Datum 25.8.2025

Opis recommender sistema

SmartPhone++

Sistem za upravljanje servisa i online prodaju

Predmetni profesori :

dr. sc. Elmir Babović , dr. sc. Denis Mušić

Student :

Jasir Burić – IB170208

ProductRecom...ionService.cs

SmartPhone.Services | SmartPhone.Services.Services.ProductRecommendationService | _context

```
using Microsoft.EntityFrameworkCore;
using SmartPhone.Model.Responses;
using SmartPhone.Services.Database;
using SmartPhone.Services.Interfaces;
using MapsterMapper;

namespace SmartPhone.Services.Services
{
    2 references | Jasir Burić,
    public class ProductRecommendationService : IProductRecommendationService
    {
        private readonly SmartPhoneDbContext _context;
        private readonly IMapper _mapper;

        0 references | Jasir Burić, 4 days ago | 1 author, 1 change
        public ProductRecommendationService(SmartPhoneDbContext context, IMapper mapper)
        {
            _context = context;
            _mapper = mapper;
        }

        2 references | Jasir Burić, 15 hours ago | 1 author, 2 changes
        public async Task<List<ProductResponse>> GetRecommendationsForUserAsync(int userId, int maxRecommendations = 3)
    {
        try
        {
            // Get the user's cart and cart items from the database
            var userCart = await _context.Carts
                .Include(c => c.CartItems)
                    .ThenInclude(ci => ci.Product)
                        .ThenInclude(p => p.Category)
                .Include(c => c.CartItems)
                    .ThenInclude(ci => ci.Product)
                        .ThenInclude(p => p.ProductImages)
                .FirstOrDefaultAsync(c => c.UserId == userId && c.IsActive);

            if (userCart?.CartItems == null || !userCart.CartItems.Any())
            {
                // If no cart items, return featured products
                return await GetFeaturedRecommendationsAsync(new List<int>(), maxRecommendations);
            }
```

ProductRecom...ionService.cs

SmartPhone.Services | SmartPhone.Services.Services.ProductRecommendationService | GetRecommendationsForUserAsync(int userId, int maxRecommendations = 3)

```
                .FirstOrDefaultAsync(c => c.UserId == userId && c.IsActive);

            if (userCart?.CartItems == null || !userCart.CartItems.Any())
            {
                // If no cart items, return featured products
                return await GetFeaturedRecommendationsAsync(new List<int>(), maxRecommendations);
            }

            // Extract data from cart items
            var cartItemProductIds = userCart.CartItems.Select(ci => ci.ProductId).ToList();
            var cartItemProductNames = userCart.CartItems.Select(ci => ci.Product.Name).ToList();
            var cartItemCategoryIds = userCart.CartItems.Select(ci => ci.Product.CategoryId).ToList();

            // Use the existing logic to get recommendations
            return await GetRecommendationsAsync(cartItemProductIds, cartItemProductNames, cartItemCategoryIds, maxRecommendations);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error getting recommendations for user {userId}: {ex.Message}");
            return new List<ProductResponse>();
        }
    }

    3 references | Jasir Burić, 15 hours ago | 1 author, 2 changes
    public async Task<List<ProductResponse>> GetRecommendationsAsync(
        List<int> cartItemProductIds,
        List<string> cartItemProductNames,
        List<int> cartItemCategoryIds,
        int maxRecommendations = 3)
    {
        var recommendations = new List<ProductResponse>();

        try
        {
            // Rule 1: Find products with similar names (cases, glass, accessories)
            var nameBasedRecommendations = await GetNameBasedRecommendationsAsync(
                cartItemProductNames, cartItemProductIds, maxRecommendations / 2);
            recommendations.AddRange(nameBasedRecommendations);

            // Rule 2: Find products from different categories (complementary items)
            var categoryBasedRecommendations = await GetCategoryBasedRecommendationsAsync(
```

ProductRecom...ionService.cs

SmartPhone.Services | SmartPhone.Services.Services.ProductRecommendationService | GetRecommendationsForUserAsync(int userId, int maxRecommendations = 3)

```
64          {
65              // Rule 1: Find products with similar names (cases, glass, accessories)
66              var nameBasedRecommendations = await GetNameBasedRecommendationsAsync(
67                  cartItemProductNames, cartItemProductIds, maxRecommendations / 2);
68              recommendations.AddRange(nameBasedRecommendations);
69
70              // Rule 2: Find products from different categories (complementary items)
71              var categoryBasedRecommendations = await GetCategoryBasedRecommendationsAsync(
72                  cartItemCategoryIds, cartItemProductIds, maxRecommendations - recommendations.Count);
73              recommendations.AddRange(categoryBasedRecommendations);
74
75              // Rule 3: If we still need more recommendations, add featured products
76              if (recommendations.Count < maxRecommendations)
77              {
78                  var featuredRecommendations = await GetFeaturedRecommendationsAsync(
79                      cartItemProductIds, maxRecommendations - recommendations.Count);
80                  recommendations.AddRange(featuredRecommendations);
81              }
82
83              // Remove duplicates and ensure we don't exceed max recommendations
84              return recommendations
85                  .GroupBy(r => r.Id)
86                  .Select(g => g.First())
87                  .Take(maxRecommendations)
88                  .ToList();
89          }
90          catch (Exception ex)
91          {
92              // Log the error but return empty list to avoid breaking the UI
93              Console.WriteLine($"Error getting product recommendations: {ex.Message}");
94              return new List<ProductResponse>();
95          }
96      }
97
        1 reference | Jasir Burić, 4 days ago | 1 author, 1 change
98      private async Task<List<ProductResponse>> GetNameBasedRecommendationsAsync(
99          List<string> cartItemProductNames,
100         List<int> cartItemProductIds,
101         int maxCount)
102     {
103         if (!cartItemProductNames.Any() || maxCount <= 0)
```

100 % | 0 | 1 | Ln: 33 | Ch: 17 | SPC | CRLF

ProductRecom...ionService.cs

SmartPhone.Services | SmartPhone.Services.Services.ProductRecommendationService | GetRecommendationsForUserAsync(int userId, int maxRecommendations = 3)

```
141                 var productResponse = _mapper.Map<ProductResponse>(product);
142                 recommendations.Add(productResponse);
143             }
144         }
145     }
146 }
147
148 return recommendations;
149 }
150
        1 reference | Jasir Burić, 15 hours ago | 1 author, 2 changes
151 private async Task<List<ProductResponse>> GetCategoryBasedRecommendationsAsync(
152     List<int> cartItemCategoryIds,
153     List<int> cartItemProductIds,
154     int maxCount)
155 {
156     if (!cartItemCategoryIds.Any() || maxCount <= 0)
157         return new List<ProductResponse>();
158
159     var recommendations = new List<ProductResponse>();
160
161     // Get cart items to analyze their brands and names for smarter recommendations
162     var cartItems = await _context.CartItems
163         .Include(ci => ci.Product)
164         .Where(ci => cartItemProductIds.Contains(ci.ProductId))
165         .ToListAsync();
166
167     // If no cart items found, try getting products directly
168     if (!cartItems.Any())
169     {
170         var products = await _context.Products
171             .Where(p => cartItemProductIds.Contains(p.Id))
172             .ToListAsync();
173
174         // Create pseudo cart items for compatibility
175         cartItems = products.Select(p => new Database.CartItem
176         {
177             Product = p,
178             ProductId = p.Id
179         }).ToList();
180     }
```

100 % | 0 | 1 | Ln: 33 | Ch: 17 | SPC | CRLF

ProductRecom...ionService.cs

SmartPhone.Services | SmartPhone.Services.Services.ProductRecommendationService | GetRecommendationsForUserAsync(int userId, int maxRecommendations = 3)

```
182                 // Extract brands and key product information from cart items
183                 var cartBrands = cartItems
184                     .Select(ci => ci.Product.Brand)
185                     .Where(brand => !string.IsNullOrEmpty(brand))
186                     .Distinct()
187                     .ToList();
188
189                 var cartProductNames = cartItems
190                     .Select(ci => ci.Product.Name)
191                     .ToList();
192
193                 // Define complementary category mappings
194                 var complementaryCategories = GetComplementaryCategories();
195
196                 foreach (var categoryId in cartItemCategoryIds)
197                 {
198                     if (recommendations.Count >= maxCount) break;
199
200                     if (complementaryCategories.TryGetValue(categoryId, out var complementaryCategoryIds))
201                     {
202                         foreach (var complementaryCategoryId in complementaryCategoryIds)
203                         {
204                             if (recommendations.Count >= maxCount) break;
205
206                             // Get products from complementary categories
207                             var complementaryProducts = await _context.Products
208                                 .Include(p => p.Category)
209                                 .Include(p => p.ProductImages)
210                                 .Where(p => p.IsActive &&
211                                            p.StockQuantity > 0 &&
212                                            p.CategoryId == complementaryCategoryId &&
213                                            !cartItemProductIds.Contains(p.Id))
214                                 .ToListAsync();
215
216                             // Filter products to match brands or product compatibility
217                             var compatibleProducts = complementaryProducts
218                                 .Where(p => IsProductCompatibleWithCart(p, cartBrands, cartProductNames))
219                                 .OrderByDescending(p => p.IsFeatured)
220                                 .ThenByDescending(p => p.StockQuantity)
221                                 .Take(maxCount - recommendations.Count)
222                                 .ToList();
```

Snipping Tool

Screenshot copied to clipboard
Automatically saved to screenshots folder.

Markup and share

ProductRecom...ionService.cs

SmartPhone.Services | SmartPhone.Services.Services.ProductRecommendationService | GetRecommendationsForUserAsync(int userId, int maxRecommendations = 3)

```
224                             foreach (var product in compatibleProducts)
225                             {
226                                 if (recommendations.Count >= maxCount) break;
227
228                                 if (!recommendations.Any(r => r.Id == product.Id))
229                                 {
230                                     var productResponse = _mapper.Map<ProductResponse>(product);
231                                     recommendations.Add(productResponse);
232                                 }
233                             }
234                         }
235                     }
236                 }
237
238                 return recommendations;
239             }
240
                2 references | Jasir Burić, 4 days ago | 1 author, 1 change
241             private async Task<List<ProductResponse>> GetFeaturedRecommendationsAsync(
242                 List<int> cartItemProductIds,
243                 int maxCount)
244             {
245                 if (maxCount <= 0)
246                     return new List<ProductResponse>();
247
248                 var featuredProducts = await _context.Products
249                     .Include(p => p.Category)
250                     .Include(p => p.ProductImages)
251                     .Where(p => p.IsActive &&
252                                p.StockQuantity > 0 &&
253                                p.IsFeatured &&
254                                !cartItemProductIds.Contains(p.Id))
255                     .OrderByDescending(p => p.StockQuantity)
256                     .Take(maxCount)
257                     .ToListAsync();
258
259                 return featuredProducts.Select(p => _mapper.Map<ProductResponse>(p)).ToList();
260             }
261
                3 references | Jasir Burić, 4 days ago | 1 author, 1 change
262             private List<string> ExtractKeyWords(string productName)
```

100 % | 0 | 1 | Ln: 33 | Ch: 17 | SPC | CRLF

ProductRecom...ionService.cs

SmartPhone.Services | SmartPhone.Services.Services.ProductRecommendationService | GetRecommendationsForUserAsync(int userId, int maxRecommendations = 3)

3 references | Jasir Burić, 4 days ago | 1 author, 1 change

```
private List<string> ExtractKeyWords(string productName)
{
    if (string.IsNullOrWhiteSpace(productName))
        return new List<string>();

    // Split by common separators and filter out common words
    var commonWords = new HashSet<string>(StringComparer.OrdinalIgnoreCase)
    {
        "the", "a", "an", "and", "or", "but", "in", "on", "at", "to", "for", "of", "with", "by"
    };

    var words = productName.Split(new[] { ' ', '-', '_', '.', ',' }, StringSplitOptions.RemoveEmptyEntries)
        .Where(word => word.Length > 1 && !commonWords.Contains(word.ToLower()))
        .ToList();

    return words;
}
```

1 reference | Jasir Burić, 15 hours ago | 1 author, 1 change

```
private bool IsProductCompatibleWithCart(Database.Product product, List<string> cartBrands, List<string> cartProductNames)
{
    // If no brands in cart, allow any
    if (!cartBrands.Any())
        return true;

    // Check if product brand matches any cart item brand
    if (!string.IsNullOrEmpty(product.Brand))
    {
        foreach (var cartBrand in cartBrands)
        {
            if (product.Brand.Contains(cartBrand, StringComparison.OrdinalIgnoreCase) ||
                cartBrand.Contains(product.Brand, StringComparison.OrdinalIgnoreCase))
            {
                return true;
            }
        }
    }

    // Check if product name contains any brand from cart items
    if (!string.IsNullOrEmpty(product.Name))
```

bool ProductRecommendationService.IsProductCompatibleWithCart(Product product, List<string> cartBrands, List<string> cartProductNames)

100 % | 0 | 1 | Ln: 33 | Ch: 17 | SPC | CRLF

ProductRecom...ionService.cs

SmartPhone.Services | SmartPhone.Services.Services.ProductRecommendationService | GetRecommendationsForUserAsync(int userId, int maxRecommendations = 3)

```
    // Check if product name contains any brand from cart items
    if (!string.IsNullOrEmpty(product.Name))
    {
        foreach (var cartBrand in cartBrands)
        {
            if (product.Name.Contains(cartBrand, StringComparison.OrdinalIgnoreCase))
            {
                return true;
            }
        }
    }

    // Check for model compatibility (e.g., "S24" in cart, recommend "S24" accessories)
    foreach (var cartProductName in cartProductNames)
    {
        var cartKeyWords = ExtractKeyWords(cartProductName);
        var productKeyWords = ExtractKeyWords(product.Name);

        // Check if they share significant keywords (models, series, etc.)
        var commonKeywords = cartKeyWords.Intersect(productKeyWords, StringComparer.OrdinalIgnoreCase).ToList();

        // If they share at least one meaningful keyword, consider them compatible
        if (commonKeywords.Any())
        {
            return true;
        }
    }

    // For universal accessories (like generic chargers, screen protectors), check if they mention "universal" or "compatible"
    if (!string.IsNullOrEmpty(product.Name) &&
        (product.Name.Contains("universal", StringComparison.OrdinalIgnoreCase) ||
         product.Name.Contains("compatible", StringComparison.OrdinalIgnoreCase) ||
         product.Name.Contains("all phones", StringComparison.OrdinalIgnoreCase)))
    {
        return true;
    }

    return false;
}
```

1 reference | Jasir Burić, 4 days ago | 1 author, 1 change

```
private Dictionary<int, List<int>> GetComplementaryCategories()
```

100 % | 0 | 1 | Ln: 33 | Ch: 17 | SPC | CRLF

ProductRecom...ionService.cs

SmartPhone.Services | SmartPhone.Services.Services.ProductRecommendationService | GetRecommendationsForUserAsync(int userId, int maxRecommendations = 3)

```
        try
        {
            // Rule 1: Find products with similar names (cases, glass, accessories)
            var nameBasedRecommendations = await GetNameBasedRecommendationsAsync(
                cartItemProductNames, cartItemProductIds, maxRecommendations / 2);
            recommendations.AddRange(nameBasedRecommendations);

            // Rule 2: Find products from different categories (complementary items)
            var categoryBasedRecommendations = await GetCategoryBasedRecommendationsAsync(
                cartItemCategoryIds, cartItemProductIds, maxRecommendations - recommendations.Count);
            recommendations.AddRange(categoryBasedRecommendations);

            // Rule 3: If we still need more recommendations, add featured products
            if (recommendations.Count < maxRecommendations)
            {
                var featuredRecommendations = await GetFeaturedRecommendationsAsync(
                    cartItemProductIds, maxRecommendations - recommendations.Count);
                recommendations.AddRange(featuredRecommendations);
            }

            // Remove duplicates and ensure we don't exceed max recommendations
            return recommendations
                .GroupBy(r => r.Id)
                .Select(g => g.First())
                .Take(maxRecommendations)
                .ToList();
        }
        catch (Exception ex)
        {
            // Log the error but return empty list to avoid breaking the UI
            Console.WriteLine($"Error getting product recommendations: {ex.Message}");
            return new List<ProductResponse>();
        }
    }

    1 reference | Jasir Burić, 4 days ago | 1 author, 1 change
    private async Task<List<ProductResponse>> GetNameBasedRecommendationsAsync(
        List<string> cartItemProductNames,
        List<int> cartItemProductIds,
        int maxCount)
    {
        if (!cartItemProductNames.Any() || maxCount <= 0)
```

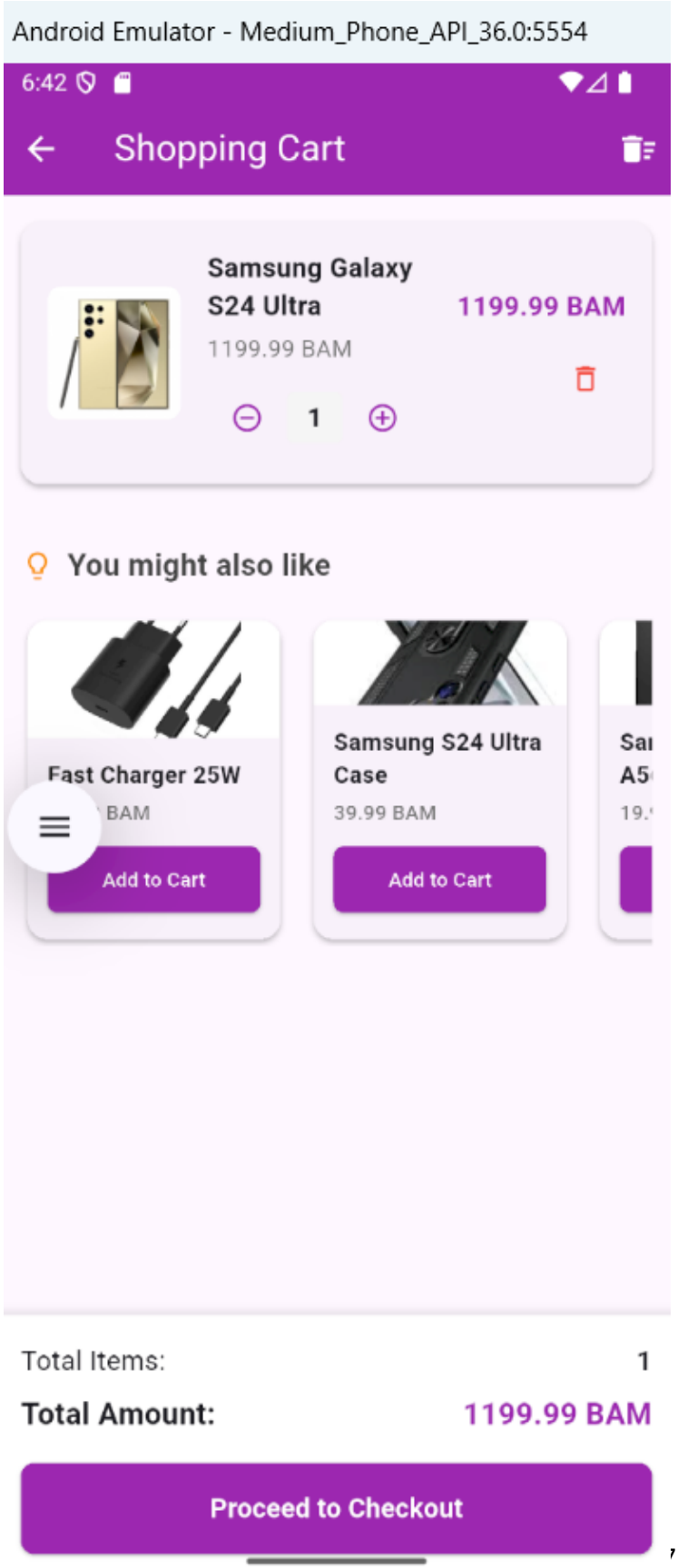
100 % | 0 | 1 | Ln: 33 | Ch: 17 | SPC | CRLF

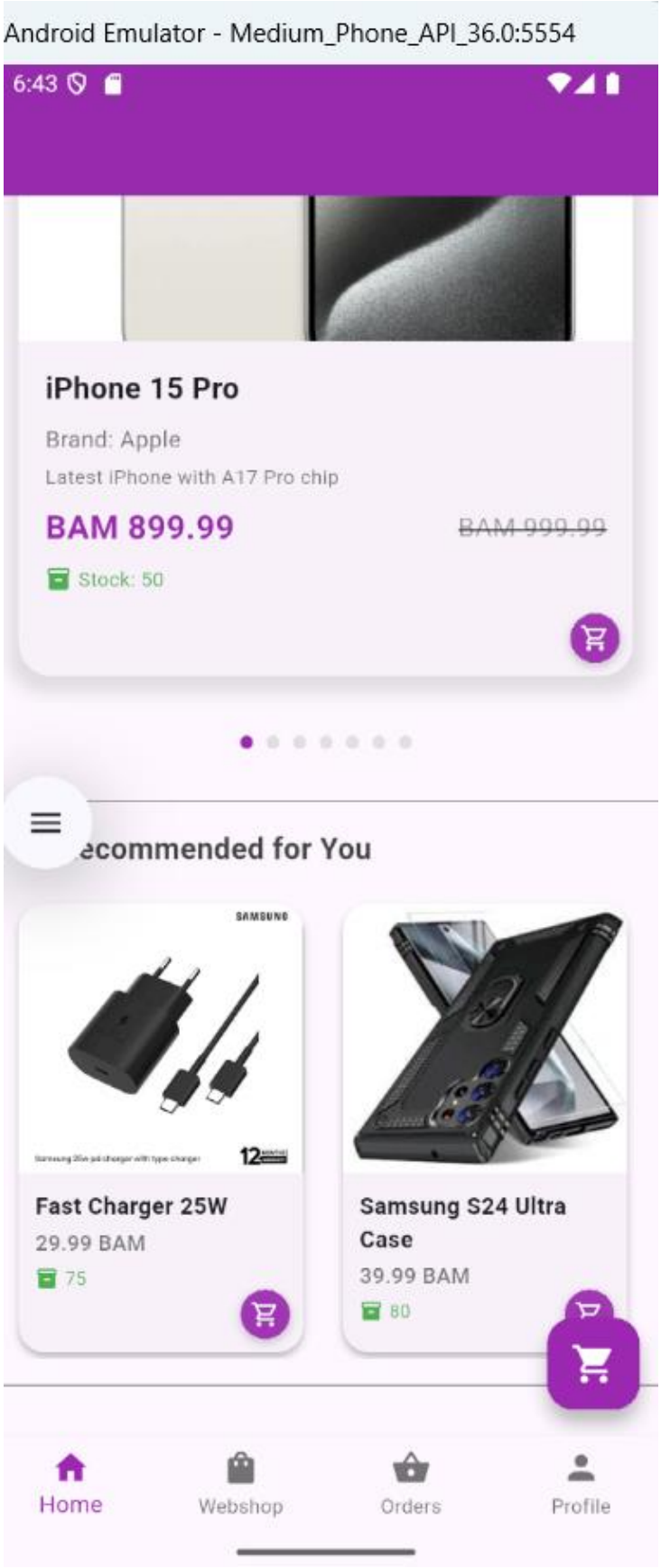
ProductReco...onService.cs*

SmartPhone.Services | SmartPhone.Services.Services.ProductRecommendationService | GetComplementaryCategories()

```
        return false;
    }

    1 reference | Jasir Burić, 4 days ago | 1 author, 1 change
    private Dictionary<int, List<int>> GetComplementaryCategories()
    {
        // Define which categories complement each other
        return new Dictionary<int, List<int>>
        {
            // Smartphones -> Accessories, Cases, Chargers, Audio
            { 1, new List<int> { 4, 5, 6 } },

            // Tablets -> Accessories, Cases, Chargers
            { 2, new List<int> { 4, 5, 6 } },

            // Laptops -> Accessories, Chargers
            { 3, new List<int> { 4, 6 } },

            // Accessories -> Smartphones, Tablets (for cross-selling)
            { 4, new List<int> { 1, 2 } },

            // Phone Cases -> Smartphones, Tablets
            { 5, new List<int> { 1, 2 } },

            // Chargers -> Smartphones, Tablets, Laptops
            { 6, new List<int> { 1, 2, 3 } }
        };
    }
}
}
```

100 % | 0 | 1 | Ln: 357 | Ch: 47 | SPC | CRLF


Android Emulator - Medium_Phone_API_36.0:5554
6:42
Shopping Cart
Samsung Galaxy S24 Ultra
1199.99 BAM
1199.99 BAM
1
You might also like
Fast Charger 25W
BAM
Add to Cart
Samsung S24 Ultra Case
39.99 BAM
Add to Cart
Total Items:
1
Total Amount:
1199.99 BAM
Proceed to Checkout


Android Emulator - Medium_Phone_API_36.0:5554
6:43
iPhone 15 Pro
Brand: Apple
Latest iPhone with A17 Pro chip
BAM 899.99
BAM 999.99
Stock: 50
ecommended for You
SAMSUNG
Fast Charger 25W
29.99 BAM
75
Samsung S24 Ultra Case
39.99 BAM
80
Home
Webshop
Orders
Profile

Opis sistema preporuka – Association Rules Learning

U aplikaciji **SmartPhone++** implementiran je sistem preporuka koji koristi **pravila udruženja (Association Rules Learning)**. Cilj ovog sistema je da korisnicima prikaže dodatne proizvode koji se najčešće kupuju zajedno sa artiklima koje već imaju u korpi.

Na primjer, ako korisnik doda **Samsung Galaxy S24 Ultra** u korpu, sistem može preporučiti **masku, zaštitno staklo ili punjač**, jer se ti proizvodi često kupuju zajedno. Ovakav pristup pomaže korisnicima da lakše pronađu prateću opremu, dok prodavnici donosi povećanje prodaje kroz **cross-selling**.

Logika sistema

1. **Analiza korpe** – aplikacija prepoznaje proizvode koje je korisnik dodao u korpu (npr. telefon).
2. **Pravila asocijacije** – koriste se unaprijed definisana pravila i obrasci iz baze podataka:
    - Telefoni → Maske, stakla, punjači
    - Laptopi → Torbe, punjači
    - Tableti → Tastature, futrole
3. **Preporuka proizvoda** – na osnovu podudaranja kategorija, ključnih riječi ili brenda, sistem vraća preporuke korisniku.
4. **Fallback logika** – ako se ne nađe dovoljno relevantnih proizvoda, sistem prikazuje **featured proizvode** (najprodavanije ili najpopularnije artikle).

Prednosti ovog pristupa

- **Jednostavan i efikasan** – preporuke se baziraju na pravilima koja su laka za razumjeti i proširiti.
- **Personalizovan osjećaj** – korisnik dobija preporuke direktno povezane sa artiklima u korpi.
- **Podrška cross-selling strategiji** – povećava ukupnu vrijednost narudžbe.