**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 (модел)

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. Галерии

2. Снимки

3. Събития

4. Секция отзиви от клиенти

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

6. Връзка потребител - администратор

**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. Ето някои от проблемите, които то решава:

* 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.