Fakultet informacijskih tehnologija

eSalon – Recommender sistem
Razvoj softvera II

Predmetni profesori:

Prof. dr. Elmir Babović

Prof. dr. Denis Mušić

Student:

Ena Bećić, IB210012

Mostar, januar 2026.

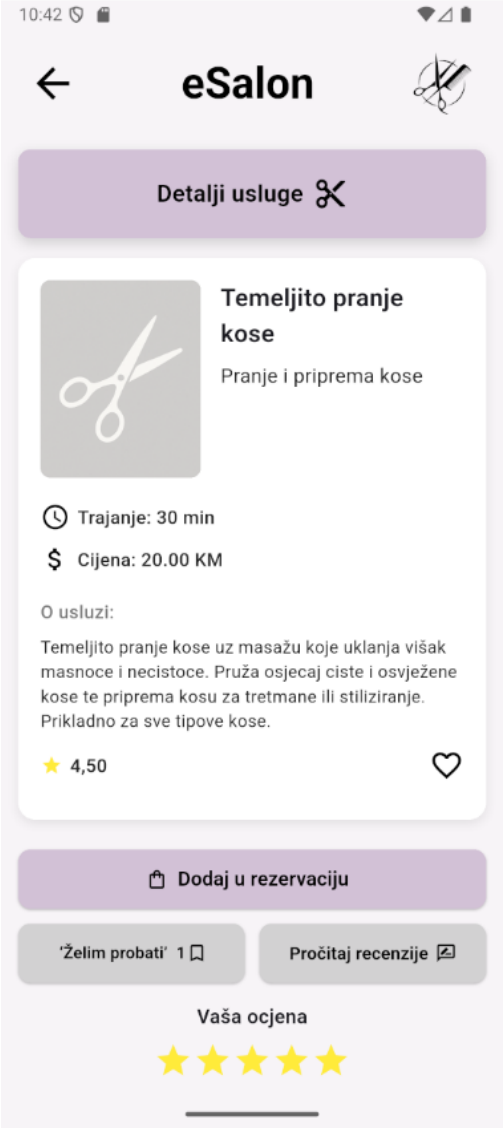
Sadržaj

1. Uvod

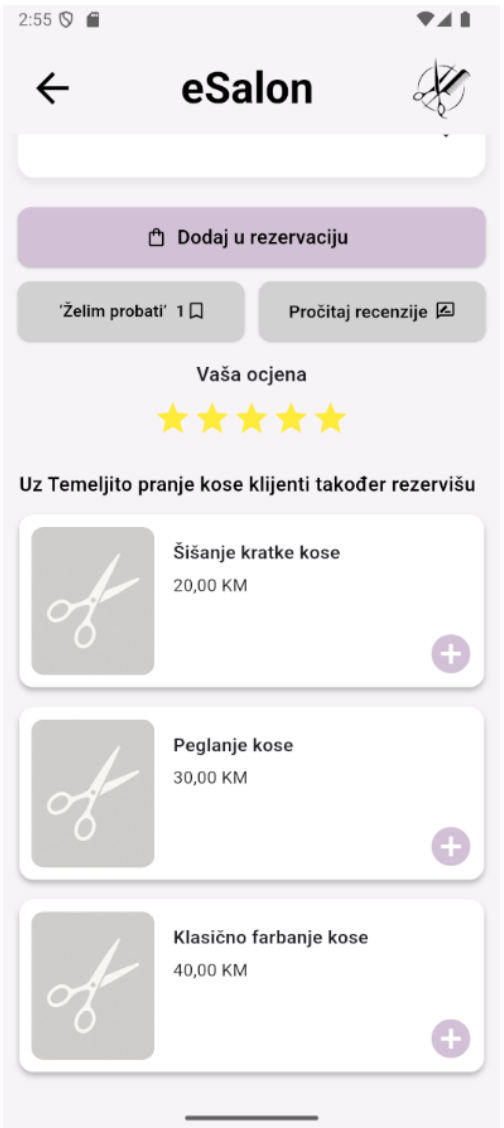
Na aplikaciji **eSalon** implementiran je sistem preporuke s ciljem pružanja personaliziranog korisničkog iskustva prilikom odabira usluga. Osnovni cilj sistema je povećati zadovoljstvo klijenata, omogućavajući im lakši pristup uslugama koje odgovaraju njihovim preferencama i navikama. Implementiran je algoritam preporuke zasnovan na **collaborative filtering** pristupu, koji koristi historiju rezervacija klijenta kako bi generisao relevantne preporuke usluga. Implementacija sistema preporuke, kao i lokacija njegovog source code-a, detaljno su predstavljeni u nastavku rada.

2. Collaborative filtering

Nakon što klijent pristupi detaljima određene usluge, skrolanjem, imat će uvid u dodatne preporuke. Konkretno, preporuke će biti generisane na osnovu zajedničkog ponašanja klijenata. Na primjer, kada klijent rezerviše određenu uslugu zajedno s drugim uslugama, ti podaci će biti korišteni za kreiranje preporuke za druge klijente. Na taj način, klijentima se nude usluge koje su drugi klijenti često kombinovali s trenutno pregledavanom uslugom. Da bi klijent pristupio ovom dijelu aplikacije, potrebno je da se prijavi na aplikaciju, a zatim da odabere željenu uslugu i pristupi njenim detaljima (**Slika 1**), te skrolanjem detalja usluge će imati uvid u usluge koje klijenti rezervišu, uz tu odabranu uslugu (**Slika 2**).



Slika 1. Prikaz detalja usluge



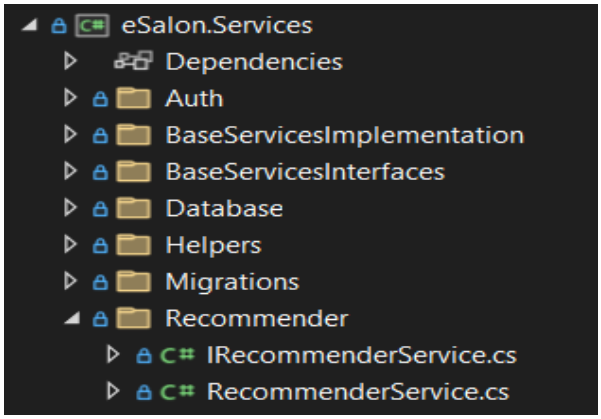
Slika 2. Prikaz preporučenih usluga

2.1. Implementacija

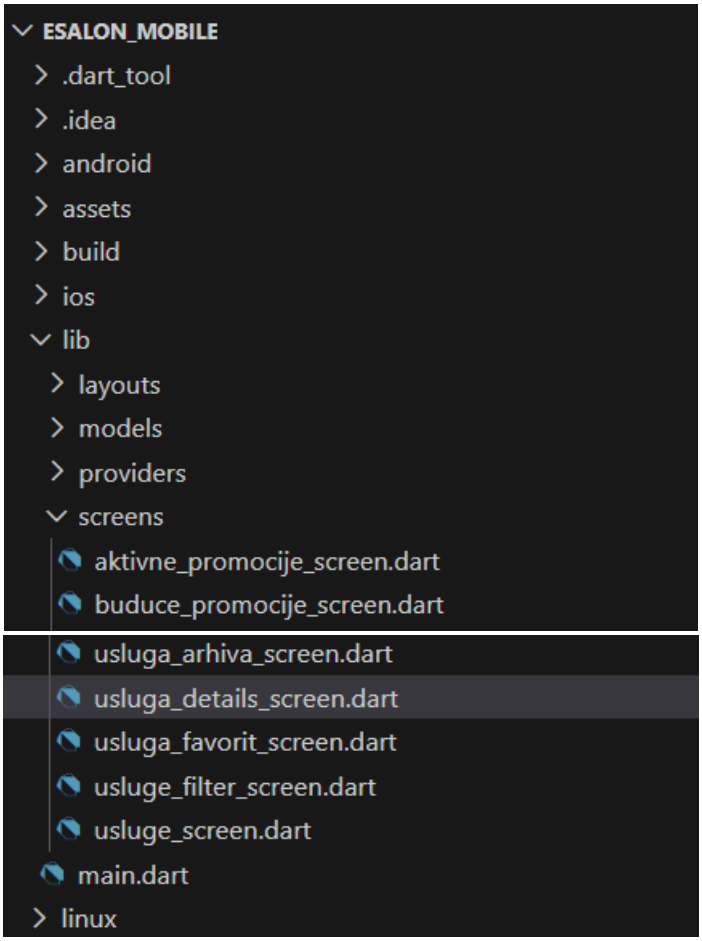
Za pristup source code-u, koji će biti objašnjen u nastavku, potrebno je navigirati na backendu u: **\eSalon_RS2\eSalon\eSalon.Services\Recommender\RecommenderService.cs** (**Slika 3**),

a na frontendu u:

\eSalon_RS2\eSalon\UI\esalon_mobile\lib\screens\usluga_details_screen.dart (**Slika 4**).



Slika 3. Lokacija u projektu (backend)



Slika 4. Lokacija u projektu (frontend)

2.1.1 Priprema okruženja za treniranje i korištenje modela preporuke

Prije same implementacije, definisane su varijable potrebne za treniranje i korištenje modela preporuke. Instanca MapsterMapper-a je **_mapper** koja služi za mapiranje entiteta iz baze u modele koje aplikacija koristi, dok **_context** predstavlja ESalonContext i omogućava pristup podacima o rezervacijama i uslugama. Za rad sa ML.NET bibliotekom instancirana je varijabla **mlContext**, neophodna za treniranje i predviđanje modela, dok je **model** tipa ITransformer i u njoj će biti sačuvan trenirani model preporuke. Objekat **isLocked** osigurava thread-safe pristup modelu kako bi se spriječilo paralelno treniranje, a **trainingTask** prati eventualno pokrenuto treniranje kako bi se čekalo njegovo završavanje prije korištenja modela. Zatim, **ModelFilePath** predstavlja naziv fajla u kojem će biti trajno sačuvan trenirani model - **"services-recommender-model.zip".**

```
private readonly IMapper _mapper;
private readonly ESalonContext _context;
private static MLContext? mlContext;
private static ITransformer? model;
private static readonly object isLocked = new();
private static Task? trainingTask;


private const string ModelFilePath = "services-recommender-model.zip";

0 references | Ena Bečić, 6 days ago | 1 author, 2 changes
public RecommenderService(ESalonContext context, IMapper mapper)
{
    _context = context;
    _mapper = mapper;
}
```

Slika 5. Priprema

2.1.2 Implementacija metode GetRecommendedServices

Metoda **GetRecommendedServices** koristi **co-purchase pristup**, što znači da analizira usluge koje se često rezervišu zajedno. Ako je klijent već rezervisao određenu uslugu, metoda koristi taj podatak da predvidi koje druge usluge bi mu mogle biti zanimljive na temelju sličnih rezervacija drugih klijenata.

Prilikom izvršavanja, metoda prolazi kroz nekoliko ključnih koraka:

1. **Provjera postojanja usluge u rezervacijama**: Ako usluga koju klijent trenutno pregledava nije bila rezervisana ranije, metoda odmah vraća praznu listu, jer nije moguće preporučiti nešto što još nije rezervisano i korišteno.
2. **Učitavanje ili treniranje modela za preporuke**: Ako model za preporuku još nije učitan, metoda ga učitava sa diska. Ako model ne postoji, poziva se funkcija TrainData() (**Slika 7**) koja trenira novi model koristeći postojeće podatke o zajedničkim rezervacijama usluga.
3. **Prikupljanje svih usluga iz baze podataka**: Nakon što je model učitan, dohvataju se sve usluge iz baze, osim one koja je trenutno pregledavana, jer nije potrebno preporučiti samu sebe.
4. **Predviđanje sličnosti između usluga**: Za svaku uslugu u bazi, koristi se trenirani model da se predvidi koliko je ta usluga povezana sa trenutno pregledavanom uslugom, na osnovu zajedničkih rezervacija. Predviđanja se vrše pomoću PredictionEngine-a, koji generira **co-purchase score** za svaku kombinaciju usluga.
5. **Dodavanje rezultata predviđanja u listu**: Svaka usluga i njen score se dodaju u listu, kako bi se pratili svi rezultati predviđanja za daljnju obradu.
6. **Sortiranje i filtriranje preporuka**: Nakon što su sve usluge procijenjene, lista se sortira prema score-u (od najvećeg prema najmanjem). Na kraju se odabiru tri usluge s najvišim score-om, tj. one koje se najčešće rezervišu zajedno s trenutno pregledavanom uslugom.
7. **Vraćanje preporučenih usluga**: Na kraju, metoda vraća listu preporučenih usluga mapiranu u model Model.Usluga.

```
2 references | Ena Bećić, 6 days ago | 1 author, 2 changes
public async Task<List<Model.Usluga>> GetRecommendedServices(int uslugaId)
{
    var exists = await _context.StavkeRezervacijes
        .AnyAsync(x => x.UslugaId == uslugaId && !x.IsDeleted);

    if (!exists)
        return new List<Model.Usluga>();

    if (mlContext == null)
    {
        bool needsTraining = false;

        lock (isLocked)
        {
            if (mlContext == null)
            {
                mlContext = new MLContext();

                if (File.Exists(ModelFilePath))
                {
                    using var stream = new FileStream(ModelFilePath, FileMode.Open, FileAccess.Read, FileShare.Read);
                    model = mlContext.Model.Load(stream, out _);
                }
                else
                {
                    needsTraining = true;
                }
            }
        }

        if (needsTraining)
        {
            lock (isLocked)
            {
                trainingTask ??= TrainData();
            }

            await trainingTask;
        }

    }
```

```
    if (model == null)
        return new List<Model.Usluga>();

    var services = await _context.Uslugas
        .Where(x => x.UslugaId != uslugaId && !x.IsDeleted)
        .ToListAsync();

    using var predictionEngine = mlContext.Model.CreatePredictionEngine<ServiceEntry, CopurchasePrediction>(model);

    var predictions = new List<(Usluga, float)>();

    foreach (var service in services)
    {
        var prediction = predictionEngine.Predict(new ServiceEntry
        {
            ServiceID = (uint)uslugaId,
            CoPurchaseServiceID = (uint)service.UslugaId
        });

        predictions.Add((service, prediction.Score));
    }

    var topServices = predictions
        .OrderByDescending(x => x.Item2)
        .Take(3)
        .Select(x => x.Item1)
        .ToList();

    return _mapper.Map<List<Model.Usluga>>(topServices);
}
```

Slika 6. Implementacija metode za preporuke često rezervisanih usluga uz odabranu uslugu

2.1.3 Implementacija metode TrainData

Slika 7. Implementacija metode **TrainData()**

Na slici (**Slika 7**) je prikazana metoda **TrainData**(), koja vrši treniranje modela preporuka ukoliko model još nije istreniran. Metoda prikuplja sve rezervacije iz baze podataka, uključujući stavke rezervacija, i kreira skup podataka ServiceEntry koji predstavlja parove usluga koje su klijenti

rezervisali zajedno. Svaki par usluga dobija vrijednost Label = 1f, što označava zajedničku rezervaciju.

Nakon pripreme podataka, metoda učitava podatke u ML.NET koristeći LoadFromEnumerable, te definira parametre za **Matrix Factorization** algoritam, uključujući imena kolona za usluge i labelu, vrstu funkcije gubitka, kao i hiperparametre poput Alpha, Lambda, broja iteracija i regularizacije C. Model se zatim trenira pozivom Fit(trainData) i rezultat se sprema u varijablu model.

Trenirani model se onda sačuva u fajl pod nazivom "services-recommender-model.zip", kako bi se mogao ponovo koristiti u metodi GetRecommendedServices (**Slika 6**), za predviđanje koje usluge klijenti često rezervišu zajedno.