

**ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО МЕХАНОТЕХНИКА, ЕЛЕКТРОНИКА, ТЕЛЕКОМУНИКАЦИИ И ТРАНСПОРТ**

**„ХРИСТО БОТЕВ“, ГР. ШУМЕН**

**ДИПЛОМЕН ПРОЕКТ**

**Тема: Image Gallery „GlimpseHub”**

**Разработил: Марин Неделчев Неделчев**

**от професия код 481030 „Приложен програмист”**

**специалност код 4810301 „Приложно програмиране”**

**Ръководител-консултант: инж. Николай Христов**

2024 г.

С Ъ Д Ъ Р Ж А Н И Е

I.Въведение

II. Функционалности и правата на достъп които приложението предлага

1. Нива на достъп и правомощия в приложението

III. Използвани технологии при разработката

1. Обща информация за приложението

2. Програмни продукти използвани при разработката

3. Програмни езици използвани при разработката

4. Библиотеки използвaни при разработка

IV. Структура на приложението и организация на кода

1. Структура на приложението

2. Разделеняне на кода, функции и предназначение на всеки раздел

3. Достъп и зависимости на отделните проекти в Solution, видимост

4. Съображения при избора на връзки между класовете

V. База данни

1. Избор на база данни

2. Подход при разработката и внедряване на базата

3. Създаване на база данни и реализация в проекта

4. Модели-описание на релации и организация на данни те във всяка таблица

5. Особености и разлики в моделите и таблиците от базата

6. Типове релаци и между таблиците им оделите

VI. Фронт-енд визуализация и контролери

1. Razor Pages

2. Razor Views

3. Статични ресурси.

VII. Функционалност на приложението която, следва да се имплементира, надграждане.

1. Галерия със снимки:

2. Потребителски профили

3. Коментари и оценки

4. Поддръжка на различни формати за снимки

5. Коментари под даден филм от потребители, които са били на прожекция на филма

6. Връзка потребител – администратор

7.

VIII. Заключение

IX.Използвана литература.

**I.Въведение**

Приложението "GlimpseHub" има за цел да създаде виртуална общност от любители на фотографията и визуалното изкуство, като предоставя платформа за споделяне на изображения. Някои от проблемите, които се решават:

* 1. **Ограничен достъп до публикуване на снимки:** Традиционните методи за споделяне на снимки, като социалните мрежи или електронната поща, могат да имат ограничения, които варират от настройки за поверителност до ограничения в размера на файловете. "GlimpseHub" предоставя платформа, където потребителите могат свободно да качват снимки и да ги споделят с други потребители.
  2. **Откриване на нови таланти и вдъхновение:** Любителите на фотографията могат да се вдъхновяват от работата на други потребители и да откриват нови таланти чрез разглеждане на галериите в "GlimpseHub". Това предоставя възможност за обмен на идеи и техники, както и за подкрепа и възхвала за качествената работа.
  3. **Съхранение и организация на снимки:** "GlimpseHub" позволява на потребителите да създават галерии, които да съхраняват техните снимки. Това не само помага за организацията на снимките, но и позволява на потребителите да създават персонализирани колекции от своите любими изображения.
  4. **Обратна връзка и общностно участие:** Платформата предоставя възможност за обмен на рейтинги и коментари между потребителите. Това не само създава възможност за конструктивна обратна връзка, но и поддържа взаимодействие и връзки между членовете на общността.

**II. Функционалности и правата на достъп които**

**приложението предлага:**

Функционалностите и правата на достъп, които приложението предлага, са както следва:

1.Нива на достъп и правомощия:

**1.1. Гост (GUEST):** Това са всички потребители, които достъпват приложението без да имат регистрация в него. Техната анонимност е гарантирана. Гостът има право да разглежда наличните изображения, както и детайлите за тях. Може да вижда и секцията за контакти.

**1.2. Потребител (USER):** Представлява регистриран потребител, който притежава всички права на гост потребител, но също така има възможност да запази лични данни (имейл и парола) и да ги променя по-късно.

**1.3. Администратор (ADMIN):** Регистриран потребител със специфична роля. Притежава всички права на потребител, но също така има и допълнително право да добавя, редактира и изтрива галерии.

**III. Използвани технологии при разработката**

**1. Обща информация за приложението:**

Приложението е разработено с помощта на Model-View-Controller (MVC) Framework в .NET 6.

MVC е архитектурен шаблон, който разделя приложението на три основни компонента:

* Модел **(Model)**: Представлява данните и бизнес логиката на приложението. Тук се определя какви данни ще се използват и как ще се обработват.
* Изглед **(View)**: Отговаря за визуалната част на приложението, която потребителите виждат и с което могат да взаимодействат. HTML и CSS се използват за създаване на изгледите.
* Контролер **(Controller)**: Обработва заявките от потребителите, взима необходимите данни от модела и ги предоставя на изгледа. Тук се съдържа логиката за обработка на заявките.

1. **Програмни продукти, използвани при разработката:**

* Microsoft Word: Използван за съставяне на документацията към проекта.
* Visual Studio 2022: Интегрирана среда за разработка (IDE), която предоставя мощни инструменти за създаване и управление на .NET приложенията.
* Microsoft SQL Server 2019 и Microsoft SQL Management Studio: За създаване и управление на релационни бази данни.
* Color Picker: Инструмент за избор на цветове, вероятно използван за дизайн на потребителския интерфейс.
* Git и TortoiseGit: За управление на изходния код и колаборация между разработчиците, позволявайки им да работят ефективно в екип.

1. **Програмни езици, използвани при разработката:**

* C# (.NET 6): Използван за сървърната логика на приложението (backend). C# е мощен, обектно-ориентиран език, който е основен елемент на .NET платформата.
* T-SQL: Използван за манипулиране на данни в базата данни чрез Entity Framework.
* HTML и CSS: Използвани за създаване на визуалната част на уеб приложението (frontend).
* JavaScript: Използван за добавяне на допълнителна интерактивност и логика към уеб приложението (frontend).

**4. Библиотеки, използвани при разработката:**

4.1. Библиотеки за сървърната част (Backend):

* Entity Framework: ORM библиотека, която улеснява работата с бази данни чрез C#. Тя предоставя възможност за лесно създаване, четене, актуализиране и изтриване на данни от базата данни.
* Microsoft Identity: Библиотека за управление на потребителските акаунти и ролите в системата. Позволява регистрация, вход, управление на потребителски права и други.
* LINQ: Language Integrated Query (LINQ) е част от .NET Framework, която позволява написването на заявки за данни директно в C# или VB.NET кода, което прави работата с данните по-лесна и ефективна.
* Cloudinary: Cloudinary е облачна услуга за управление на изображения и видео файлове. Тази услуга предоставя мощни възможности за качване, обработка, оптимизация и доставка на медийни файлове в уеб приложенията. С Cloudinary разработчиците могат лесно да интегрират сложни функции за обработка на медийни файлове в техните проекти без необходимостта да се грижат за инфраструктурата за съхранение и обработка на файлове.

4.2. Библиотеки за визуалната част (Frontend):

* Bootstrap: CSS библиотека, която предоставя набор от готови компоненти и стилове за построяване на уеб страници. Това позволява бързо и лесно създаване на съвременни и атрактивни интерфейси.
* Font Awesome: Библиотека от иконки, която предоставя голям набор от векторни икони, които могат да бъдат лесно вградени в уеб страници. Това допълнително подобрява външния вид и функционалността на приложението.

**IV. Структура на приложението и организация на кода.**

1. Структура на приложението:

Приложението е разработено въз основа на ASP.NET MVC и е разширено и преструктурирано с цел да позволи сканиране и управление на кода както и от страна на разработчиците, така и от страна на екипа по тестване и експлоатация. Кода е разделен и организиран в няколко проекта в обща Solution.

2. Разделяне на кода, функции и предназначение на всеки раздел:

Разделянето на кода в различни раздели в приложението има за цел да организира и структурира проекта по начин, който улеснява разработката, поддръжката и разширяването му. Вотът ще разгледаме по-подробно всяка част:

* Application: Console Application, която стартира приложението и управлява останалите компоненти.
* wwwroot: Съдържа статични ресурси като изображения, JavaScript и CSS файлове.
* Areas: Съдържа скафолдирани Identity Razor Pages, адаптирани за целите на приложението.
* Controllers (Контролери): Тук са разположени класове, които управляват заявките и отговорите към приложението. В тези класове се съдържа логиката, която обработва входящите заявки от потребителите и определя какъв отговор да бъде върнат. Контролерите свързват моделите и изгледите (views) на приложението.
* Views (Изгледи): Те съдържат HTML и Razor код, който дефинира визуалния интерфейс на приложението. Изгледите са тези части от приложението, които се визуализират в браузъра на потребителя. Разделянето на изгледите в папки може да структурира логически и визуално компонентите на приложението.
* appsettings.json: Този файл съдържа различни конфигурационни настройки за приложението, като например настройки за връзка с базата данни, настройки за сигурност и други параметри, които могат да бъдат настройвани от администратора на приложението.
* Program.cs: Това е входната точка на приложението. Тук се стартира приложението и се конфигурира хостът, в който ще се изпълнява приложението. Този клас е отговорен за стартирането на приложението и свързването му с хоста.

2.2. Data/Models – папка която съдържа всички entities, които имат представление в базата данни. Разделението на отделните класове-модели в подпапки е организирано целенасочено, но те всички са регистрирани в namespace Data.Models.

Тази папка включва следните раздели:

* Enums – Съдържа енумерации, които се ползват в класовете, на практика в тях се съдържат дефиниция (шифър), как да се интерпретират числовите стойности на отделните колони в базата със стойности от тип енумерация.
* Models – Съдържа базовите абстрактни класове наследени от моделите, както и модели на класове репрезентиращи таблици в базата данни.

2.3. Data – Папка съдържаща информацията нужна на ORM Entity Framework как да направи и свърже Models със таблиците от базата данни:

* Data – Съдържа ApplicationDbContext, наследник на DbContext. Указва кои таблици ще бъдат създадени на базата на съответните класове от Models, също така указва в метода „protected override void OnModelCreating(ModelBuilder builder)“ чрез fluent API правилата които не могат да се дефинират чрез конвенционално именуване или атрибути. Например композитни ключове за mapping таблици.
* ApplicationDbContext.cs – Клас съдържащ метод който да връща конфигуриран ApplicationDbContext (в него трябва да е зададен ConnectionString за да се осъществи връзка с базата данни.
* Migrations – Директория, в която се съдържат ми грациите, които отразяват промените в структурата на базата при началното и създаване както и в процеса на промяна по времена експлоатация на приложението.

3. Съображения при избора на връзки между класовете:

При по строяването на връзките “dependencies”се съблюдават две

правила.

* Да може да функционира приложението тоест във всяка библиотека-проект да има достъп до ресурси(класове)намиращи се в други библиотеки-проекти.
* Да се избегне CircularReference. Не се допуска две библиотеки да са видими взаимно пряко или чрез посредник, вместо това трябва достъпът да бъде иерархичен. По тази причина видно от зависимостите може да се заключи, че на върха на йерархията на проектит е в Sоlution се намира Application конзолното приложение от което се стартира цялото приложение.

**V. База данни.**

1. Избор на база данни:

При разработката на приложението е избрана релационна база данни Microsoft SQL Server и приложение за управлението и Microsoft SQL Management Studio, в което може да се визуализира базата както и да се пишат Transact SQL заявки към базата.

2. Подход при разработката и внедряване на базата:

2.1. Подходи при създаване на нова база данни:

* Data Base First – При този подход базата се изгражда посредством скрипт написан и изпълнен на Transact SQL, като всички правила, Ограничения, процедурии т.н. се дефинират в SQL. След като базата е готова на всяка таблица от нея трябва дa се създаде отразяващя клас, който да репрезентира данните от базата. Създаването на тези класове е възможно да се изпълни и чрез Scaffold автоматично.

Основно предимство на този подход е, че базата може да се дефинира по -оптимално и ясно в T-SQL. Могат да се използват всички “features” на конкретният SQL език вместо само такива валидни за всички SQL езици.

Недостатък на този подход е че базата дефиниран а през T-SQL е обвързана с конкретният избор на SQL сървър, изисква се познание в избраният SQL език.

* Code First – При този подход базата се изгражда след и възоснова на класове в C#. За да се създаде базата, се генерират Миграции, които се прилагат към базата. Всяка такава миграция съдържа автоматично генерирани T-SQL правила. Като могат да се извършват промени по класовете и да се допълват миграциите в последствие.

Основно предимство на този подход е, че ORM кореспондира с избраният SQL сървър, което позволява на програмиста да не използва чист SQL и предоставя възможност с много малко усилия да се смени SQL сървъра например с Postgre или MySQL в последствие.

Недостатъците на този подход са:

Понеже се разчита на Provider клас който да генерира SQL код, се изисква познание как да се укажат правила при самите класове които дадоведат до правилна интерпретация на класовете. Пример затова са атрибути, FluentApi правила, или именоване на свойствата на класовете по конвенция. Друг недостатък е, че ако се изискват функционалности от базата каквито само конкретната база предлага, то това не е най-удачният вариант защото Provider класовете за всеки тип база поддържат общи интерфейси, тоест самообща функционалност.

2.2. Избран подход и обосновка:

За разработката на приложението е предпочетен подхода Code First понеже в процеса на разработка ще се променят класовете и ще се разчита на миграции за да се отразят тези промени, също така Code First предлага възможност да се разработи приложението без използване на T-SQL а фокусът се измества в посока на C# класовете, където може да се възползва програмистът от Intelisense функциите на Visual Studio 2022. За неконвенционалните правила се разширява методът “protected override void OnModelCreating(ModelBuilder builder)”Тук се задават правила, като композитни ключове например. В допълнение към конвенционалните правила и FluentApi правилата, наместа са използвани атрибутни обозначения в класовете дефиниращи таблиците модели \* (Models).

3. Създаване на база даннии реализация в проекта.

Класовете репрезентиращи базата данни се намират в папката Data/Models и могат да се виж приложението към дипломният проект**.**

3.1. Идентификаторите:

Всеки клас, който репрезентира конкретната таблица в базата данни, би следвало да има идентификатор (ID). Този идентификатор може да бъде от числов или string тип (GUID). Когато се цели по-голяма сигурност на данните и необходимост да се предвиди идентификатор на отделни записи при зловреден достъп отвън, се препоръчва string тип за идентификатор, а когато това не е нужно се използва int. В конкретното приложение се използва GUID за идентифициране на потребителите или AppUser класа, докато за по-голяма простота се използва int32 тип. Това е така, защото За по-нататъчно улеснение ще наричаме класовете съдържащи Entity типове (или репрезентиращи таблици в база данни) – Модели.

Потребителят има право да достъпва своите данни или да вижда данни на конкретен потребител, когато това е позволено (например в коментари се вижда ако имат анализирана секция), съобразно GDPR. Съображенията включват необходимостта да не се допуска достъпът до данни на произволни потребители чрез for-цикъл например. Някои от таблиците имат така наречени композитни ключове, вместо идентификатор от тип int или string. Това се налага в случаите, когато има множество.на редове с еднакъв M apping ключове, съдържащи „връзка“ между две таблица или 2 Foreign Key-а със съответни типове, съответстващи на таблица. В този случай ако се цели уникалност на връзката, за да се елиминира повторението на конкретната връзка се използва композитен ключ, генериран на базата на двата Foreign ключа от връзката, по този начин при последващ опит да бъде създаден нов запис със същите ключове това не се позволява от базата данни, тъй като вече има наличен запис с такъв ID.

3.2. Обща функционалност във (за) всички модели:

Всеки модел съдържа базова функционалност, която е обща за всички модели-деца. За тази цел са създадени два класа-родители, които представят общата функционалност за всички модели-деца.

Класът BaseEntityData е родителски клас на BaseEntity<T>, като всички класове "са BaseEntityData", но не всички "са BaseEntity".

**\*Абстрактните класове не позволяват инстанциране, тяхната цел е да предоставят функционалност на наследниците им и да позволят на следващите им класове да бъдат представени като абстрактния тип при дефинирането на данните (например работа с колекции и др.).**

BaseEntityData:

* Съдържа функционалност, която не е необходима за всеки модел. При инстанциране на който и да е наследник си запълва собственото си свойство DateOfCreation от тип DateTime.
* Съдържа свойство от тип bool с името IsDeleted, което отразява дали даденият запис е изтрит от базата данни или не. Това позволява "Soft-Deletion" на записите в базата данни. При промяна на това свойство то е модифицирано така, че да запомни датата на изтриване на обекта, като при следващо изтриване или възстановяване на този запис, датата на изтриване не се променя. Поради тази причина типът на данните на

DateOfDeletion е nullable\* DateTime, тъй като неизтритите записи нямат дата на изтриване, докато DateTime не поддържа null.

* DateOfDeletion и DateOfCreation се задават автоматично на текущата дата и час според GMT\*.
* По този начин се поддържат функционалности за управление на вече изтритите записи.

**\*Soft-Deletion – При работа с данни, които имат външни ключове, не е лесно даден запис да бъде изтрит, ако неговият ID се явява външен ключ за друга таблица. Например, ако имаме "Потребител" и "Коментар на потребителя" и искаме да изтрием потребителя, възниква въпросът кой е оставил коментара? Подходите за справяне с този проблем са няколко. Каскадно, Задаване на Foreign Keys като null, ако това е възможно или Ignore. Всеки един от подходите горе създава множество проблеми, затова решението е софтуерно да се зададе колона в таблиците от тип bool IsDeleted (представена в базата данни чрез 0 - false и 1 - true), където вместо да се изтриват данните просто не се визуализират и борави със записи, имащи зададена стойност true.**

**\*nullable – По подразбиране типовете имат стойност, която не винаги е null. Например, типа int32, ако не бъде указана стойност по подразбиране е със стойност 0. Както и типа DateTime задава начална дата при инициализиране. C# позволява декларирането на типове, които да имат възможност да заемат стойност null умишлено, когато това не е тяхното поведение по подразбиране.**

**\*GMT – Приложението следва да се качва (deploy) в интернет и не е гарантирано, че ще се ползва само от потребители в една и съща часова зона, затова е редно да се използва универсална общо приета часова зона, която се конвертира към локалното време, което се репрезентира различно за всеки отделен потребител.**

* BaseEntity<T>: Представлява Generic абстрактен клас, в който се указва типът на идентификатора и бива наследен пряко от класове, които искаме да имат конкретен тип идентификатор. Това е важно за всички модели без mapping, където се търси уникалността на връзката. Типът е Generic, защото се търси гъвкаво указване на типа на идентификатора (string, int), при същото наследяване. Класът BaseEntityData<T> съдържа свойството Id от тип T, като над свойството с ключа съдържа атрибут, указващ на EntityFramework, че идентификаторите трябва да бъдат задавани от базата, а не ръчно (Identity\*).

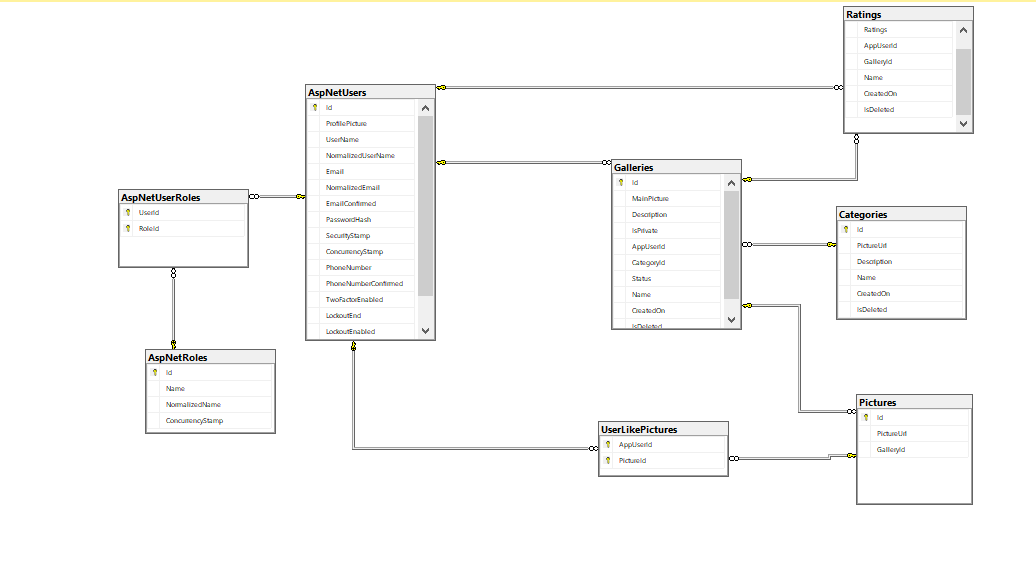
**\*Identity - при ключовете означава записите да бъдат създадени без да се указва ръчно стойността на техните ключове (Id) в базата данни. Съответно, базата извършва проверка до кой пореден номер достигнат е при настаняването на последният ключ и при своя идентификатор присвоява пореден номер. Интересното е, че при изтриване на записи и ключовете не се обновяват броите, така че е възможно да има липсващи Id-та (да не са поредни). Това е нормално, понеже основната задача на идентификаторите е да бъдат уникални, а не поредни.**

**4. Модели - Описание на релациите и организацията на данните във всяка таблица (модел).**

В приложението е използвана библиотеката Microsoft.AspNetCore.Identity. Тя предоставя базова имплементация на класа потребител, както и на таблиците, управляващи неговите права и достъп до приложението. Операциите като регистрация, влизане в системата и управление на ролите стават по-лесни с използването на тази библиотека, която предоставя възможност за преизползване на образцови решения и разширяване на функционалността. Библиотеката включва дефиниции на класовете IdentityUser и IdentityRole, както и съпътстващите ги класове за работа с базата данни - Services, включващи UserManager, SignInManager и RoleManager.

**4.1. AppUser** - Представлява клас, наследяващ IdentityUser, като му добавя нови свойства и методи, имащи отношение към конкретното приложение. От библиотеката се използват и връзващите таблиците AspNetRoles и AspNetUserRoles, които съответно асоциират конкретния потребител със съответната му роля. Библиотеката предлага и други класове, но те нямат отношение към функционалността на приложението.

Фиг. **Entity Relationship Diagram** - Диаграма на връзките в базата данни.



**\*Тривиална задача в програмирането означава задача, чието решение е ясно и лесно да се имплементира в рамките на кратко време и може да се спести посредством готов код.**

**4.2. Pictures** – Съдържа снимки като взема адреса на изображения, може да се разгледа и. Представлява таблица, в която няма външни ключове, което означава, че при популиране на базата данни, може да се започне с тази таблица без да възникнат конфликти. Ниво -1.

**4.3. Categories** – Съдържа информация за даден актьор. По начин на имплементация се регистрират актьори и продуценти в една и съща таблица. Представлява таблица, в която няма външни ключове, което означава, че при популиране на базата данни, може да се започне с тази таблица без да възникнат конфликти. Ниво -1.

**4.4. Galleries** – Съдържа информация за конкретенa галерия. В тази таблица има препратка към снимките,категорията които са на галерията, Тази таблица не може да се популира преди да се популира таблицата с актьори за това тя е от Ниво -2.

**4.5. Ratings** – Съдържа информация за прожекция на конкретен филм, тук има информация за мястото на което се състоя прожекцията и колко струват билетите. В тази таблица има препратка към идентификатор на филм. Тази таблица не може да се популира преди да се популира таблицата с филми за това тя е от Ниво -3.

**4.6. UserLikePictures** – Mapping таблица свързваща User с Pictures. Тук връзките трябва да са уникални и затова идентификаторът е композитен ключ на базата (FK\_AppUserId & FK\_PictureId).

Таблицата показва кога някой от потребителите е харесал някоя от снимките и с помоща на FluentApi правилата които съм направил в ApplicationDbContext.cs забранява на този потребител повторно да хареса тази снимка. Тази таблица не може да се популира преди да се популира таблицата с потребители и снимки за това тя е от Ниво -3.

**\*Ниво – Ред на добавяне на данни в базата данни, не се допуска да се добави запис, който има външен ключ препратка към несъществуващ ID в другата таблица поради всичко още недобавени записи. Ако две таблица са от едно ниво, техният ред може да се размени при добавянето на данни.**

5. Особености и разлики в модела и таблиците от базата данни:

5.1. Имена на таблиците и класовете:

В базата данни таблиците имат имена в множествено число, защото съдържат записи и всеки ред от таблицата е отделен запис със собствен идентификатор, т.е., в базата данни се определят правила за подреждане на множество данни като всяка таблица е контейнер за множество записи.

В модела отделните класове имат имена в единствено число, защото всеки модел описва 1 конкретен тип обект, т.е., индивидуални особености за всеки запис.

5.2. Разлики при моделите и таблиците в базата данни:

Някои данни се представят по различен начин в базата данни и в кода, поради естеството на работата на базата данни.

**\*Тегло: При теорията на графите, представлява допълнителна информация към дадено ребро или дъга от разясняваща данни, като дължина на траекторията, разход на транспорт, време за пътуване и т.н. В конкретния случай се разбира информация, свързана със съответствието на два записа от различни таблица с допълнителна информация.**

**\*Енумерации: В езика C# енумерацията представлява двойка от [числова стойност + стрингова стойност], докато в базата данни се репрезентира само като числова стойност. Следователно смисълът от всяка числова стойност на енумерацията е дефиниран в приложението в C# и само то може да разчете, какво означава например Status = 1 в случая (Open). Енумерацията Status съдържа възможностите [Pending, Open, Closed].**

* Представяне на сложни данни като string:

Представянето на сложни данни като текст е важно, тъй като базата данни може да съхранява текстови данни за множество обекти, които се отнасят само до един обект (запис). Това може да предотврати създаването на допълнителни таблиците със свързващи ги допълнителни mapping таблици. Примерът за този случай е изображен в таблицата "Galleries" и модела "Gallery" (галерия). В този конкретен случай имаме колекция от множество препратки към изображения, но те се отнасят само до една галерия. Следователно не е необходимо да се създава таблица с записи за всяко изображение, тъй като информацията от тази таблица не се използва от множество галерии, а само от една. Вместо това множеството от изображения се съхранява като един string, който се разчита от C# и се превръща в колекция от string, като всеки от тях представлява отделна връзка към изображение. При нужда от по-сложно представяне на данните, когато данните са индивидуални, те могат да бъдат прочетени като JSON сериализация и десериализация.

**\*JSON (JavaScript Object Notation) е формат, който позволява представянето на данни като колекции и/или типове във вид на низове, които лесно се пренасят в WEB. Той е аналогичен на стандарта XML. Тези формати могат да пренасят информация за състоянието, но не и за поведението на класовете.**

**\*Сериализацията е процесът на преход от обект към низово представяне в JSON или XML формат.**

**\*Десериализацията е процесът на преход от низово представяне в JSON или XML формат към конкретен сложен тип.**

6.Типове релации между таблиците и моделите:

6.1. One-To-One (Един към един):

При тази релация в модел A има външен ключ към модел B и обратно, в модел B има външен ключ към модел A. Освен това, тези външни ключове в двете таблиците са със зададено ограничение Unique. Това се използва в приложението при Galleries -> Categories, защото една галерия може да има само една категория поне в моето приложение понеже не е напълно завършено.

6.2. One-To-Many (Един към много):

При тази релация в модел A има външен ключ към модел B. При модел B това се представя като колекция от всички обекти A, които са свързани към конкретния обект. В приложението пример за тази релация е отношението Gallery и Pictures. Една галерия може да има много снимки.

6.3. Many-To-Many (Много към много):

При тази релация моделите A и B имат колекция от общи записи във връзващата трета таблица C. В двете модели A и B са налични колекции с връзки, които са идентични. В моето приложение няма такава връзка към момента, но мислех, че ще е полезно да се спомене и тази релация.

**VI. Фронт-енд визуализация и контролери.**

1. Razor Pages:

Използват се Razor Pages за управление на потребителските данни като log-in, log-out, register, достъп до данните на потребителя и други.

1. Razor Views: Използват се Razor Views за визуализиране на страниците от приложението със индивидуална бизнес логика. Всички Views се подпъхват във общия \_Layout.cshtml, който също представлява View, като е имплементирана логика по зареждане на индивидуален CSS и JavaScript за всяко View. \_Layout.cshtml съдържа всички връзки към CSS и JS, като той трябва да е достъпен за всеки View. Примерите за такива връзки включват глобален CSS, Bootstrap и други. Освен това, \_Layout.cshtml съдържа неизменяема обща част от приложението, като например навигационната лента и Footer. В някои от Views е вградена и редакция на нови филми, като се използва библиотеката VueJS през CDN, за да се постигне динамично добавяне на нови полета от формите. Това е така, защото за даден филм броят на кадрите и актьорите не е предварително дефиниран и може да варира. Достъпът до Views е ограничен в някои случаи само за потребители с роля администратор, което е със сървърна логика на ниво Action или дори на ниво Controller, освен при FrontEnd, което е за удобство и сигурност.
2. Статични ресурси:

Статичните ресурси като изображенията, CSS и JavaScript са разположени в папката wwwroot, като с цел разграничаване на кода ресурсите в тази папка са организирани в под-директории.

**VII. Функционалност на приложението, която следва да се имплементира и надгражда:**

1. Галерия със снимки:

* Галерията със снимки е основната част на сайта, където потребителите могат да разглеждат и се вдъхновяват от различни изображения.
* Тя може да бъде организирана в различни категории и албуми, които да помагат на потребителите да намират снимките, които търсят.

2. Потребителски профили:

* Потребителските профили позволяват на потребителите да създадат персонализирани акаунти, които да съхраняват техните качени снимки и настройки.
* С профилите потребителите могат да управляват своите снимки, да създават албуми и да споделят техните творби с останалите потребители.

3. Коментари и оценки:

* Възможността за коментари и оценки позволява на потребителите да споделят своите мнения и впечатления за конкретни снимки.
* Това създава интерактивност и общностност на сайта, като позволява на потребителите да се ангажират със съдържанието и да се възползват от мнението на други потребители.

4. Поддръжка на различни формати за снимки:

* Сайтът трябва да поддържа различни формати за снимки, за да може потребителите да качват изображения във формати като JPEG, PNG, GIF и други.

5. Търсене и филтриране:

* Функцията за търсене и филтриране позволява на потребителите да откриват снимки по различни критерии, като например категории, етикети, автори и други.
* Това прави намирането на конкретни изображения по-лесно и удобно за потребителите.

6. Административен панел:

* Административният панел предлага на администраторите възможността да управляват всички аспекти на сайта, включително качването на снимки, потребителските профили, коментарите и други функции.
* Той осигурява контрол и модерация на съдържанието на сайта и прави администрирането му по-лесно и ефективно.

7. Responsive дизайн:

* Responsive дизайн означава, че уебсайтът се адаптира към различни размери на екраните на устройствата, като компютри, таблети и мобилни телефони.
* Това гарантира, че потребителите имат оптимално преживяване, независимо от устройството, което използват.

**VIII. Заключение**

Приложението ми не е завършено, но в документацията се опитах да обесня почти всичко, което може да се получи в края бих казал, че съм готов поне на 60%, не бих казал, че е най-доброто, което можех да направя, защото мога и повече но съм доволен от това което направих за това време в което работих.

Нещо допълнително което не съм споменал дотук е администраторският профил:

Email-Address: [marin@admin.com](mailto:marin@admin.com) Password: AdminAccount123,

Ако по някъква причина приложението не тръгва вижте фаилът appsettings.json и поправете ConnectionString-а с работещ такъв най-често трябва да смените само Server =…; с вашият сървър.

**IX. Използвана литература.**

1. Боровска Пл., Компютърни системи, изд. Сиела, 2005.

2. Боянов Л., К. Боянов и др., Компютърни мрежи и телекомуникации, изд. “Авангард Прима”, София, 2014.

3. Васил Георгиев. Съвременните технологии за конкурентна обработка. Издателство „Св. Кл. Охридски”, 2013.

4. Владимир Димитров, Васил Георгиев. Съвременните модели за информационно-технологично обслужване. Издателство „Св. Кл. Охридски”, 2012.

5. Илиева С., В. Лилов, И. Манова, Изграждане на софтуерни приложения, 2006, издателство СУ “Кл. Охридски”.

6.Илиева С., В. Лилов, И. Манова, Методи и подходи за разработване на софтуерни системи, 2010, издателство СУ “Кл. Охридски”.