**Bachelor thesis**

**On the subject:**

**Development of a web application for buy and sell a Smartphones**

Student: **Yavor Iliev Ganchev**

Matriculation number: **201217033**

Subject area: **Computer science**

Supervisor of the bachelor thesis: **B.Sc. Mariana Goranova**

Konsultant der Bachelorarbeit: **Prof. Dr.-Ing. Velko Iltschev**

**Sofia**

**2020**

Увод

Глава 1 Обзор на използваните технологии и анализ на други системи  
  
Технологиите, които бяха използвани за реализирането на приложението са следните:  
  
**Visual Studio 2017** - това е работната среда (IDE) върху която беше разработен целия проект на програмния език C# и .NET технологиите към него.  
  
**Asp.Net Core** - тази технология отговаря за сървърната функционалност на приложението. Когато бива изпратен request от потребителя по HTTP проткола, сървърът определя на база на тази заявка, какъв response да върне обратно към потребителя.

**Razor Pages** – тази технология отговаря за графичния потребителски интерфейс.   
  
**Entity Framework Core** - тази технология отговаря за моделирането на базата от данни чрез подхода Code First  
  
**MS SQL Server Studio** - тази технология отговаря за управлението на информацията на физическата база данни, която е съхранена локално на този компютър

Глава 2 Проектиране на системата

* 1. Функционални изисквания

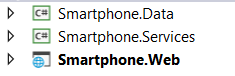
Функционалните изисквания за реализацията на приложението са да имаме първо база от данни, където ще бъде съхранена информацията за всяка една таблица.

Таблиците, които са предвидени за приложението са следните:

* Users
* Makes
* Models
* Advertisements

След това ще бъде необходима функционалност за извършване на основните операции CRUD ( Create, Read, Update, Delete ) като тези операции ще бъдат прилагани върху обявите, които потребителите ще публикуват и съответно менижират.   
  
Следваща функционалност е създаване на потребител чрез регистрационна форма и съответно да влиза в потребителски профил чрез email и парола на логин формата. За реализирането на тази функционалност беше използвана технологията Asp.Net Core Identity.  
  
Следваща функционалност е възможността на потребител да филтрира дадена обява по следните критерии: марка, модел, цена от-до, модел на телефона от-до, MP на камерата от-до, избор на рам памет и вградена памет на телефона.  
  
Следваща функционалност е когато потребителя публикува своята обява, да има възможност да прикачва снимки от компютъра си.

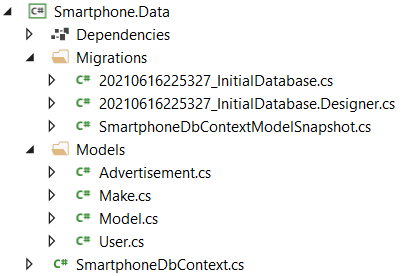
* 1. Проектиране на приложението
     1. Системна архитектура на приложението

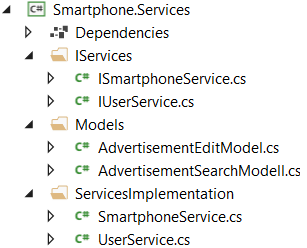
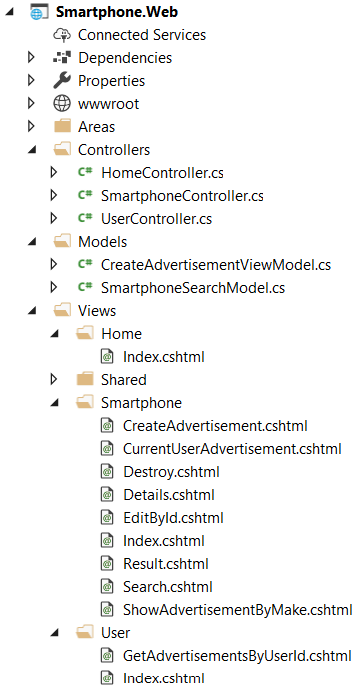
Системната архитектура на приложението беше изградена от три системни компонента, които ще бъдат описани в тази дипломна работа, като следва:  
  
  
   
  
  
  
  
1 Smartphone.Data - идеята на този компонент е да изгради първо модела на нашата база от данни чрез класове на С# в които са попълнени атрибутите и връзките между класовете (таблиците в базите данни). Наследявайки класа DbContext, получаваме пълната функционалност, която ни е необходима за да изградим нашата реална база. Накрая се създават миграции, от които се генерира чистия SQL код, въз основа на нашето моделиране на базата.. За всичко това се използва технологията Entity Framework Core, която ни дава тази функционалност да използваме.

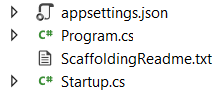
2 Smartphone.Services - идеята на този компонент е да съхрани цялата функционалност на приложението, така наречената “ бизнес логика “. За тази цел бяха създадени интерфейси, в които са декларирани функционалностите, след което се имплементират. Този компонент има връзка с първия компонент, тей като тези функционалности изискват връзка с базата данни и затова той беше рефериран. Когато създадем инстанция на нашия SmartphoneDbContext, получаваме пълната информация на нашата база данни ( таблици, записи).

3 Smartphone.Web - в този компонент е нашия сървър, който оперира цялото приложение. Въз основа на това, какво потребителя праща като заявка по HTTP протокола, нашият сървър и по-конкретно нашият Controller определя какъв отговор да изпрати към потребителя. В това приложение е използван основно SmartphoneController класа, който наследява класа Controller. По този начин ние използваме цялата функционалност, която се предоставя от ASP.NET Core. Във всеки контролер се съдържат IActionResult, и в зависимост кой IActionResult бива прехванат от контролера, ще се върне функционалността към потребителя, описана в съответния прихванат IActionResult.

* + 1. Програмна архитектура на приложението





* + 1. Алгоритми
  1. Проектиране на базата от данни
     1. ER модел на базата от данни
        1. Обекти

Обектите ще бъдат 4 както следва:  
  
обект потребител

Обект марка

Обект модел

Обект обява

* + - 1. Атрибути на обектите

Атрибути за обект потребител:  
  
1 Id - [първичен ключ]

2 Username

3 Email

4 Password

5 Phone number

6 First Name

7 Last Name

8 Address

9 City

Атрибути за обект марка:

1 Id - [първичен ключ]

2 Name - име на модела

Атрибути за обект модел:

1 Id - [първичен ключ]

2 Name

3 MakeId - [чужд ключ, сочещ към Id на таблица Makes]

4 ModelYear

5 RAM

6 Back Camera

7 Memory

Атрибути за обект телефон:

1 Id - [първичен ключ]

2 Name

3 Price

4 Description

5 Image1

6 Image2

7 MakeId - [чужд ключ] към таблица Makes

8 ModelId - [чужд ключ] към таблица Models

9 UserId- [чужд ключ] към таблица Users

* + - 1. Правила и отношения между обектите

Марка - Модел : 1 - много

Модел - Марка : 1 - 1

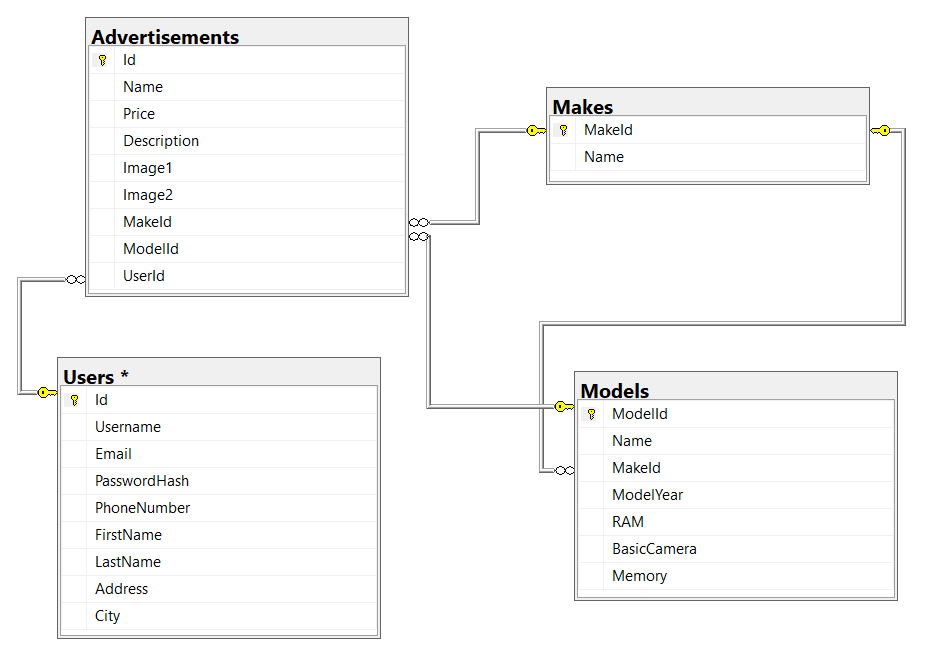
Потребител - Обява : 1 - много

Обява - Потребител : 1 – 1

Обява - Марка : 1 – 1

Обява - Модел : 1 - 1

* + 1. Проектиране на релационния модел на базата от данни

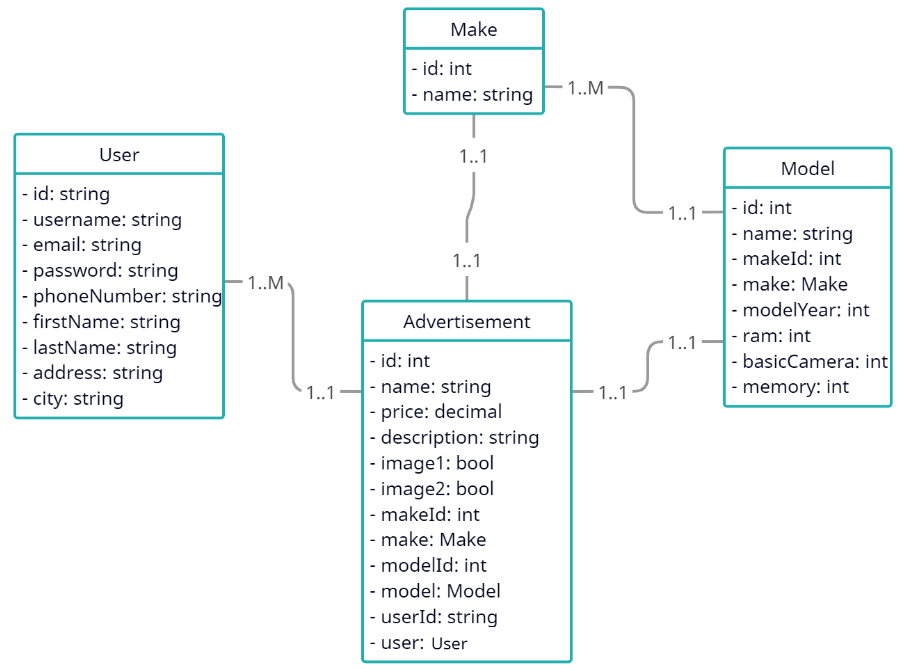


* + 1. Нормализация на релационната база от данни
  1. Проектиране на графичния потребителски интерфейс
  2. Изводи

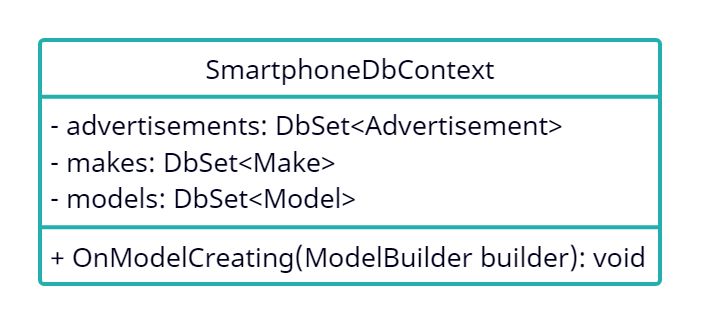
Глава 3 Програмна реализация

Клас-диаграма с описание на класовете и техните методи

3.1  
  
  
  
  
  
UML клас диаграма, показвайки кои класове какви атрибути съдържат, както и изградените връзки между класовете.



Така беше изграден модела в системния компонент Smartphone.Data. Той служи за моделиране на нашата база данни с помощта на Entity Framework Core и Code First Approach.  
  
3.2

  
  
  
using System;

using System.Collections.Generic;

using System.Text;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using Smartphone.Data.Models;

namespace Smartphone.Data

{

public class SmartphoneDbContext : IdentityDbContext<User>

{

public DbSet<Advertisement> Advertisements { get; set; }

public DbSet<Make> Makes { get; set; }

public DbSet<Model> Models { get; set; }

public SmartphoneDbContext(DbContextOptions<SmartphoneDbContext> options)

: base(options)

{

}

protected override void OnModelCreating(ModelBuilder builder)

{

base.OnModelCreating(builder);

builder.Entity<Model>()

.HasOne(e => e.Make)

.WithMany(e => e.Models)

.HasForeignKey(e => e.MakeId);

builder.Entity<Advertisement>()

.HasOne(e => e.User)

.WithMany(e => e. Advertisements)

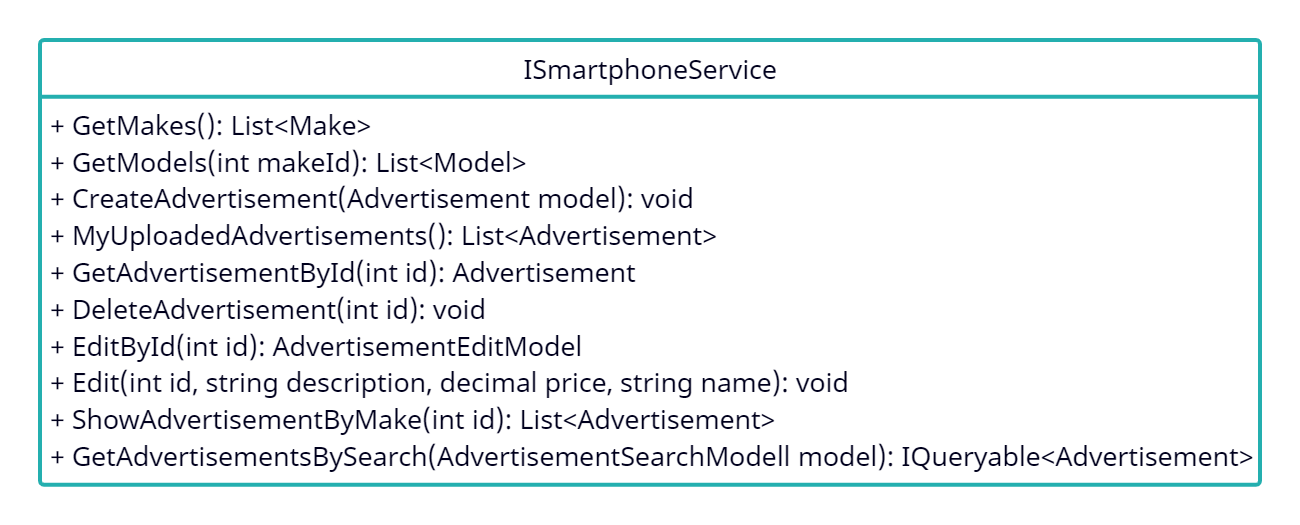
.HasForeignKey(e => e.UserId);

}

}

}

Първоначално наследяваме класът IdentityDbContext<ApplicationUser>, за да получим функционалността от Entity Framework Core, за създаване на реалната база от данни. Сетовете служат за да изградят нашите таблици в SQL. Беше използван стандартен конструктор, след което беше пренаписан метода OnModelCreating. Целта на този метод е да опише релацията между таблиците с помощта на Fluent API.   
  
3.3



В този сървис бяха декларирани функционалностите, които ще бъдат имплементирани както следва:  
  
**GetMakes()** - за получаване от базата данни колекция с всички марки  
  
**GetModels(int makeId)** - за получаване на модели от конкретна марка, която тя бъде селектирана от потребителя

**CreateAdvertisement (Advertisement model)** - за създаване на обект по конкретен модел, който бива записан в базата данни  
  
**MyUploadedAdvertisements()** - за получаване на колекция от смартфони, които текущия потребител е публикувал  
  
**GetAdvertisementById (int id)** - за получаване на дейтална информация на даден смартфон по id  
  
**DeleteAdvertisement (int id)** - за изтриване на смартфон от базата данни по даден id

**EditById(int id)** - за при избиране на текущ смартфон да му се редактират характеристиките, при зареждане на желания такъв да се генерира текущата му информация за характеристиките, която ще бъде съответно редактирана  
  
**Edit(int id, string description, decimal price, string name)** - за редактиране на запис в базата данни по зададени стойности

**ShowAdvertisementByMake (int id)** - за получаване на колекция от смартфони на база избрана марка от потребителя  
  
**GetAdvertisementsBySearch (AdvertisementSearchModell model)** - за получаване на колекция от смартфони, която е филтрирана по критерии на потребителя

using Smartphone.Data.Models;

using Smartphone.Services.Models;

using System.Collections.Generic;

using System.Linq;

namespace Smartphone.Services.IServices

{

public interface ISmartphoneService

{

List<Make> GetMakes();

List<Model> GetModels(int makeId);

void CreateAdvertisement(Advertisement model);

List<Advertisement> MyUploadedAdvertisements();

Advertisement GetAdvertisementById(int id);

void DeleteAdvertisement(int id);

AdvertisementEditModel EditById(int id);

void Edit(int id, string description, decimal price, string name);

List<Advertisement> ShowAdvertisementByMake(int id);

IQueryable<Advertisement> GetAdvertisementsBySearch(AdvertisementSearchModell model);

}

}

3.4 Следващия интерфейс,в който бяха декларирани функционалностите, които ще бъдат имплементирани

  
  
  
**GetCurrentUserPersonalData (string id)** - за получаване на лична информация относно конкретен потребител, който се определя на база подаденото id  
  
**GetAdvertisementsByUserId (string id)** - за получаване на колекция със смартфони на потребител по въведено id, която притежава   
  
using Smartphone.Data.Models;

using System.Collections.Generic;

namespace Smartphone.Services.IServices

{

public interface IUserService

{

User GetCurrentUserPersonalData(string id);

List<Advertisement> GetAdvertisementsByUserId(string id);

}

}  
  
3.5

  
  
Този имплементационен клас използва 3 обекта, които се прилагат по дизайн шаблона Dependency Injection. Това се постига като инстанцираме тези три обекта в конструктора на текущия клас SmartphoneService. По този начин когато направим инстанция на класа SmartphoneService, тогава ще се инстанцират и останалите зависими обекти. Така печелим оптимизация на паметта. Сега ще бъдат обеснени подробно всеки един метод какво прави.

public class SmartphoneService : ISmartphoneService

{

private readonly SmartphoneDbContext db;

private readonly IHttpContextAccessor http;

private readonly IHostingEnvironment env;

public SmartphoneService(IHostingEnvironment env, IHttpContextAccessor http, SmartphoneDbContext db)

{

this.db = db;

this.http = http;

this.env = env;

}

}

**CreateAdvertisement (Advertisement model)** - този метод има за цел да създаде нова обява от потребителя в базата данни. Това се случва по следния начин: Когато потребителя въведе цялата информация за неговата обява на страницата за създаване и изпрати заявката си към сървъра, той всъщност изпраща дадения модел. Затова нашия метод получава този модел и започва работата по него.

Съответно създаваме си един локален обект, на когото атрибути задаваме стойностите, които идват от модела. Както следва от кода например:   
Price = model.Price ; това означава, че лявата част е атрибута на нашия обект, който създаваме вмомента. Дясната част е стойността, която идва от потребителя по модела, която бива задавана към нашия локален обект. Накрая добавяме към нашата таблица нашия текущ обект и записваме промените. По този начин записваме нов запис в нашата база данни.

public void CreateAdvertisement(Advertisement model)

{

var ad = new Advertisement

{

Price = model.Price,

Model = model.Model,

Make = model.Make,

User = model.User,

Image1 = model.Image1,

Image2 = model.Image2,

Description = model.Description,

MakeId = model.MakeId,

ModelId = model.ModelId,

Name = model.Name,

UserId = http.HttpContext.User.FindFirstValue(ClaimTypes.NameIdentifier)

};

db.Advertisements.Add(ad);

db.SaveChanges();

}

**GetMakes()** - този метод има за цел да достъпи базата данни и по - конкретно таблицата Makes. Селектирват се колоните ID и Name и тази информация се съхранява в списък, който се връща като резултат. Самата идея на метода е, той да бъде приложен за Dropdown меню функцтионалността.

public List<Make> GetMakes()

{

var makes = db.Makes.Select(e=>new Make

{

MakeId = e.MakeId,

Name = e.Name

}).ToList();

return makes;

}

**GetModels(int makeId)** - този метод има за цел да достъпи таблицата Models в нашата база данни. Тей като искаме да селектираме даден модел, той принадлежи съответно на дадена марка. Поради тази причина метода изисква да му бъде подаден параметър, който е Id на марката. Съответно нашата заявка към базата ще има WHERE клауза с условие, Id от таблицата Models да съвпадне със същото такова от таблица Makes. Както и в предния метод, селектирваме само ID и Name колоните, защото този метод ще бъде използван само за функционалността за Dropdown менюто. На база селектирана марка, ще се селектират конкретните модели, които принадлежат към въпросната марка.

public List<Model> GetModels(int MakeId)

{

var models = db.Models.Where(

e => e.Make.MakeId == MakeId).Select(e=>new Model

{

ModelId = e.ModelId,

Name = e.Name

}).OrderBy(e => e.Name).ToList();

models.Insert(0, new Model { ModelId = 0, Name = "Select Model" });

return models;

}

**MyUploadedAdvertisements()** - този метод има за цел да достъпи таблицата Advertisements и в зависимост кой е логнатия потребител, да бъдат селектирани неговите публикувани обяви и да се върнат в списък като резултат. Това се реализирва с WHERE клауза на която казваме Id на текущия потребител да съвпадне с таблицата Advertisements колоната за UserId. Накрая извлечената информация се записва в списък, който се връща като резултат

public List<Advertisement> MyUploadedAdvertisements()

{

var advertisements = db

.Advertisements

.Where(e => e.UserId == http.HttpContext.User.FindFirstValue(ClaimTypes.NameIdentifier))

.Select(e => new Advertisement

{

Id = e.Id,

Name = e.Name,

Make = e.Make,

Model = e.Model,

Price = e.Price

})

.ToList();

return advertisements;

}

**GetAdvertisementById(int id) -** този метод има за цел да достъпи таблицата Advertisements и да селектира цялата информация от нея. Идеята е всяка обява да реферира към по-детайлна информация за самата нея. Съответно тази референция идва от подадения параметър Id.

public Advertisement GetAdvertisementById(int id)

{

var ad = db.Advertisements.Select(e=>new Advertisement

{

Name = e.Name,

Description = e.Description,

Price = e.Price,

Id = e.Id,

Image1 = e.Image1,

Image2 = e.Image2,

Make = e.Make,

MakeId = e.MakeId,

Model = e.Model,

ModelId = e.ModelId,

UserId = e.UserId,

User = e.User

}).FirstOrDefault(e => e.Id == id);

return ad;

}

**DeleteAdvertisement(int id)** - този метод има за цел да изтрие конкретна обява по Id, което се подава като параметър на този метод. Първо се проверява дали съществува такава обява с подаденото Id. В случай, че да я изтриваме от базата данни и също така изтриваме и прикачените файлове към обявата.

public void DeleteAdvertisement(int id)

{

var ad = db.Advertisements.Find(id);

if (ad == null)

{

return;

}

System.IO.DirectoryInfo di = new DirectoryInfo

(env.WebRootPath + "/files/" +

http.HttpContext.User.FindFirstValue(ClaimTypes.NameIdentifier)

+ "/"

+ ad.MakeId + "/" + ad.ModelId);

foreach (FileInfo file in di.GetFiles())

{

file.Delete();

}

di.Delete();

db.Remove(ad);

db.SaveChanges();

System.IO.DirectoryInfo dii = new DirectoryInfo

(env.WebRootPath + "/files/" +

http.HttpContext.User.FindFirstValue(ClaimTypes.NameIdentifier)

+ "/"

+ ad.MakeId);

if(dii.GetDirectories().Length==0)

{

dii.Delete();

}

}

**EditById(int id)** - този метод има за цел да генерира информацията за атрибутите на обявата когато се отвори страницата за редактиране. Така потребителя може по-лесно да редактира своята обява. Атрибутите, които ще могат да се редактират са: Name, Price, Description.

public AdvertisementEditModel EditById(int id)

=> db

.Advertisements

.Where(e => e.Id == id)

.Select(a => new AdvertisementEditModel

{

Name = a.Name,

Price = a.Price,

Description = a.Description

}).FirstOrDefault();

**Edit(int id, string description, decimal price, string name)** - този метод има за цел да извършва манипулация в базата данни като редактира информацията на записите. Това се случва като първо намерим по Id обявата, която ще искаме да редактираме. След което задаваме новите стойности на намерения обект, които идват от модела. Самия модел се изпраща от потребителя към сървъра в страницата за Edit.

public void Edit(int id, string description, decimal price, string name)

{

var ad = db.Advertisements.Find(id);

if (ad == null)

{

return;

}

ad.Description = description;

ad.Price = price;

ad.Name = name;

db.SaveChanges();

}

**ShowAdvertisementByMake(int id)** - този метод има за цел да покаже всички обяви от избрана марка. Затова метода приема като параметър Id , тъй като в него се посочва от коя марка да се извлекат обявите.

public List<Advertisement> ShowAdvertisementByMake(int id)

{

var ad = db.Advertisements.Where(e => e.Make.MakeId == id)

.Select(e => new Advertisement

{

Id = e.Id,

Name = e.Name,

Make = e.Make,

Model = e.Model,

Price = e.Price

})

.ToList();

return ad;

}

**GetAdvertisements(AdvertisementSearchModell model)** - този метод има за цел да върне като резултат тези обяви, които са заявени през филтъра на страницата Search от потребителя.

public IQueryable<Advertisement> GetAdvertisements(AdvertisementSearchModell model)

{

var result = db.Advertisements.AsQueryable();

if (model.MakeId.HasValue)

{

result = result.Where(e => e.Make.MakeId == model.MakeId).Select(e => new Advertisement

{

Id = e.Id,

Name = e.Name,

Price = e.Price,

Model = e.Model,

Make = e.Make

});

}

if (model.ModelId != 0 && model.ModelId.HasValue)

{

result = result.Where(e => e.Model.ModelId == model.ModelId).Select(e => new Advertisement

{

Id = e.Id,

Name = e.Name,

Price = e.Price,

Model = e.Model,

Make = e.Make

});

}

if (model.MemoryAreChecked != null && model.RamAreChecked != null)

{

result = result.Where(e => model.MemoryAreChecked.Contains(e.Model.Memory) && model.RamAreChecked.Contains(e.Model.RAM)).Select(e => new Advertisement

{

Id = e.Id,

Name = e.Name,

Price = e.Price,

Model = e.Model,

Make = e.Make

});

}

else if (model.RamAreChecked != null)

{

result = result.Where(e => model.RamAreChecked.Contains(e.Model.RAM)).Select(e => new Advertisement

{

Id = e.Id,

Name = e.Name,

Price = e.Price,

Model = e.Model,

Make = e.Make

});

}

else if (model.MemoryAreChecked != null)

{

result = result.Where(e => model.MemoryAreChecked.Contains(e.Model.Memory)).Select(e => new Advertisement

{

Id = e.Id,

Name = e.Name,

Price = e.Price,

Model = e.Model,

Make = e.Make

});

}

if (model.PriceFrom.HasValue)

{

result = result.Where(e => e.Price >= model.PriceFrom).Select(e => new Advertisement

{

Id = e.Id,

Name = e.Name,

Price = e.Price,

Model = e.Model,

Make = e.Make

});

}

if (model.ModelYearFrom.HasValue)

{

result = result.Where(e => e.Model.ModelYear >= model.ModelYearFrom).Select(e => new Advertisement

{

Id = e.Id,

Name = e.Name,

Price = e.Price,

Model = e.Model,

Make = e.Make

});

}

if (model.ModelYearTo.HasValue)

{

result = result.Where(e => e.Model.ModelYear <= model.ModelYearTo).Select(e => new Advertisement

{

Id = e.Id,

Name = e.Name,

Price = e.Price,

Model = e.Model,

Make = e.Make

});

}

if (model.BasicCameraFrom.HasValue)

{

result = result.Where(e => e.Model.BasicCamera >= model.BasicCameraFrom).Select(e => new Advertisement

{

Id = e.Id,

Name = e.Name,

Price = e.Price,

Model = e.Model,

Make = e.Make

});

}

if (model.BasicCameraTo.HasValue)

{

result = result.Where(e => e.Model.BasicCamera <= model.BasicCameraTo).Select(e => new Advertisement

{

Id = e.Id,

Name = e.Name,

Price = e.Price,

Model = e.Model,

Make = e.Make

});

}

if (model.PriceTo.HasValue)

{

result = result.Where(e => e.Price <= model.PriceTo).Select(e => new Advertisement

{

Id = e.Id,

Name = e.Name,

Price = e.Price,

Model = e.Model,

Make = e.Make

});

}

if (!(model.PriceFrom.HasValue) && !(model.PriceTo.HasValue)

&& !(model.MakeId.HasValue) && model.RamAreChecked == null && model.MemoryAreChecked == null

&& !(model.BasicCameraFrom.HasValue) && !(model.BasicCameraTo.HasValue)

&& !(model.ModelYearFrom.HasValue) && !(model.ModelYearTo.HasValue))

{

result = result.Select(e => new Advertisement

{

Id = e.Id,

Name = e.Name,

Price = e.Price,

Model = e.Model,

Make = e.Make

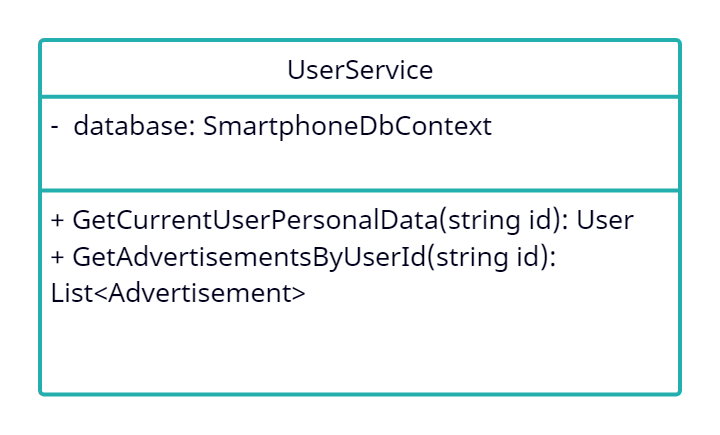
});

}

return result;

}

3.6



Този имплементационен клас използва 1 обект, който се инстанцира в конструктора на класа по дизайн шаблона Dependency Injection. Този обект е нашата база от данни, от която ще вземаме информация.

**GetCurrentUserPersonalData(string id)** - целта на този метод е да извлече данните от таблицата с потребители, като търсения потребител се определя на база подадения параметър id в метода.

public User GetCurrentUserPersonalData(string id)

{

var user = database.Users

.Where(e=>e.Id == id)

.Select(e=>new User

{

City = e.City,

FirstName = e.FirstName,

LastName = e.LastName,

PhoneNumber = e.PhoneNumber,

UserName = e.UserName,

Id = e.Id

}).FirstOrDefault();

return user;

}

**GetAdvertisementsByUserId(string id)** - целта на този метод е да извлече списък с обявите, които даден потребител е публикувал, като дадения потребител се определя по зададения параметър id.

public List<Advertisement> GetAdvertisementsByUserId(string id)

{

var advertisements = database.Advertisements.Where(e => e.UserId == id).Select

(e => new Advertisement

{

Id = e.Id,

Name = e.Name,

Make = e.Make,

Model = e.Model,

Price = e.Price,

}).ToList();

return advertisements;

}

3.6

Това са клас-моделите, които бяха използвани за попълване на атрибутите на обектите.

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

namespace Smartphone.Services.Models

{

public class AdvertisementSearchModell

{

public int? Id { get; set; }

public int? MakeId { get; set; }

public int? ModelId { get; set; }

public int? BasicCameraFrom { get; set; }

public int? BasicCameraTo { get; set; }

public decimal? PriceFrom { get; set; }

public decimal? PriceTo { get; set; }

public int? ModelYearFrom { get; set; }

public int? ModelYearTo { get; set; }

[BindProperty]

public List<int> RamAreChecked { get; set; }

[BindProperty]

public List<int> MemoryAreChecked { get; set; }

}

}

namespace Smartphone.Services.Models

{

public class AdvertisementEditModel

{

public string Name { get; set; }

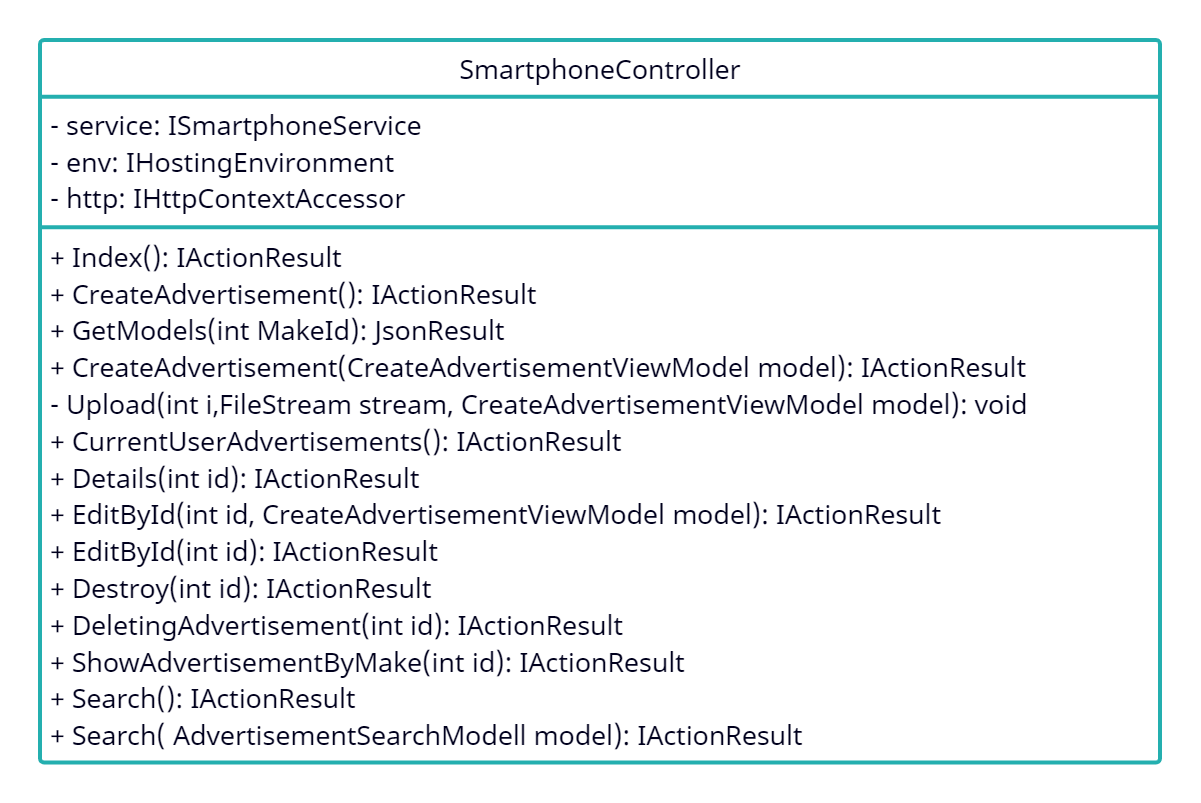
public decimal Price { get; set; }

public string Description { get; set; }

}

}

3.7 Следва да разгледаме системния компонент Smartphone.Web. Тук се намира нашия сървър, който комуникира с потребителя по HTTP протокола. Той се базира на архитектурата MVC (Model-View-Controller). Контролерър е компонента, който приема заявката от потребителя и му връща съответния отговор. Начинът по който работи контролера е следния: имаме настроен router, който прихваща от URL заявката по следния шаблон ( controller/action/id? ) Action-a е името на нашите методи, които сме създали и които изпълняват функционалностите. Важно е името на метода да съвпадне с името на View файла. Само тогава ще се отвори конкретното View. Сега следва да бъдат разгледани:

  
Това са обектите, които са необходими на нашия сървър, с които той ще работи и използва функционалност. Те се инстанцират през конструктора на класа.

private readonly ISmartphoneService service;

private readonly IHostingEnvironment env;

private readonly IHttpContextAccessor http;

public SmartphoneController(IHttpContextAccessor http,ISmartphoneService service,IHostingEnvironment env)

{

this.service = service;

this.env = env;

this.http = http;

}

[Authorize]  
**CreateAdvertisement()** - този метод има за цел да върни View компонент, който представлява страницата за попълване на информация за обява на смартфон. Тей като потребителя ще има възможност от Dropdown меню да избира своята марка и модел на смартфона, използваме нашата Service функционалност “GetMakes" за да извлечем данните от таблицата Марки. Важно е да се отбележи, че този метод е маркиран като Authorize, това означава, че само потребители, които са влезнали в акаунта си ще могат да достъпят тази страница.

[Authorize]

public IActionResult CreateAdvertisement()

{

var makes = service.GetMakes();

ViewBag.Makes = makes;

return View();

}

**Index()** - този метод има за цел да отвори заглавната страница на приложението

public IActionResult Index()

{

return View();

}

**GetModels(int MakeId)** - този метод има за цел да попълни Dropdown листа за моделите, като те се генерират на база подадено id на марката.

public JsonResult GetModels(int MakeId)

{

var models = service.GetModels(MakeId);

return Json(new SelectList(models, "ModelId", "Name"));

}

**CreateAdvertisement(CreateAdvertisementViewModel model)** - този метод има за цел да създаде запис в базата данни, който представлява обява на смартфон. Също така има за цел да запише файловете на сървъра, които са прикачени от локалния компютър на потребителя. Той е с POST заявка.

[HttpPost]

public IActionResult CreateAdvertisement(CreateAdvertisementViewModel model)

{

bool[] array = new bool[6];

if (model.Image1 != null)

{

array[1] = true;

}

if (model.Image2 != null)

{

array[2] = true;

}

string userId = http.HttpContext.User.FindFirstValue(ClaimTypes.NameIdentifier);

int makeId = model.MakeId;

int modelId = model.ModelId;

string path = env.WebRootPath + "/files/";

bool existUserFolder = System.IO.Directory.Exists(path + userId);

if(!existUserFolder)

{

var folderpath = path + userId + "/";

System.IO.Directory.CreateDirectory(folderpath);

bool existMakeFolder = System.IO.Directory.Exists(folderpath + makeId);

if(!existMakeFolder)

{

var folderpathh = folderpath + makeId + "/";

System.IO.Directory.CreateDirectory(folderpathh);

bool existModelFolder = System.IO.Directory.Exists(folderpathh + modelId);

if(!existModelFolder)

{

var folderpathhh = folderpathh + modelId + "/";

System.IO.Directory.CreateDirectory(folderpathhh);

for (int i = 1; i <= 5; i++)

{

if (array[i])

{

var file = folderpathhh + i + ".jpg";

using (var stream = new FileStream(file, FileMode.Create))

{

Upload(i, stream, model);

}}

} }}

}else

{

var folderpath = path + userId + "/";

bool existMakeFolder = System.IO.Directory.Exists(folderpath + makeId);

if (existMakeFolder)

{

var folderpathh = folderpath + makeId + "/";

bool existModelFolder = System.IO.Directory.Exists(folderpathh + modelId);

if (!existModelFolder)

{

var folderpathhh = folderpathh + modelId + "/";

System.IO.Directory.CreateDirectory(folderpathhh);

for (int i = 1; i <= 5; i++)

{

if (array[i])

{

var file = folderpathhh + i + ".jpg";

using (var stream = new FileStream(file, FileMode.Create))

{

Upload(i, stream, model);

}

}

}

}

} else

{

var folderpathh = folderpath + makeId + "/";

System.IO.Directory.CreateDirectory(folderpathh);

bool existModelFolder = System.IO.Directory.Exists(folderpathh + modelId);

if (!existModelFolder)

{

var folderpathhh = folderpathh + modelId + "/";

System.IO.Directory.CreateDirectory(folderpathhh);

for (int i = 1; i <= 5; i++)

{

if (array[i])

{

var file = folderpathhh + i + ".jpg";

using (var stream = new FileStream(file, FileMode.Create))

{

Upload(i, stream, model);

}

}

}

} else

{

var folderpathhh = folderpathh + modelId + "/";

var file = folderpathhh + "1.jpg";

using (var stream = new FileStream(file, FileMode.Create))

{

}

}

}

}

var ad = new Advertisement

{

MakeId = model.MakeId,

Image1 = array[1],

Image2 = array[2],

ModelId = model.ModelId,

Price = model.Price,

Description = model.Description,

Name = model.Name

};

service.CreateAdvertisement(ad);

return RedirectToAction(nameof(CurrentUserAdvertisements));

}

**Upload(int i,FileStream stream, CreateAdvertisementViewModel model)** - този метод е помещен за класа, който служи в горния метод за прикачване на файловете

private void Upload(int i,FileStream stream, CreateAdvertisementViewModel model)

{

switch (i)

{

case 1:

model.Image1.CopyTo(stream);

break;

case 2:

model.Image2.CopyTo(stream);

break;

default: break;

}

}

**CurrentUserAdvertisements()** - този метод има за цел да извлече на потребителя неговите публикувани обяви ако има такива.

public IActionResult CurrentUserAdvertisements()

{

var advertisements = service.MyUploadedAdvertisements();

return View(advertisements);

}

**Details(int id)** - този метод има за цел да отвори страницата, върху която ще се изобрази детайлната информация за дадена обява, която се определя по зададения параметър id.

public IActionResult Details(int id)

{

var advertisement = service.GetAdvertisementById(id);

if (advertisement == null)

{

return NotFound();

}

return View(advertisement);

}

**EditById(int id, CreateAdvertisementViewModel model)** - този метод има за цел да манипулира информацията на даден запис в базата данни. Предава се с POST заявка.

[HttpPost]

public IActionResult EditById(int id, CreateAdvertisementViewModel model)

{

service.Edit(id, model.Description, model.Price, model.Name);

return RedirectToAction(nameof(CurrentUserAdvertisements));

}

**EditById(int id)** - този метод има за цел когато потребителя кликне на опцията Edit на публикувана от него обява в менюто My Advertisements да му се зареди страницата с избрания запис и да се генерира информацията за име, цена и описание на обявата.

public IActionResult EditById(int id)

{

var advertisement = service.EditById(id);

if(advertisement == null)

{

return NotFound();

}

return View(new CreateAdvertisementViewModel

{

Name = advertisement.Name,

Price = advertisement.Price,

Description = advertisement.Description

});

}

**Destroy(int id)** - този метод има за цел да покаже View на потребителя, на което той да потвърди, че желае да изтрие избраната обява.

public IActionResult Destroy(int id) => View(id);

**DeletingAdvertisement(int id)** - този метод има за цел да изтрие избрания запис от потребителя от базата данни.

public IActionResult DeletingAdvertisement(int id)

{

service.DeleteAdvertisement(id);

return RedirectToAction(nameof(Index));

}

**ShowAdvertisementByMake(int id)** - този метод има за цел да покаже View на потребителя, на което може да види всички обяви, които са с марка, която той е избрал по id от началната страница на приложението

public IActionResult ShowAdvertisementByMake(int id)

{

var advertisements = service.ShowAdvertisementByMake(id);

return View(advertisements);

}

**Search()** - този метод има за цел да покаже View на потребителя, на което ще му се покажат опциите, по които той може да търси желан от него смартфон.

[Authorize]

public IActionResult Search()

{

var makes = service.GetMakes();

ViewBag.Makes = makes;

return View();

}

**Search( AdvertisementSearchModell model)** - този метод има за цел да изпрати модел до базата данни, който описва какво желае потребителя да извлече от базата данни таблицата Advertisements.

public IActionResult Search( AdvertisementSearchModell model)

{

var advertisements = service.GetAdvertisementsBySearch(model);

return View("Result", advertisements);

}

Сега ще бъдат представени View на приложението. Това са Razor Pages .cshtml. Благодарение на тази технология имаме възможността да генерираме front-end код и C# код в един файл за представяне на информацията визуално.

**CreateAdvertisement** - целта на тази страница е да нарисува формата, по която потребителя ще попълне информация за обявата на смартфона

@model CreateAdvertisementViewModel

@{

ViewData["Title"] = "CreateAdvertisement";

}

<form method="post" enctype="multipart/form-data">

<select id="MakeId" asp-for="MakeId" asp-items='@(new SelectList(ViewBag.Makes,"MakeId","Name"))'>

<option>Select Make</option>

</select> <select id="ModelId" asp-for="ModelId" asp-items='@(new SelectList(string.Empty,"ModelId","Name"))'></select>

<script src="~/lib/jquery/dist/jquery.js"></script>

<script>

$(function () {

$("#MakeId").change(function () {

var url = "/Smartphone/GetModels";

var ddlsource = "#MakeId";

$.getJSON(url, { MakeId: $(ddlsource).val() }, function (data) {

var items = '';

$("#ModelId").empty();

$.each(data, function (i, row) {

items += "<option value='" + row.value + "'>" + row.text + "</option>";

});

$("#ModelId").html(items);

})

});

});

</script>

<div

<label asp-for="Name"></label>

<input asp-for="Name" />

</div>

<div

<label asp-for="Price"></label>

<input asp-for="Price" />

</div>

<div>

<label asp-for="Image1"></label>

<input type="file" asp-for="Image1" />

</div>

<label asp-for="Image2"></label>

<input type="file" asp-for="Image2" />

<label asp-for="Description"></label>

<textarea asp-for="Description"> </textarea>

<input type="submit" value="Create" />

</form>

**CurrentUserAdvertisement** - целта на тази страница е да нарисува таблица, която да изобрази публикуваните обяви на текущия потребител

@using Smartphone.Data.Models

@model List<Advertisement>

@{

ViewData["Title"] = "MySmartphones";

}

<table class="table table-bordered table-hover">

<thead>

<tr>

<th>

Name

</th>

<th>

Price

</th>

<th>

Make

</th>

<th>

Model

</th>

<th>

Option

</th>

</tr>

</thead> <tbody>

@foreach (var item in Model)

{

<tr>

<th>

@item.Name

</th>

<th>

@item.Price

</th>

<th>

@item.Make.Name

</th>

<th>

@item.Model.Name

</th>

<th>

<a href="/smartphone/details/@item.Id">Check</a>

<a href="/smartphone/editbyid/@item.Id" class="btn-warning">Edit</a>

<a href="/smartphone/destroy/@item.Id" class="btn-danger">Delete</a>

</th>

</tr>

}

</tbody>

</table>

**Destroy** - целта на тази страница е да направи Confirmation page, на която потребителя да потвърди изтриването на избраната от него обява

@model int

@{

ViewData["Title"] = "Destroy";

}<h2>Are you sure, you want to delete this Advertisement?</h2>

<a href="/smartphone/DeletingAdvertisement/@Model" class="btn btn-danger">Confirm Delete</a>

<a href="/smartphone/currentuseradvertisement" class="btn btn-default">Back</a>

**Details** - целта на тази страница е да визуализира цялата информация за избраната обява като например: характеристиките на смартфона, потребителя, който е публикувал тази обява

@using Smartphone.Data.Models

@model Advertisement

@{

ViewData["Title"] = "Details";

}

<h1>Name - @Model.Name</h1>

<h2>Price - @Model.Price</h2>

<h3>Description:</h3>

<h4>@Model.Description</h4>

<h5>Model Year - @Model.Model.ModelYear</h5>

<h5>Ram - @Model.Model.RAM GB</h5>

<h5>Memory - @Model.Model.Memory GB</h5>

<h5>Back Camera - @Model.Model.BasicCamera</h5>

@if (Model.Image1 == true)

{

<img src="~/files/@Model.UserId/@Model.MakeId/@Model.ModelId/1.jpg" width="300" height="300"/>

}

@if (Model.Image2 == true)

{

<img src="~/files/@Model.UserId/@Model.MakeId/@Model.ModelId/2.jpg" width="300" height="300"/>

}<h1>Contact:</h1>

<div>

@Model.User.FirstName

@Model.User.LastName

@Model.User.PhoneNumber

<a href="/user/index/@Model.UserId" >Check profile of @Model.User.UserName</a>

</div>

**EditById** - целта на тази страница е да изпрати POST заявка към сървъра и да направи промяна на записа в базата данни, тоест на избраната обява от потребителя

@model CreateAdvertisementViewModel

@{

ViewData["Title"] = "Edit";

}

<form method="post">

<div

<label asp-for="Name"></label>

<input asp-for="Name" />

</div>

<div

<label asp-for="Price"></label>

<input asp-for="Price" />

</div>

<label asp-for="Description"></label>

<textarea asp-for="Description"> </textarea>

<input type="submit" value="sub" />

</form>

Index - това е началната страница на приложението. На тази страница потребителя може да кликне върху някоя марка и да прегледа какви обяви има публикувани на въпросната марка

@{

ViewData["Title"] = "Home Page";

}

<a href="/smartphone/showadvertisementbymake/1">

<img border="0" src="~/images/samsungimg.jpg" width="100" height="100">

</a>

<a href="/smartphone/showadvertisementbymake/2">

<img border="0" src="~/images/appleimg.jpg" width="80" height="80">

</a>

<a href="/smartphone/showadvertisementbymake/3">

<img border="0" src="~/images/huaweiimg.jpg" width="80" height="80">

</a>

<a href="/smartphone/showadvertisementbymake/4">

<img border="0" src="~/images/nokiaimg.jpg" width="80" height="80">

</a>

<a href="/smartphone/showadvertisementbymake/5">

<img border="0" src="~/images/xiaomiimg.jpg" width="80" height="80">

</a>

<a href="/smartphone/showadvertisementbymake/6">

<img border="0" src="~/images/sonyimg.jpg" width="80" height="80">

</a>

<a href="/smartphone/showadvertisementbymake/7">

<img border="0" src="~/images/motorolaimg.jpg" width="80" height="80">

</a>

<a href="/smartphone/showadvertisementbymake/8">

<img border="0" src="~/images/htcimg.jpg" width="100" height="100">

</a>

<a href="/smartphone/showadvertisementbymake/9">

<img border="0" src="~/images/lgimg.jpg" width="120" height="100">

</a>

<a href="/smartphone/showadvertisementbymake/11">

<img border="0" src="~/images/asusimg.jpg" width="100" height="100">

</a>

<a href="/smartphone/showadvertisementbymake/10">

<img border="0" src="~/images/alcatelimg.jpg" width="100" height="100">

</a>

**Result** - тази страница има за цел да визуализира таблица с обяви, които потребителя в страницата Search е попълнил като характеристики, затова какви точно обяви търси.

@using Smartphone.Data.Models

@model IQueryable<Advertisement>

@{

ViewData["Title"] = "MySmartphones";

}

@if (Model.Count() == 0)

{

<h1> Not Found </h1>

}

else

{

<table class="table table-bordered table-hover">

<thead>

<tr>

<th>

Name

</th>

<th>

Price

</th>

<th>

Make

</th>

<th>

Model

</th>

<th>

Option

</th>

</tr>

</thead>

<tbody> @foreach (var item in Model)

{

<tr>

<th>

@item.Name

</th>

<th>

@item.Price

</th>

<th>

@item.Make.Name

</th>

<th>

@item.Model.Name

</th>

<th>

<a href="/smartphone/details/@item.Id">Check</a>

</th>

</tr>

}

</tbody>

</table>

}

**Search** - тази страница има за цел да изобрази възможностите за попълване на характеристики за смартфон от потребителя

@using Smartphone.Services.Models

@model AdvertisementSearchModell

@{

ViewData["Title"] = "Search";

}

<form method="post">

<label asp-for="PriceFrom">Price From</label>

<input asp-for="PriceFrom" />

<label asp-for="PriceTo">Price To</label>

<input asp-for="PriceTo" />

<label asp-for="ModelYearFrom">Model year from</label>

<input asp-for="ModelYearFrom" placeholder="2005" />

<label asp-for="ModelYearTo">Model year to</label>

<input asp-for="ModelYearTo" placeholder="2021" />

<label asp-for="BasicCameraFrom">Basic camera (MP) from</label>

<input asp-for="BasicCameraFrom" />

<label asp-for="BasicCameraTo">Basic camera (MP) to</label>

<input asp-for="BasicCameraTo" />

<select id="MakeId" asp-for="MakeId" asp-items='@(new SelectList(ViewBag.Makes,"MakeId","Name"))'>

<option>Select Make</option>

</select>

<select id="ModelId" asp-for="ModelId" asp-items='@(new SelectList(string.Empty,"ModelId","Name"))'></select>

<h3>RAM:</h3>

<div>

<input name="RamAreChecked" type="checkbox" value="1" />

<label>1 GB</label>

</div>

<div> <input name="RamAreChecked" type="checkbox" value="2" />

<label>2 GB</label>

</div>

<div>

<input name="RamAreChecked" type="checkbox" value="4" />

<label>4 GB</label>

</div>

<div>

<input name="RamAreChecked" type="checkbox" value="6" />

<label>6 GB</label>

</div>

<div>

<input name="RamAreChecked" type="checkbox" value="8" />

<label>8 GB</label>

</div>

<h3>Memory:</h3>

<div>

<input name="MemoryAreChecked" type="checkbox" value="4" />

<label>4 GB</label>

</div>

<div>

<input name="MemoryAreChecked" type="checkbox" value="6" />

<label>6 GB</label>

</div>

<div>

<input name="MemoryAreChecked" type="checkbox" value="8" />

<label>8 GB</label>

</div>

<div>

<input name="MemoryAreChecked" type="checkbox" value="16" />

<label>16 GB</label>

</div>

<div>

<input name="MemoryAreChecked" type="checkbox" value="32" />

<label>32 GB</label>

</div>

<div>

<input name="MemoryAreChecked" type="checkbox" value="64" />

<label>64 GB</label>

</div>

<div>

<input name="MemoryAreChecked" type="checkbox" value="128" />

<label>128 GB</label>

</div>

<div>

<input name="MemoryAreChecked" type="checkbox" value="256" />

<label>256 GB</label>

</div>

<div>

<input name="MemoryAreChecked" type="checkbox" value="512" />

<label>512 GB</label>

</div>

<script src="~/lib/jquery/dist/jquery.js"></script>

<script>

$(function () {

$("#MakeId").change(function () {

var url = "/Smartphone/GetModels";

var ddlsource = "#MakeId";

$.getJSON(url, { MakeId: $(ddlsource).val() }, function (data) {

var items = '';

$("#ModelId").empty();

$.each(data, function (i, row) {

items += "<option value='" + row.value + "'>" + row.text + "</option>";

});

$("#ModelId").html(items);

})

});

});

</script>

<input type="submit" value="sub" />

</form>

**ShowAdvertisementByMake** - тази страница има за цел да визуализира таблица с обяви по зададена марка

@using Smartphone.Data.Models

@model List<Advertisement>

@{

ViewData["Title"] = "MyAdvertisements";

}<table class="table table-bordered table-hover">

<thead>

<tr>

<th>

Name

</th>

<th>

Price

</th>

<th>

Make

</th>

<th>

Model

</th>

<th>

Option

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<th>

@item.Name

</th>

<th>

@item.Price

</th>

<th>

@item.Make.Name

</th>

<th>

@item.Model.Name

</th>

<th>

<a href="/smartphone/details/@item.Id">Check</a>

</th>

</tr>

}

</tbody>

</table>

**GetAdvertisementsByUserId** - този метод има за цел да визуализира обявите на избран потребител

@using Smartphone.Data.Models

@model List<Advertisement>

@{

ViewData["Title"] = "Advertisements";

}

<table class="table table-bordered table-hover">

<thead>

<tr>

<th>

Name

</th>

<th>

Price

</th>

<th>

Make

</th>

<th>

Model

</th>

<th>

Option

</th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr> <th>

@item.Name

</th>

<th>

@item.Price

</th>

<th>

@item.Make.Name

</th>

<th>

@item.Model.Name

</th>

<th>

<a href="/smartphone/details/@item.Id">Check</a>

</th>

</tr>

}

</tbody>

</table>

**UserInfo** - тази страница има за цел да визуализира информацията на потребителя

@using Smartphone.Data.Models

@model User

@{

ViewData["Title"] = "Index";

}

<a href="/User/GetSmartphonesByUserId/@Model.Id" >Check @Model.UserName's Smartphones</a>

@Model.City

@Model.FirstName

@Model.LastName

@Model.PhoneNumber

\_**Layout** - това е Layout на приложението, което е общо и се вижда от всяка една страница

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>@ViewData["Title"] - Smartphone</title>

<environment include="Development">

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />

<link rel="stylesheet" href="~/css/site.css" />

</environment>

<environment exclude="Development">

<link rel="stylesheet" href="https://d1zq6pyjrqicyj.cloudfront.net/bnNJetTfdHsQ/MjrwgQJtoYUV/bootstrap.min.css"

asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"

asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-value="absolute" />

<link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />

</environment>

</head>

<body>

<nav class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav"> <li><a asp-area="" asp-controller="Home" asp-action="Index">Home</a></li>

<li><a asp-area="" asp-controller="Smartphone" asp-action="CreateAdvertisement">Add Smartphone</a></li>

<li><a asp-controller="Smartphone" asp-action="Search">Search</a></li> </ul>

<partial name="\_LoginPartial" />

</div>

</div>

</nav> <partial name="\_CookieConsentPartial" />

<div class="container body-content">

@RenderBody()

<hr />

<footer></footer>

</div>

<environment include="Development">

<script src="~/lib/jquery/dist/jquery.js"></script>

<script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>

<script src="~/js/site.js" asp-append-version="true"></script>

</environment>

<environment exclude="Development">

<script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.3.1.min.js"

asp-fallback-src="~/lib/jquery/dist/jquery.min.js"

asp-fallback-test="window.jQuery"

crossorigin="anonymous"

integrity="sha384-tsQFqpEReu7ZLhBV2VZlAu7zcOV+rXbYlF2cqB8txI/8aZajjp4Bqd+V6D5IgvKT">

</script>

<script src="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/bootstrap.min.js"

asp-fallback-src="~/lib/bootstrap/dist/js/bootstrap.min.js"

asp-fallback-test="window.jQuery && window.jQuery.fn && window.jQuery.fn.modal"

crossorigin="anonymous"

integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa">

</script>

<script src="~/js/site.min.js" asp-append-version="true"></script>

</environment>

@RenderSection("Scripts", required: false)

</body>

</html>

\_LoginPartial - това е Partial View което визуализира част от информацията на потребителя

@using Microsoft.AspNetCore.Identity

@using Smartphone.Data.Models

@inject SignInManager<User> SignInManager

@inject UserManager<User> UserManager

@if (SignInManager.IsSignedIn(User))

{

<form asp-area="Identity" asp-page="/Account/Logout" asp-route-returnUrl="@Url.Action("Index", "Home", new { area = "" })" method="post" id="logoutForm" class="navbar-right">

<ul class="nav navbar-nav navbar-right">

<li>

<a href="/smartphone/currentuseradvertisement">My Smartphones</a>

</li>

<li>

<a asp-area="Identity" asp-page="/Account/Manage/Index" title="Manage">Hello @UserManager.GetUserName(User)!</a>

</li>

<li>

<button type="submit" class="btn btn-link navbar-btn navbar-link">Logout</button>

</li>

</ul>

</form>

}

else

{

<ul class="nav navbar-nav navbar-right">

<li><a asp-area="Identity" asp-page="/Account/Register">Register</a></li>

<li><a asp-area="Identity" asp-page="/Account/Login">Login</a></li>

</ul>

}

StartUp клас, в който се конфигурира информация за целия проект като база данни, сървиси, route, условие на парола и много други

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Smartphone.Data;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Smartphone.Services.IServices;

using Smartphone.Services.ServicesImplementation;

using Smartphone.Data.Models;

namespace Smartphone.Web

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddTransient<ISmartphoneService, SmartphoneService>();

services.AddTransient<IUserService,UserService>();

services.AddTransient<IHttpContextAccessor, HttpContextAccessor>();

services.AddHttpContextAccessor();

services.Configure<CookiePolicyOptions>(options =>

{

// This lambda determines whether user consent for non-essential cookies is needed for a given request.

options.CheckConsentNeeded = context => true;

options.MinimumSameSitePolicy = SameSiteMode.None;

});

services.Configure<IdentityOptions>(options =>

{

options.Password.RequireDigit = false;

options.Password.RequiredLength = 6;

options.Password.RequiredUniqueChars = 0;

options.Password.RequireLowercase = false;

options.Password.RequireNonAlphanumeric = false;

options.Password.RequireUppercase = false;

});

services.AddDbContext<SmartphoneDbContext>(options =>

options.UseSqlServer(

Configuration.GetConnectionString("DefaultConnection")));

services.AddDefaultIdentity<User>()

.AddEntityFrameworkStores<SmartphoneDbContext>(); services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

app.UseDatabaseErrorPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseCookiePolicy();

app.UseAuthentication();

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller=Smartphone}/{action=Index}/{id?}");

});

}

}

}

Program клас, който е стартовата точка на цялото приложение.

using Microsoft.AspNetCore;

using Microsoft.AspNetCore.Hosting;

namespace Smartphone.Web

{

public class Program

{

public static void Main(string[] args)

{

CreateWebHostBuilder(args).Build().Run();

}

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>

WebHost.CreateDefaultBuilder(args)

.UseStartup<Startup>();

}

}

Изводи

Глава 4 Ръководство за потребителя

* 1. Ръководство за инсталация и настройка
  2. Ръководство за работа със системата

Глава 5 Тестване на системата

Заключение

Използвана литература

ПРИЛОЖЕНИЯ

Програмен код