeBonusScale trenutno iznosi 0.2, probao sam više vrijednosti ali ova daje najbolje rezultate za podatke iz seedera)

Preporuke se zatim sortiraju po finalScore u opadajućem redoslijedu i kandidatu se prikazuje prvih N rezultata.

Glavna Logika Recommender Sistema

Glavna logika za treniranje modela i generisanje preporuka nalazi se unutar JobPostingService servisa, metoda GetRecommendedJobsForCandidateAsync sadrži kompletan proces.

Putanja: ITapply.Services/Services/JobPostingService.cs

```
92 public async Task<List<JobPostingResponse>> GetRecommendedJobsForCandidateAsync(int candidateId, int count = 5)
93 {
94     var candidate = await _context.Candidates.FindAsync(candidateId);
95     if (candidate == null)
96     {
97         throw new UserException($"Candidate with ID {candidateId} not found");
98     }
99
100    TrainModelIfNotExists();
101
102    var candidateHistory = await _context.Applications
103        .Where(a => a.CandidateId == candidateId)
104        .Include(a => a.JobPosting)
105        .ThenInclude(jp => jp.JobPostingSkills)
106        .ToListAsync();
107
108    if (!candidateHistory.Any())
109    {
110        return new List<JobPostingResponse>();
111    }
112
113    var appliedJobIds = new HashSet<int>(candidateHistory.Select(a => a.JobPostingId));
114    var historicSkillIds = new HashSet<int>(candidateHistory.SelectMany(a => a.JobPosting.JobPostingSkills).Select(s => s.SkillId));
115
116    var recommendableJobs = await _context.JobPostings
117        .Include(j => j.Employer)
118        .Include(j => j.Location)
119        .Include(j => j.JobPostingSkills).ThenInclude(jps => jps.Skill)
120        .Where(j => j.Status == JobPostingStatus.Active && !appliedJobIds.Contains(j.Id))
121        .ToListAsync();
122
123    var predictionEngine = _model != null ? _mlContext.Model.CreatePredictionEngine<JobCoApplicationEntry, CoApplicationPrediction>(_model) : null;
124    var finalScores = new List<(JobPosting Job, double Score)>();
125
126    const double collaborativeBonusScale = 0.2;
127
128    foreach (var job in recommendableJobs)
129    {
130        // --- Content Score (Base Score) ---
131        double contentScore = 0;
132        if (job.JobPostingSkills.Any() && historicSkillIds.Any())
133        {
134            var jobSkillIds = job.JobPostingSkills.Select(s => s.SkillId);
135            int matchingSkills = historicSkillIds.Intersect(jobSkillIds).Count();
136            if (matchingSkills > 0)
137            {
138                int unionSkills = historicSkillIds.Count + jobSkillIds.Count() - matchingSkills;
139                contentScore = (double)matchingSkills / unionSkills;
140            }
141        }
142
143        if (contentScore == 0) continue;
144
145        // --- Collaborative Score (Bonus) ---
146        double collaborativeBonus = 0;
147        if (predictionEngine != null)
148        {
149            float cumulativePrediction = 0f;
150            foreach (var appliedJobId in appliedJobIds)
151            {
152                var prediction = predictionEngine.Predict(new JobCoApplicationEntry
153                {
154                    JobPostingId = (uint)appliedJobId,
155                    CoAppliedJobPostingId = (uint)job.Id
156                });
157                cumulativePrediction += prediction.Score;
158            }
159            if (appliedJobIds.Any())
160            {
161                // Normalize by app count and scale it down to a bonus
162                collaborativeBonus = (cumulativePrediction / appliedJobIds.Count) * collaborativeBonusScale;
163            }
164        }
165
166        // --- Final Score ---
167        double finalScore = contentScore + collaborativeBonus;
168        finalScores.Add((job, finalScore));
169    }
170
171    var finalRecommendations = finalScores
172        .OrderByDescending(x => x.Score)
173        .Take(count)
174        .Select(x => x.Job)
175        .ToList();
176
177    return finalRecommendations.Select(MapToResponse).ToList();
178 }
```

TrainModelIfNotExists metoda:

```
static MLContext _mlContext = null;
static ITransformer _model = null;
static readonly object _lock = new object();

private void TrainModelIfNotExists()
{
    lock (_lock)
    {
        if (_model != null) return;

        _mlContext = new MLContext();

        var allApplications = _context.Applications
            .Select(a => new { a.CandidateId, a.JobPostingId })
            .ToList();

        var coApplicationData = new List<JobCoApplicationEntry>();
        var groupedByCandidate = allApplications.GroupBy(a => a.CandidateId);

        foreach (var group in groupedByCandidate)
        {
            if (group.Count() > 1)
            {
                var jobIds = group.Select(g => g.JobPostingId).Distinct().ToList();
                for (int i = 0; i < jobIds.Count; i++)
                {
                    for (int j = i + 1; j < jobIds.Count; j++)
                    {
                        coApplicationData.Add(new JobCoApplicationEntry { JobPostingId = (uint)jobIds[i], CoAppliedJobPostingId = (uint)jobIds[j], Label = 1 });
                        coApplicationData.Add(new JobCoApplicationEntry { JobPostingId = (uint)jobIds[j], CoAppliedJobPostingId = (uint)jobIds[i], Label = 1 });
                    }
                }
            }
        }

        if (!coApplicationData.Any()) return;

        var trainData = _mlContext.Data.LoadFromEnumerable(coApplicationData);
        var options = new MatrixFactorizationTrainer.Options
        {
            MatrixColumnIndexColumnName = nameof(JobCoApplicationEntry.JobPostingId),
            MatrixRowIndexColumnName = nameof(JobCoApplicationEntry.CoAppliedJobPostingId),
            LabelColumnName = "Label",
            LossFunction = MatrixFactorizationTrainer.LossFunctionType.SquareLossOneClass,
            Alpha = 0.01,
            Lambda = 0.025,
            NumberOfIterations = 100,
            C = 0.00001
        };

        var estimator = _mlContext.Recommendation().Trainers.MatrixFactorization(options);
        _model = estimator.Fit(trainData);
    }
}
```

Prikaz Preporuka u Aplikaciji

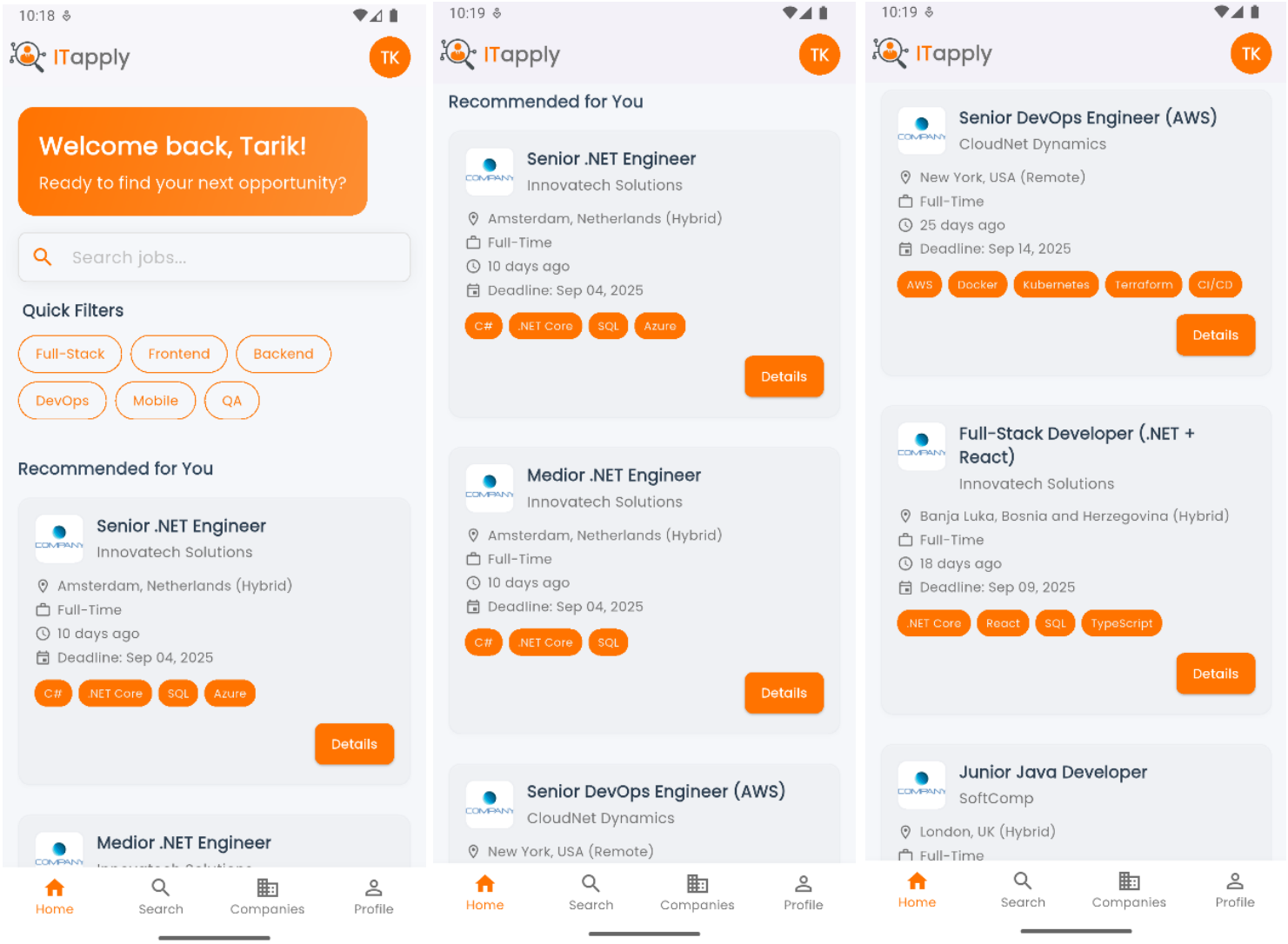
Preporuke se prikazuju na početnom ekranu (Home Screen) mobilne aplikacije za ulogovane korisnike koji imaju barem jednu prethodnu prijavu na oglas. Sekcija je jasno naznačena kao "Recommended for You".

Logika za pozivanje API-ja i prikaz preporuka se nalazi u HomeScreen widgetu.

Putanja: ITapply.UI/itapply_mobile/lib/screens/home_screen.dart

```
582 Future<void> _loadJobs() async {
583     try {
584       final jobProvider = context.read<JobPostingProvider>();
585       List<JobPosting> jobs;
586 
587       if (!widget.isGuest && _currentCandidate != null) {
588         jobs = await jobProvider.getRecommended(_currentCandidate!.id);
589 
590         if (jobs.isEmpty) {
591           jobs = await _loadActiveJobs();
592         }
593       } else {
594         jobs = await _loadActiveJobs();
595       }
596 
597       await _loadEmployerData(jobs);
598 
599       setState(() {
600         _jobs = jobs;
601         _filteredJobs = jobs;
602       });
603     } catch (e) {
604       throw Exception('Failed to load jobs: $e');
605     }
606   }
```

Home Page mobilne aplikacije:



Tehnologije

- **Machine Learning Biblioteka:** Microsoft.ML
- **Model:** MatrixFactorizationTrainer