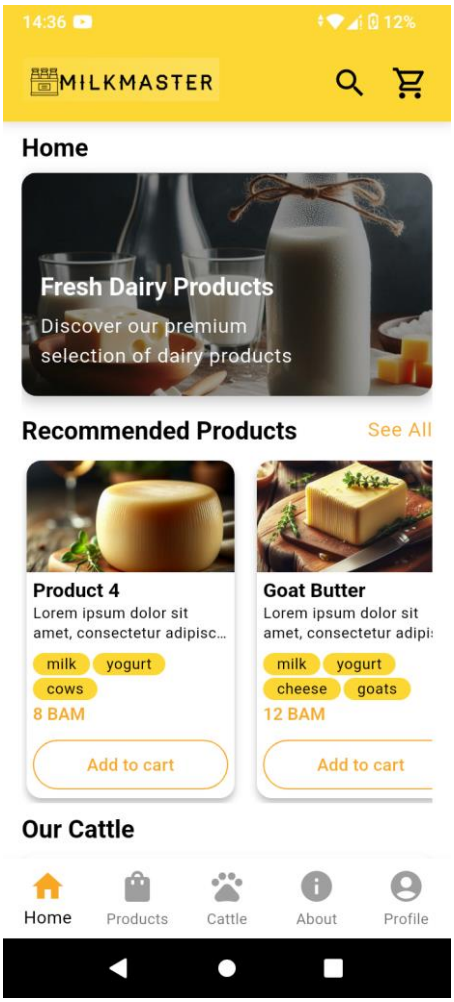
Sistem Preporuka za MilkMaster Aplikaciju

1.Uvod

MilkMaster aplikacija koristi napredni sistem mašinskog učenja za personalizovane preporuke proizvoda korisnicima. Sistem analizira istoriju narudžbi korisnika i koristi kolaborativno filtriranje kako bi predložio proizvode koji će najverovatnije biti od interesa za svakog pojedinačnog korisnika. Sistem preporuke se nalazi na linku. Implementacija na frontendu se nalazi na slijedecem linku.



Sistem preporuke u aplikaciji

2.Tehnologije i Framework

2.1. ML.NET Framework

Sistem preporuka implementiran je korišćenjem **ML.NET** framework-a, Microsoft-ovog open-source machine learning framework-a za .NET aplikacije.

Ključne komponente:

- **MLContext**: Glavni kontekst za ML operacije
- **MatrixFactorization**: Algoritam za kolaborativno filtriranje
- **ITransformer**: Trenirani model koji se koristi za predikcije

2.2. Matrix Factoriyation Algoritam

Koristi se **Matrix Factorization Trainer** sa sledećim parametrima:

- LossFunction: SquareLossOneClass (optimizovano za implicit feedback)

- Alpha: 0.01 (learning rate)

- Lambda: 0.025 (regularization parameter)

- NumberOfIterations: 100

- C: 0.00001 (regularization weight)

3. Arhitektura Sistema

3.1. Ulazni Podatci

Sistem koristi **implicit feedback** pristup baziran na stvarnim narudžbama.

Izvor podataka:

- Istorija narudžbi iz Orders tabele
- Sve stavke narudžbi iz OrderItems tabele
- Label = 1.0 za svaki kupljen proizvod (implicit positive feedback)

3.2.Mapiranje Korisnika

Zbog ograničenja ML.NET-a koji zahteva numeričke ID-ove, implementirano je mapiranje string ID-ova u numeričke, slika 1.

```
263
        2 references
264     static Dictionary<string, uint> _userIdMap = new();
        1 reference
265     static uint _userCounter = 0;
266
        2 references
267     private static uint MapUserId(string userId)
268     {
269         if (!_userIdMap.TryGetValue(userId, out var mappedId))
270         {
271             mappedId = _userCounter++;
272             _userIdMap[userId] = mappedId;
273         }
274         return mappedId;
275     }
```

Slika 1

4.Proces Treniranja Modela

4.1.Priprema Podataka

Učitavaju se sve narudžbe sa pripadajućim stavkama. Nakon toga, filtriraju se narudžbe bez korisničkog ID-a i kreira se rating lista sto je prikazano na slici 2.

```
foreach (var order in orders)
{
    if (order.UserId == null) continue;

    var mappedUserId = MapUserId(order.UserId);

    foreach (var item in order.Items)
    {
        ratings.Add(new ProductEntry
        {
            UserId = mappedUserId,
            ProductId = (uint)item.ProductId,
            Label = 1f
        });
    }
}
```

Slika 2

Nakon toga učitavaju se podatci u MLContext slika 3.

```
var traindata = mlContext.Data.LoadFromEnumerable(ratings);
```

Slika 3

4.2.Trening

Model se trenira **lazy loading** pristupom:

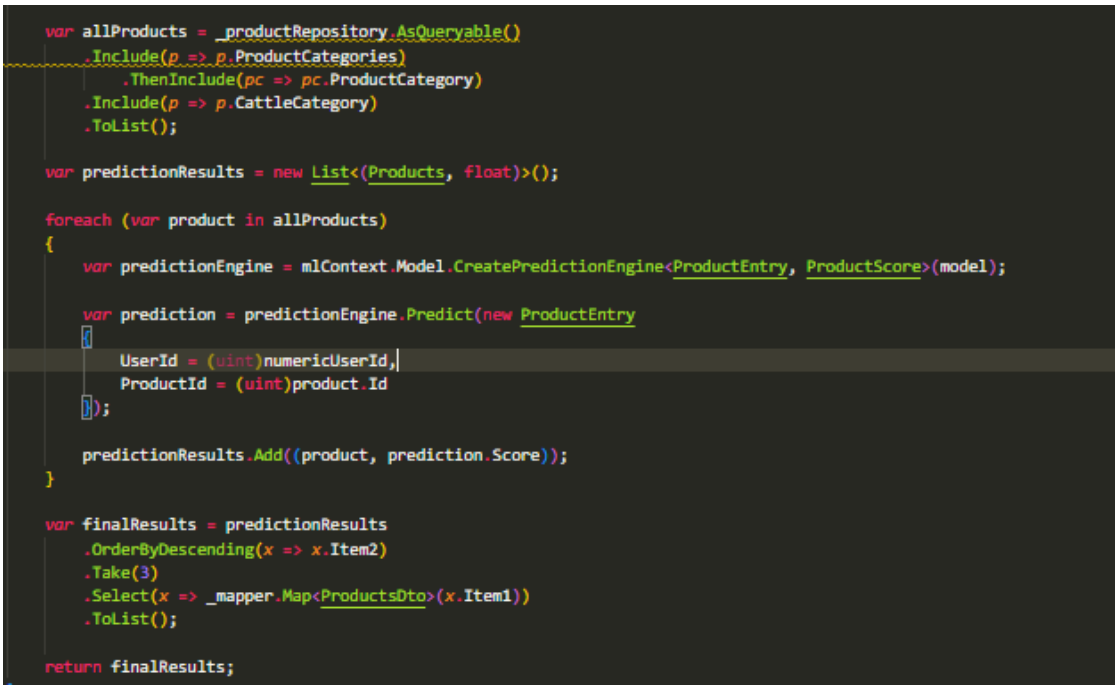
- Model se kreira samo pri prvom zahtjevu
- Koristi se **lock** mehanizam za thread-safety
- Model ostaje u memoriji za brže naknadne predikcije

```
lock (isLocked)
{
    if (mlContext == null)
    {
        mlContext = new MLContext();
        // ... priprema podataka ...
        var estimator = mlContext.Recommendation()
            .Trainers.MatrixFactorization(options);
        model = estimator.Fit(traindata);
    }
}
```

Slika 4

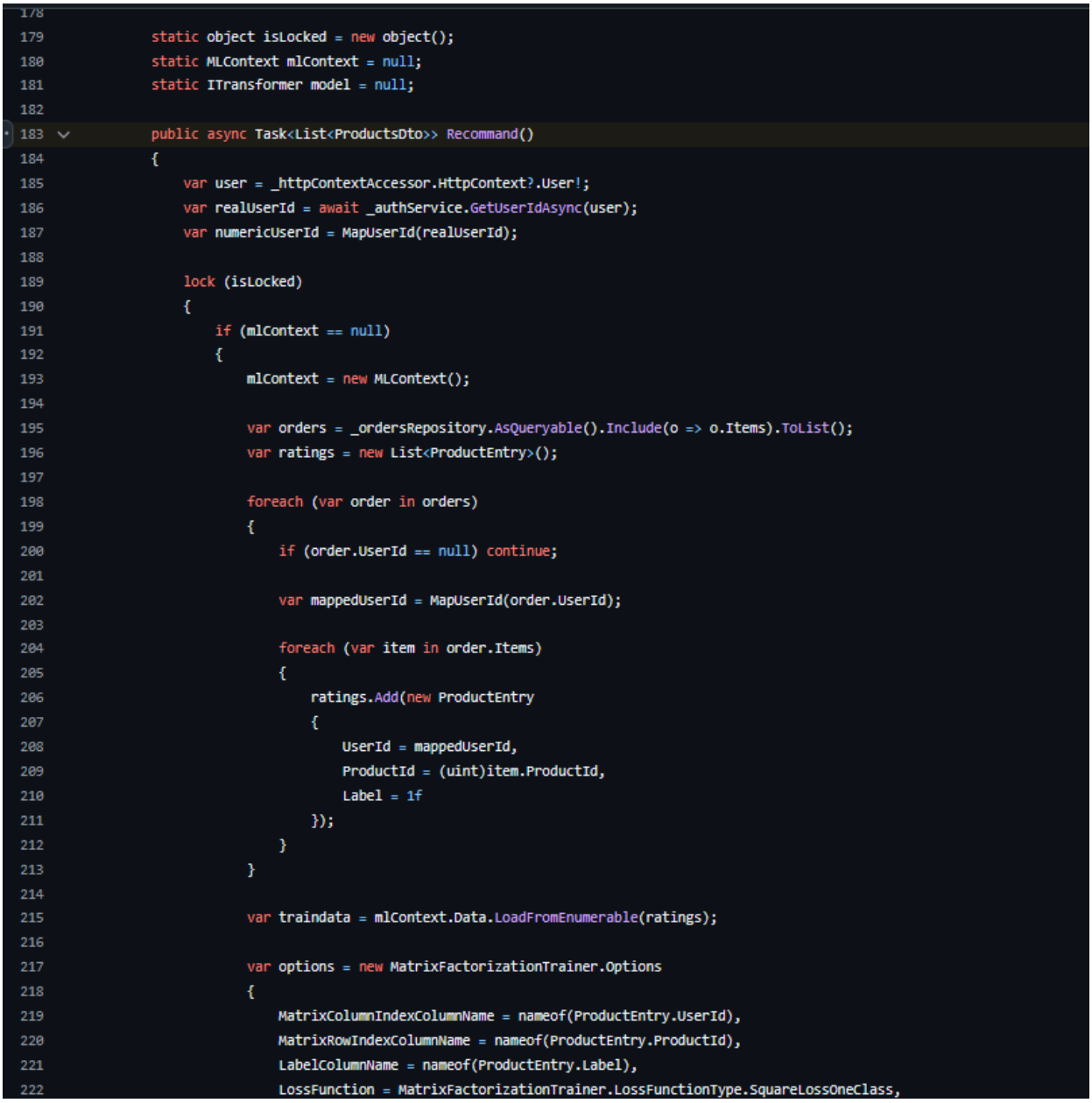
5.Generisanje Preporuka

Za svakog korisnika se učitavaju svi proizvodi iz baze, priprema se lista tuple-ova za rezultate predikcije. Za svaki proizvod kreira se **PredictionEngine** i poziva se **Predict** koji vraća **score** koji pokazuje koliko je proizvod relevantan za korisnika. Rezultati se spremaju u listu te se nakon toga filtriraju i uzimaju top 3 produkta, slika 5.



```
var allProducts = _productRepository.AsQueryable()
    .Include(p => p.ProductCategories)
        .ThenInclude(pc => pc.ProductCategory)
    .Include(p => p.CattleCategory)
    .ToList();

var predictionResults = new List<(Products, float)>();

foreach (var product in allProducts)
{
    var predictionEngine = mlContext.Model.CreatePredictionEngine<ProductEntry, ProductScore>(model);

    var prediction = predictionEngine.Predict(new ProductEntry
    {
        UserId = (uint)numericUserId,
        ProductId = (uint)product.Id
    });

    predictionResults.Add((product, prediction.Score));
}

var finalResults = predictionResults
    .OrderByDescending(x => x.Item2)
    .Take(3)
    .Select(x => _mapper.Map<ProductsDto>(x.Item1))
    .ToList();

return finalResults;
```

Slika 5



```
178
179         static object isLocked = new object();
180         static MLContext mlContext = null;
181         static ITransformer model = null;
182
183         public async Task<List<ProductsDto>> Recommand()
184         {
185             var user = _httpContextAccessor.HttpContext?.User!;
186             var realUserId = await _authService.GetUserIdAsync(user);
187             var numericUserId = MapUserId(realUserId);
188
189             lock (isLocked)
190             {
191                 if (mlContext == null)
192                 {
193                     mlContext = new MLContext();
194
195                     var orders = _ordersRepository.AsQueryable().Include(o => o.Items).ToList();
196                     var ratings = new List<ProductEntry>();
197
198                     foreach (var order in orders)
199                     {
200                         if (order.UserId == null) continue;
201
202                         var mappedUserId = MapUserId(order.UserId);
203
204                         foreach (var item in order.Items)
205                         {
206                             ratings.Add(new ProductEntry
207                             {
208                                 UserId = mappedUserId,
209                                 ProductId = (uint)item.ProductId,
210                                 Label = 1f
211                             });
212                         }
213                     }
214
215                     var traindata = mlContext.Data.LoadFromEnumerable(ratings);
216
217                     var options = new MatrixFactorizationTrainer.Options
218                     {
219                         MatrixColumnIndexColumnName = nameof(ProductEntry.UserId),
220                         MatrixRowIndexColumnName = nameof(ProductEntry.ProductId),
221                         LabelColumnName = nameof(ProductEntry.Label),
222                         LossFunction = MatrixFactorizationTrainer.LossFunctionType.SquareLossOneClass,
```

Slika prvog dijela sistema preporuke

```
                    var options = new MatrixFactorizationTrainer.Options
                    {
                        MatrixColumnIndexColumnName = nameof(ProductEntry.UserId),
                        MatrixRowIndexColumnName = nameof(ProductEntry.ProductId),
                        LabelColumnName = nameof(ProductEntry.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);
                }
            }

            var allProducts = _productRepository.AsQueryable()
                .Include(p => p.ProductCategories)
                    .ThenInclude(pc => pc.ProductCategory)
                .Include(p => p.CattleCategory)
                .ToList();

            var predictionResults = new List<(Products, float)>();

            foreach (var product in allProducts)
            {
                var predictionEngine = mlContext.Model.CreatePredictionEngine<ProductEntry, ProductScore>(model);

                var prediction = predictionEngine.Predict(new ProductEntry
                {
                    UserId = (uint)numericUserId,
                    ProductId = (uint)product.Id
                });

                predictionResults.Add((product, prediction.Score));
            }

            var finalResults = predictionResults
                .OrderByDescending(x => x.Item2)
                .Take(3)
                .Select(x => _mapper.Map<ProductsDto>(x.Item1))
                .ToList();
```

Slika drugog djela sistema preporuke

```
183         public async Task<List<ProductsDto>> Recommand()
250                 });
251
252                 predictionResults.Add((product, prediction.Score));
253             }
254
255             var finalResults = predictionResults
256                 .OrderByDescending(x => x.Item2)
257                 .Take(3)
258                 .Select(x => _mapper.Map<ProductsDto>(x.Item1))
259                 .ToList();
260
261             return finalResults;
262         }
263
264         static Dictionary<string, uint> _userIdMap = new();
265         static uint _userCounter = 0;
266
267         private static uint MapUserId(string userId)
268         {
269             if (!_userIdMap.TryGetValue(userId, out var mappedId))
270             {
271                 mappedId = _userCounter++;
272                 _userIdMap[userId] = mappedId;
273             }
274             return mappedId;
275         }
276
277     }
278     public class ProductScore
279     {
280         public float Score { get; set; }
281     }
282     public class ProductEntry
283     {
284         [KeyType(count: 10)]
285         public uint UserId { get; set; }
286         [KeyType(count:10)]
287         public uint ProductId { get; set; }
288         public float Label { get; set; }
289     }
290 }
```

Slika trećeg dijela sistema preporuke