Міністерство освіти і науки України

Львівський національний університет імені Івана Франка

Факультет електроніки та комп’ютерних технологій

Кафедра радіоелектронних і комп’ютерних систем

**КУРСОВА РОБОТА**

з курсу «Програмування та підтримка веб застосувань»

на тему: «Розробка веб-застосунку для бази даних рейтингу продукції за допомогою кросплатформного фреймворку .ASP NET CORE, MSSMS »

Виконав:

студент групи ФеІ – 34

спеціальності 122 – комп’ютерні науки

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кицун О.В.

Науковий керівник:

\_\_\_\_\_\_\_\_\_\_\_\_\_доц. Бойко Я.В.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_2023 р.

**Львів 2023**

**Зміст**

**Анотація**……………………………………………………………………..3

**Вступ**…………………………………………………………………………4

**Розділ 1.** Особливості, технології, СУБД………………………………….5

* 1. Мова програмуваня C#...................................................................5
  2. Середовище CLR………………………………………………….8
  3. Середовище розробки Visual Studio……………………………..9

1.4 Платформа ASP.NET Core………………………………………10

1.5 Entity Framework Core……………………………………………12

1.6 Системи управління базами даних……………………………...14

1.7 Microsoft SQL Server……………………………………………..16

**Розділ 2.** Ідея та можливості застосування………………………………..19

**Розділ 3.** Розробка веб-додатку …………………………………………...20

3.1 Розробка ………………………………………………………….20

3.2 Можливі доопрацювання………………………………………..43

**Висновок**……………………………………………………………………44

Список використаної літератури…………………………………………..45

Додатки……………………………………………………………………...46

**Анотація**

За допомогою мови програмування C# та її кросплатформних фреймворків ASP.NET Core та Еntity Framework core а також бази даних Microsoft SQL було створено веб-застосунок для оцінювання якості продукції певної категорії

**Abstract**

Using the C # programming language and its cross-platform ASP.NET Core and Entity Framework core frameworks, as well as the Microsoft SQL database, a web application was created to assess the quality of a certain category of products.

**Вступ**

Всі ми знаємо, що розробка веб-застосунку складається з двох частин, так званого frontend-у та backend-у, і якщо для реалізації першого обирати технології та мови програмування не доводиться, то для реалізації серверної частини існує безліч засобів та способів реалізації . Ці дві частини веб-застосунку незалежно від обраних вами технологій, легко поєднуються між собою. Доцільність поєднання тих чи інших технологій залежить від потреб та поставленого завдання. Моя робота побудована на основі поєднання таких технології. Моя робота полягає створенні веб-застосунку для оцінювання якості продукції, за допомогою мови програмування C# та її кросплатформних фреймворків ASP.NET Core та Еntity Framework core, з’єднання його з базою даних

Мета роботи: Дослідити методику розробки веб сайту за допомогою архітектури MVC на фреймворку ASP.NET Core . Зрозуміти логіку та принципи роботи цієї архітектури, та визначити її основні переваги та недоліки. Також освоїти основні функції та властивості Еntity Framework Сore для роботи з базами даних, при розробці веб-застосунків. А також дослідити методику роботи «Сode first» при роботі з базами даних. На основі отриманих знань створити веб-застосунок на якому б користувачі могли би як оцінювати так бачити рейтинг продукції.

**Розділ 1.** **Особливості, технології, СУБД**

* 1. **Мова програмування C#**

С# є основною мовою розробки програм на платформі .NET корпорації Microsoft. У ньому вдало поєднуються випробувані засоби програмування з останніми нововведеннями і надається можливість для ефективного і практичного написання програм, призначених для обчислювального середовища сучасних підприємств. Це, без сумніву, одна з найважливіших мов програмування XXI століття.

Історія основної гілки мов програмування, яка призвела до появи C #, сходить до 60-х років, а саме, до часу виникнення мови B. Останній є типовим представником ранніх імперативних мов програмування. Мова B була придумана у 1963 році творчим колективом розробників, основним творцем мови прийнято вважати К. Томпсона з Технологічного інституту Массачусетсу (Ken Thompson, MIT). Основною метою розробки мови була реалізація операційної системи UNIX. Вже існувала мова PL / I, що застосовувалася у той час для мейнфреймів виробництва компанії IBM, була досить громіздкою і менше підходила для поставленої задачі, ніж нове, оригінальне рішення вчених-практиків.

Наступним кроком в "алфавіті" мов програмування, провідному до мови C #, стала мова C, яка була винайдена на основі мови B в 1972 році. Авторами нової мови програмування стали К. Томпсон і Д. Рітчі (Dennis Ritchie), які працювали в дослідницькій лабораторії компанії AT&T. У варіанті C мови B розширився за рахунок явного використання типів, структур і ряду нових операцій. Подальший розвиток мови відбувалося в тій же організації. І знову приблизно через 10 років, в 1984 році, Б. Страуструп виступив з проектом мови С + + - ООП-розширення мови C, в якому вводиться поняття класу як об'єкта даних.

Нарешті, вже в 2000 році, тобто більш ніж через 15 років, корпорація Microsoft випустила у світ C + + нового покоління під назвою C # ("Сі шарп"), основним постулатом якого є вислів: "будь-яка сутність є об'єктом". Мова заснована на строгій компонентній архітектурі і реалізує передові механізми забезпечення безпеки коду.

Як вже зрозуміло у мови програмування C # "багату спадщину". Він є прямим спадкоємцем двох найбільш вдалих мов програмування: С і C ++. Він також має тісні родинні зв'язки з ще однією мовою - Java.

Також слід сказати, що мова C # має декілька версій на даний момент їх існує 9. Про це і буде йти мова далі. З моменту випуску вихідної версії 1.0 розвиток C #дуже швидко розвивався. Незабаром після версії 1.0 корпорація Microsoft випустила версію 1.1, в яку було внесено чимало коректив, але мало значних можливостей. Однак ситуація абсолютно змінилася після випуску версії C # 2.0. Поява версії 2.0 стало поворотним моментом в історії розвитку С #, оскільки в неї було введено багато нових засобів, в тому числі узагальнення, часткові типи і анонімні методи, які ґрунтовно розширили межі можливостей і область застосування цієї мови, а також підвищили його ефективність. Поява цієї версії продемонструвало високі стандарти корпорації Microsoft до підтримки цієї мови в довгостроковій перспективі. Наступною значною віхою в історії розвитку C # став випуск версії 3.0. У зв'язку з впровадженням багатьох нових властивостей у версії C # 2.0 можна було очікувати деякого уповільнення в розвитку С #, оскільки програмістам потрібен час для їх освоєння, але цього не сталося. З появою версії 3.0 корпорація Microsoft впровадила ряд нововведень, абсолютно змінили загальне уявлення про програмування. До числа цих нововведень ставляться, серед іншого, лямбда-вирази, мова інтегрованих запитів (LINQ), методи розширення та неявно типізовані змінні. Звичайно, всі ці нові можливості дуже важливі, оскільки вони надали

помітний вплив на розвиток даної мови, але серед них особливо виділяються дві: мова інтегрованих запитів (LINQ) і лямбда-вирази. Вони вносять абсолютно новий акцент в програмування на C # і ще глибше підкреслюють його провідну роль в безперервної еволюції мов програмування. У версії 5 мова була розширена набором класів для роботи з асинхронними операціями. З'явилися нові ключові слова async, await, а також TAP шаблон (Task-based Asynchronous Pattern). Версія 6 несла в собі зміни, які часто називають «синтаксичним цукром» (syntactic sugar), який слугує для спрощення написання конструкцій. Перелік всіх спрощень можна знайти на порталі <https://metanit.com/> . До нововведень у версії 7 увійшли: бінарні літерали - для запису чисел в двійковій системі числення (0b0001); локальні функції; кортежі (tuples), дозволяють методу повертати кілька значень одночасно; зіставлення з шаблоном (pattern matching) та можливість повертати з методи посилання на об'єкт. У 8 версії було додано: методи інтерфейсів за замовчуванням (Default interface methods), Null-оголошення присвоєння(Null-coalescing assignment), яке також є синтаксичним цукром, також було додано Nullable reference types (Розробники прийшли до висновку, що буде краще, якщо все посилальні типи не будуть підтримувати null за замовчуванням. При цьому їх null можна буде зробити сумісним, якщо декорувати його знаком.), Asynchronous streams( У нову версію була додана нова конструкція foreach await. Тепер використання асинхронного потоку буде вимагати додавання ключового слова await.) та Pattern matching enhancements (У восьму версію було додано ряд відповідностей шаблоном, конструкція is була розширена для перевірки вираження на відповідність шаблону. При цьому шаблони можна вкладати один в одного. Підтримується чотири види шаблонів: type, constant, discard, var.) До версії 9.0 було додано: властивості тільки для ініціалізації, Методи доступу init і поля тільки для читання, записи (Запис - це все ще клас, але ключове слово record наділяє клас додатковим поведінкою, яке подібно до поведінки значення. Взагалі кажучи, записи визначаються своїм змістом, а не ідентичністю. В цьому відношенні записи набагато ближче до структур, але все ж є посилальними типами. Хоча записи можуть бути змінними, вони в першу чергу створені для кращої підтримки незмінних моделей даних.), With-вирази, шаблони відношень, вирази new і цільової тип, логічні шаблони та багато чого іншого

**1.2 Середовище CLR**

Середовище CLR управляє виконанням коду .NET. Діє вона за наступним принципом. Результатом компіляції програми на C # є не виконуваний код, а файл, який містить особливого роду псевдокод, званий Microsoft Intermediate Language, MSIL (проміжна мова Microsoft). Псевдокод MSIL визначає набір переносяться інструкцій, що не залежать від конкретного процесора. По суті, MSIL визначає стерпний мову асемблера. Слід, однак, мати на увазі, що, незважаючи на позірну схожість псевдокоду MSIL з байт-кодом Java, це все ж таки різні поняття.

Призначення CLR - перетворити проміжний код у виконуваний код по ходу виконання програми. Отже, будь-яка програма, скомпільована в псевдокод MSIL, може бути виконана в будь-якому середовищі, де є реалізація CLR. Саме таким чином частково досягається переносимість в середовищі .NET.

Псевдокод MSIL перетвориться в виконуваний код за допомогою JIT-компілятора. Скорочення JIT означає точно в строк і відображає оперативний характер даного компілятора. Процес перетворення коду відбувається наступним чином. При виконанні програми серед CLR активізує JIT-компілятор, який перетворює псевдокод MSIL в власний код системи на вимогу для кожної частини програми. Таким чином, програма на C # фактично виконується як власний код, незважаючи на те, що спочатку вона скомпільована в псевдокод MSIL. Це означає, що така програма виконується так само швидко, як і в тому випадку, коли вона початково скомпільована в власний код, але в той же час вона набуває все переваги переносимості псевдокоду MSIL.

Крім псевдокоду MSIL, при компіляції програми на С # виходять також метадані, які служать для опису даних, які використовуються в програмі, а також забезпечують просте взаємодія одного коду з іншим. Метадані містяться в тому ж файлі, що і псевдокод MSIL.

Також у межах цього блоку я вважаю потрібним розповісти про загальну систему типів (Common Type System) або скорочено система CTS та загальномовне специфікація (Common Language Specification). Специфікація CTS повністю описує всі можливі типи даних і всі програмні конструкції, підтримувані виконуючою середовищем, вказує, яким чином ці сутності можуть взаємодіяти один з одним, і як вони представлені в форматі метаданих .NET Важливо розуміти, що окремо взятий мову, сумісний з .NET, може не підтримувати абсолютно все функціональні засоби, які визначаються специфікацією CTS.

Існує родинна загальномовне специфікація (Common Language Specification) або скорочено специфікація CLS, де описано підмножина загальних типів і програмних конструкцій, яке повинні підтримувати всі мови програмування для .NET. Таким чином, якщо ви будуєте використовувати типи .NET, що відкривають доступ тільки до сумісних з CLS засобів, то можете бути впевнені в тому, що їх зможуть споживати всі мови, які підтримують .NET. І навпаки, якщо ви застосовуєте тип даних або програмну конструкцію, яка виходить за межі CLS, тоді ви не можете гарантувати, що кожна мова програмування для .NET виявиться здатним взаємодіяти з вашою бібліотекою коду .NET. На щастя, як ви побачите далі в розділі, компілятору C# досить просто повідомити про необхідність перевірки всього коду на предмет сумісності з CLS.

**1.3 Середовище розробки Visual Studio**

У більшості людей, коли вони чують «C#», одразу уявляють перед очима вікно програми Visual Studio. Для при всій моїй повазі до продукту JetBrains Project Rider-у, але на сьогоднішній день середовище розробки Visual Studio є беззаперечним лідером серед C # розробників, адже це інтегроване середовище розробки має дуже широкі можливості, інтуїтивно зрозумілий інтерфейс. А також просто дуже зручне у використанні. Слід сказати, що середовище має підтримку інші мови, та ряд різних технологій, як для розробки програм для Windows, Android і iOS, а також сучасних веб-додатків і хмарних служб. Також важливим є те, що розробниками Visual Studio, як і самої мови C # є компанія Microsoft, що означає постійну підтримку та розширення функціоналу. До того ж Visual Studio має безкоштовну версію, якою може скористатися кожен бажаючий. Повноцінна Visual Studio доступна на Windows та MacOS, для користувачів Linux існує тільки кросплатформнний редактор коду зі значно меншим функціоналом Visual Studio Code.

**1.4 Платформа ASP.NET Core**

Платформа ASP.NET Core представляє технологію від компанії Microsoft, призначену для створення різного роду веб-додатків: від невеликих веб-сайтів до великих веб-порталів і веб-сервісів. З одного боку, ASP.NET Core є продовженням розвитку платформи ASP.NET. Але з іншого боку, це не просто черговий реліз. Вихід ASP.NET Core фактично означає революцію всієї платформи, її якісна зміна.

Розробка над платформою почалася ще в 2014 році. Тоді платформа умовно називалася ASP.NET vNext. У червні 2016 року вийшов перший реліз платформи. А в листопаді 2020 року побачила версія ASP.NET Core 5.0, яка власне і буде охоплена в поточному керівництві.

ASP.NET Core тепер повністю є opensource-фреймворком. Всі вихідні файли фреймворку доступні на GitHub. ASP.NET Core може працювати поверх крос-платформної середовища .NET Core, яка може бути розгорнута на основних популярних операційних системах: Windows, Mac OS, Linux. І таким чином, за допомогою ASP.NET Core ми можемо створювати крос-платформні додатки. І хоча Windows як середовище для розробки і розгортання програми досі превалює, але тепер вже ми не обмежені тільки цією операційною системою. Тобто ми можемо запускати веб-додатки не тільки на ОС Windows, але і на Linux і Mac OS. А для розгортання веб-додатки можна використовувати традиційний IIS, або крос-платформний веб-сервер Kestrel.

Завдяки модульності фреймворка всі необхідні компоненти веб-додатки можуть завантажуватися як окремі модулі через пакетний менеджер Nuget. Крім того, на відміну від попередніх версій платформи немає необхідності використовувати бібліотеку System.Web.dll.

ASP.NET Core включає в себе фреймворк MVC, який об'єднує функціональність MVC, Web API і Web Pages. У попередніх версії платформи дані технології реалізувалися окремо і тому містили багато дублюючої функціональності. Зараз же вони об'єднані в одну програмну модель ASP.NET Core MVC. А Web Forms повністю пішли в минуле. Крім об'єднання вищезазначених технологій в одну модель в MVC був доданий ряд додаткових функцій. Однією з таких функцій є тег-хелпери (tag helper), які дозволяють більш органічно поєднувати синтаксис html з кодом С #.

ASP.NET Core характеризується розширюваністю. Фреймворк побудований з набору щодо незалежних компонентів. І ми можемо або використовувати вбудовану реалізацію цих компонентів, або розширити їх за допомогою механізму спадкування, або зовсім створити і застосовувати свої компоненти зі своїм функціоналом.

Також було спрощено управління залежностями і конфігурація проекту. Фреймворк тепер має свій легкий контейнер для впровадження залежностей, і більше немає необхідності застосовувати сторонні контейнери, такі як Autofac, Ninject. Хоча при бажанні їх також можна продовжувати використовувати.

В якості інструментарію розробки ми можемо використовувати останні випуски Visual Studio, починаючи з версії Visual Studio 2015. Крім того, ми можемо створювати додатки в середовищі Visual Studio Code, яка є крос-платформної і може працювати як на Windows, так і на Mac OS X і Linux.

Для обробки запитів тепер використовується новий конвеєр HTTP, який заснований на компонентах Katana і специфікації OWIN. А його модульність дозволяє легко додати свої власні компоненти.

Якщо підсумувати, то можна виділити наступні ключові відмінності ASP.NET Core від попередніх версій ASP.NET:

* Новий легкий і модульний конвеєр HTTP-запитів
* Можливість розгортати додаток як на IIS, так і в рамках свого власного процесу
* Використання платформи .NET Core і її функціональності
* Поширення пакетів платформи через NuGet
* Інтегрована підтримка для створення та використання пакетів NuGet
* Єдиний стек веб-розробки, що поєднує Web UI і Web API
* Конфігурація для спрощеного використання в хмарі
* Вбудована підтримка для впровадження залежностей
* Можливість розширення
* Кросплатформеність: можливість розробки і розгортання додатків ASP.NET на Windows, Mac і Linux
* Розвиток як open source, відкритість до змін

Ці та інші особливості і можливості стали основою для нової моделі програмування.

**1.5 Entity Framework Core**

Entity Framework Core (EF Core) являє собою об'єктно-орієнтовану, легковажну і розширювану технологію від компанії Microsoft для доступу до даних. EF Core є ORM-інструментом (object-relational mapping - відображення даних на реальні об'єкти). Тобто EF Core дозволяє працювати базами даних, але є більш високий рівень абстракції: EF Core дозволяє абстрагуватися від самої бази даних і її таблиць і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами, первинними і зовнішніми ключами, але на концептуальному рівні, який нам пропонує Entity Framework, ми вже працюємо з об'єктами.

Entity Framework Core підтримує безліч різних систем баз даних. Таким чином, ми можемо через EF Core працювати з будь-якої СУБД, якщо для неї є потрібний провайдер.

За замовчуванням на даний момент Microsoft надає ряд вбудованих провайдерів: для роботи з MS SQL Server, для SQLite, для PostgreSQL. Також є провайдери від сторонніх постачальників, наприклад, для MySQL.

Також варто відзначити, що EF Core надає універсальний API для роботи з даними. І якщо, наприклад, ми вирішимо змінити цільову СУБД, то основні зміни в проекті будуть стосуватися насамперед конфігурації і настройки підключення до відповідних провайдерам. А код, який безпосередньо працює з даними, отримує дані, додає їх в БД і т.д., залишиться колишнім.

Entity Framework Core багато успадкував від своїх попередників, зокрема, Entity Framework 6. У той же час треба розуміти, що EF Core - це не нова версія по відношенню до EF 6, а зовсім інша технологія, хоча в цілому принципи роботи у них будуть збігатися . Тому в рамках EF Core використовується своя система версій. Поточна версія - 5.0 була випущена в листопаді 2020 року. І технологія продовжує розвиватися.

Як технологія доступу до даних Entity Framework Core може використовуватися на різних платформах стека .NET. Це і стандартні платформи типу Windows Forms, консольні додатки, WPF, UWP і ASP.NET Core. При цьому кроссплатформенная природа EF Core дозволяє задіяти її не тільки на ОС Windows, але і на Linux і Mac OS X.

Центральної концепцією Entity Framework є поняття сутності або entity. Сутність визначає набір даних, які пов'язані з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами і їх колекціями.

Будь-яка сутність, як і будь-який об'єкт з реального світу, має низку властивостей. Наприклад, якщо сутність описує людину, то ми можемо виділити такі властивості, як ім'я, прізвище, зріст, вік. Властивості необов'язково представляють прості дані типу int або string, але можуть також представляти і більш комплексні типи даних. І у кожної сутності може бути одна або кілька властивостей, які будуть відрізняти цю сутність від інших і будуть унікально визначати цю сутність. Подібні властивості називають ключами.

При цьому суті можуть бути пов'язані асоціативної зв'язком один-до-багатьох, один-до-одного і багато-до-багатьох, подібно до того, як в реальній базі даних відбувається зв'язок через зовнішні ключі.

Відмінною рисою Entity Framework Core, як технології ORM, є використання запитів LINQ для вибірки даних з БД. За допомогою LINQ ми можемо створювати різні запити на вибірку об'єктів, в тому числі пов'язаних різними асоціативними зв'язками. А Entity Framework при виконанні запиту транслює вираження LINQ в вирази, зрозумілі для конкретної СУБД (як правило, в вирази SQL).

**1.6 Система управління базами даних**

Система управління базами даних (СУБД) - це комплекс програмно-мовних засобів, що дозволяють створити бази даних і управляти даними. Іншими словами, СУБД - це набір програм, що дозволяє організовувати, контролювати і адмініструвати бази даних. Більшість сайтів не можуть функціонувати без бази даних, тому СУБД використовується практично повсюдно.

Основні функції СУБД:

* керування даними в зовнішній пам'яті (на дисках);
* керування даними в оперативній пам'яті з використанням дискового кешу;
* журналізація змін (збереження історії), резервне копіювання і відновлення бази даних після збоїв;
* підтримка мов БД (мова визначення даних, мова маніпулювання даними).

Кожна СУБД ґрунтується на будь-якої моделі даних, це є одним з ознак класифікації. За моделлю даних СУБД бувають:

1. Ієрархічні. У цій моделі даних використовується уявлення БД у вигляді дерева, що складається з даних різних рівнів.
2. Мережеві. Дана модель є розширенням ієрархічного підходу. Ієрархічна модель має на увазі, що запис-нащадок може мати строго одного предка, в той час як в мережевій структурі нащадок може мати будь-яку кількість предків.
3. Реляційні. СУБД, орієнтовані на організацію даних як набір пов'язаних записів і атрибутів в двовимірної таблиці.
4. Об'єктно-орієнтовані. Для управління БД, заснованими на об'єктної моделі даних. Як правило ґрунтуються на об'єктно-орієнтованих мовах програмування.
5. Об'єктно-реляційні. Об'єднує в собі концепції реляційної моделі з додатковими об'єктно-орієнтованими можливостями.

Сьогодні, як і раніше найбільш популярними при створенні веб-додатків і сервісів залишаються реляційні бази даних. Для керування базами даних використовується мова SQL (Structured Query Language - структурована мова запитів). Спочатку SQL був інструментом роботи користувача з базою даних, однак з часом мова ускладнився і став швидше інструментом розробника, ніж кінцевого користувача.

Різні рейтинги найпопулярніших СУБД очолюють Oracle, MySQL, Microsoft SQL Server, PostgreSQL. Про Microsoft SQL Server і буде йти мова далі

**1.7** **MS SQL Server**

SQL Server є однією з найбільш популярних систем управління базами даних (СКБД) в світі. Дана СУБД підходить для самих різних проектів: від невеликих додатків до великих високонавантажених проектів.

SQL Server був створений компанією Microsoft. Перша версія вийшла в 1987 році. А поточною версією є версія 16, яка вийшла в 2016 році і яка буде використовуватися в поточному керівництві.

SQL Server довгий час був винятково системою управління базами даних для Windows, проте починаючи з версії 16 ця система доступна і на Linux.

SQL Server характеризується такими особливостями як:

* Продуктивність. SQL Server працює дуже швидко.
* Надійність і безпека. SQL Server надає шифрування даних.
* Простота. З даної СУБД відносно легко працювати і вести адміністрування.

Центральним аспектом в MS SQL Server, як і в будь-який СУБД, є база даних. База даних являє сховище даних, організованих певним способом. Нерідко фізично база даних представляє файл на жорсткому диску, хоча таке відповідність необов'язково. Для зберігання і адміністрування баз даних застосовуються системи управління базами даних (database management system) або СУБД (DBMS). І якраз MS SQL Server є однією з такою СУБД.

Для організації баз даних MS SQL Server використовує реляційну модель. Ця модель баз даних була розроблена ще в 1970 році Едгаром Коддом. А на сьогоднішній день вона фактично є стандартом для організації баз даних.

Реляційна модель передбачає зберігання даних у вигляді таблиць, кожна з яких складається з рядків і стовпців. Кожен рядок зберігає окремий об'єкт, а в стовпчиках розміщуються атрибути цього об'єкта.

Для ідентифікації кожного рядка в рамках таблиці застосовується первинний ключ (primary key). В якості первинного ключа може виступати один або декілька стовпців. Використовуючи первинний ключ, ми можемо посилатися на певну рядок в таблиці. Відповідно два рядки не можуть мати один і той же первинний ключ.

Через ключі одна таблиця може бути пов'язана з іншого, тобто між двома таблицями можуть бути організовані зв'язку. А сама таблиця може бути представлена ​​у вигляді відносини ( "relation").

Для взаємодії з базою даних застосовується мова SQL (Structured Query Language). Клієнт (наприклад, зовнішня програма) відправляє запит на мові SQL за допомогою спеціального API. СУБД належним чином інтерпретує і виконує запит, а потім посилає клієнту результат виконання.

Спочатку мова SQL був розроблений в компанії IBM для системи баз даних, яка називалася System / R. При цьому сама мова називався SEQUEL (Structured English Query Language). Хоча в підсумку ні база даних, ні сам язик не були згодом офіційно опубліковані, за традицією сам термін SQL нерідко вимовляють як "сиквел".

У 1979 році компанія Relational Software Inc. розробила першу систему управління баз даних, яка називалася Oracle і яка використовувала мову SQL. У зв'язку з успіхом даного продукту компанія була перейменована в Oracle.

Згодом стали з'являтися інші системи баз даних, які використовували SQL. У підсумку в 1989 році Американський Національний Інститут Стандартів (ANSI) кодифікував мову і опублікував його перший стандарт. Після цього стандарт періодично оновлювався і доповнювався. Останнє його оновлення відбулося в 2011 році. Але незважаючи на наявність стандарту нерідко виробники СУБД використовують свої власні реалізації мови SQL, які трохи відрізняються один від одного.

Виділяються два різновиди мови SQL: PL-SQL і T-SQL. PL-SQL використовується в таких СУБД як Oracle і MySQL. T-SQL (Transact-SQL) застосовується в SQL Server.

**2.Ідея та можливості застосування**

Ідея проекту відображена у самій назві моєї роботи. Я намагався створити продукт за допомогою якого користувачі могли б дізнатися інформацію про певний рід товарів, залишити відгук про нього, а також побачити експертну думку, та рейтинг мого товару на інших платформах. У моєму випадку родом продукту було обрано фільми. Тобто за допомогою мого веб-додатку можна переглянути основну інформацію про той чи інший фільм, переглянути його трейлер, а також побачити оцінки критиків з найпопулярніших платформ таких як Metacritic та Rotten Tomatoes, а також оцінки та коментарі користувачів ресурсу, а також самому оцінити та написати власний відгук. Тобто цей ресурс можна використовувати при пошуку фільму для перегляду. Також за допомогою певних додатків можна значно розширити функціонал ресурсу, де користувач міг би бачити та давати оцінки певного рядку товарів, а також міг би прослідковувати все оцінені товари, а також додавати ряд товарів у фаворити, для подальшого якнайшвидшого оцінювання. Також в майбутньому планується додати функцію завантаження .xls документу з усіма обраними даними.

**3.** **Розробка веб-додатку**

**3.1 Розробка**

Для початку необхідно було створити проект(ASP.NET Core web application) у середовищі розробки, я створював його за запропонованим MVC шаблоном

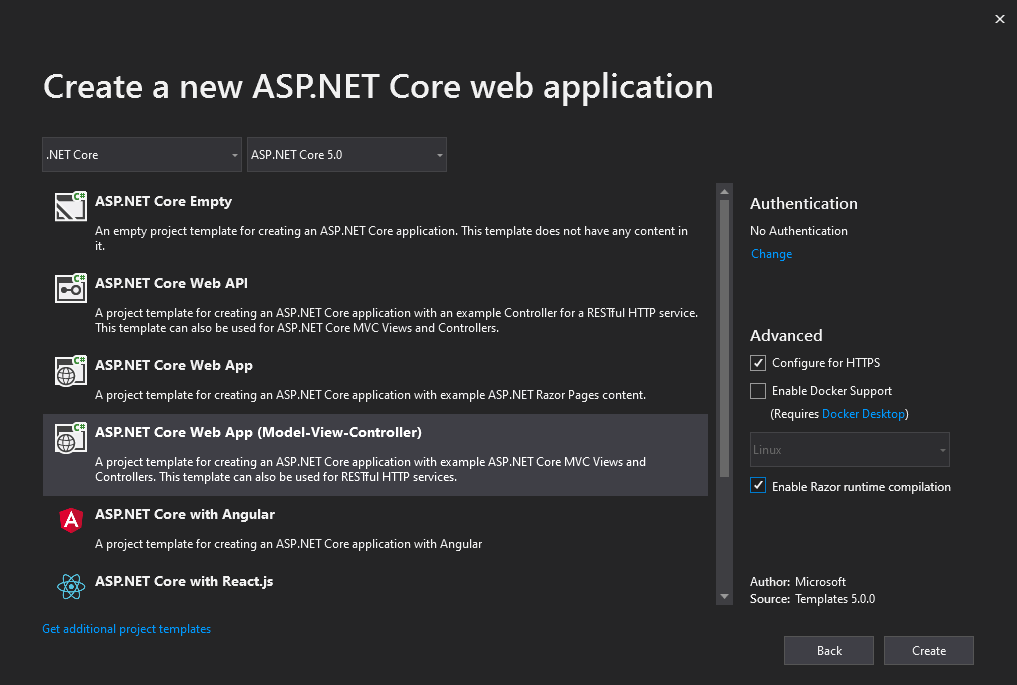


Рис 1. Вікно для створення проекту

Після створення я отримав проект зі базовими необхідними компонентами, сервісами, та підключеними залежностями та структурою, та одразу міг починати роботу. На початку проект мав таку структуру

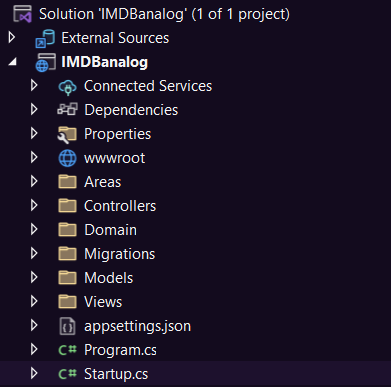


Рис 2. Структура проекту при створенні

Після цього потрібно було додати всі необхідні NuGet пакети. Які були необхідні для розробки проекту, в тому числі пакети для роботи з базами даних. Повний їх список виглядає ось так.

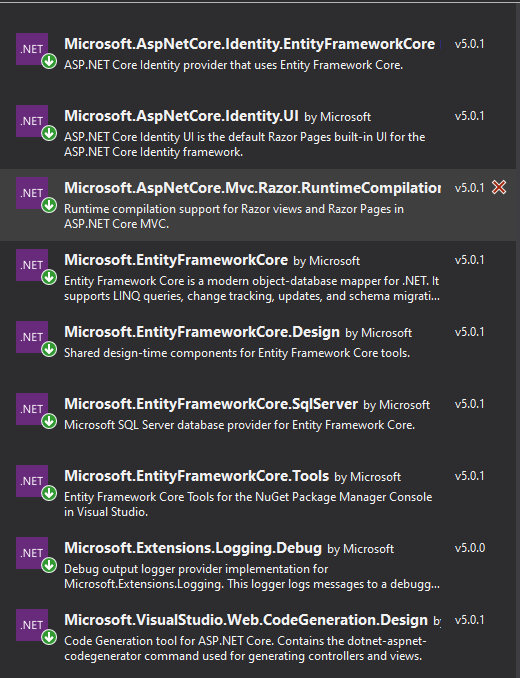


Рис 3.Список встановлених пакетів

Після цього я для початку взявся за авторизацію, для цього створив відповідну папку Areas, підключив спеціальні пакети AspNetCore.Identity. В ході роботи я додав певні поля у модель користувача. Для цього створив спеціальний клас який наслідувався від інтерфейсу IdentityUser. Тобто я додав до таблиці поля FirstName та LastName

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Identity;

namespace IMDBanalog.Areas.Identity.Data

{

public class ApplicationUser : IdentityUser

{

[PersonalData]

[Column(TypeName = "varchar(100)")]

public string FirstName { get; set; }

[PersonalData]

[Column(TypeName = "varchar(100)")]

public string LastName { get; set; }

}

}

А також дещо змінив представлення сторінок для входу та реєстрації (додав у форму відповідні поля). Та дещо змінив стилі. Код представлення для сторінки реєстрації має наступний вигляд

@page

@model RegisterModel

@{

ViewData["Title"] = "Register";

Layout = "~/Areas/Identity/Pages/\_AuthLayot.cshtml";

}

<style>

form{

margin:0px 20px 10px 20px;

}

</style>

<form **asp-route-returnUrl**="@Model.ReturnUrl" method="post">

<div **asp-validation-summary**="All" class="text-danger"></div>

<div class="row">

<div class="col-md-6">

<div class="form-group">

<label **asp-for**="Input.FirstName"></label>

<input **asp-for**="Input.FirstName" class="form-control" />

<span **asp-validation-for**="Input.FirstName" class="text-danger"></span>

</div>

</div>

<div class="col-md-6">

<div class="form-group">

<label **asp-for**="Input.LastName"></label>

<input **asp-for**="Input.LastName" class="form-control" />

<span **asp-validation-for**="Input.LastName" class="text-danger"></span>

</div>

</div>

</div>

<div class="form-group">

<label **asp-for**="Input.Email"></label>

<input **asp-for**="Input.Email" class="form-control" />

<span **asp-validation-for**="Input.Email" class="text-danger"></span>

</div>

<div class="row">

<div class="col-md-6">

<div class="form-group">

<label **asp-for**="Input.Password"></label>

<input **asp-for**="Input.Password" class="form-control" />

<span **asp-validation-for**="Input.Password" class="text-danger"></span>

</div>

</div>

<div class="col-md-6">

<div class="form-group">

<label **asp-for**="Input.ConfirmPassword"></label>

<input **asp-for**="Input.ConfirmPassword" class="form-control" />

<span **asp-validation-for**="Input.ConfirmPassword" class="text-danger"></span>

</div>

</div>

</div>

<button type="submit" class="btn btn-primary btn-block">Register</button>

</form>

@section Scripts {

<**partial** **name**="\_ValidationScriptsPartial" />

}

В результаті ми маємо ось таку форму в полями які необхідно заповнити

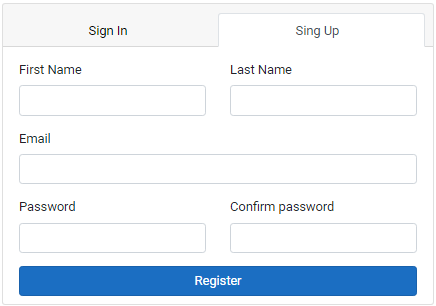


Рис 4 Вигляд сторінки для реєстрації

Схожі дії також відбулися з представленням для сторінки входу. Правда кількість необхідних полів для заповнювання там значно менша. І сама сторінка має такий вигляд.

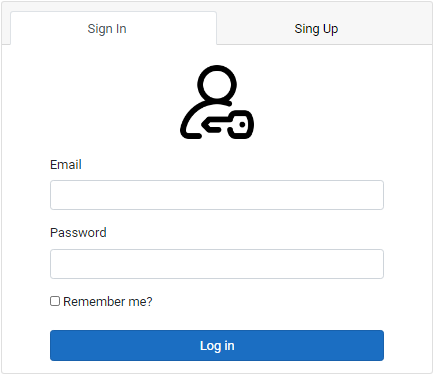


Рис 5. Вигляд сторінки для входу

Код відповідного представлення дуже схожий з представленням для реєстрації. І виглядає ось так

@model LoginModel

@{

ViewData["Title"] = "Log in";

Layout = "~/Areas/Identity/Pages/\_AuthLayot.cshtml";

}

<div class="col-md-10 offset-md-1">

<section>

<div class="login-form-icon">

<img src="/img/login.png" alt="Alternate Text" class="img-thumbnail"style="width:100px;" />

</div>

<form id="account" method="post">

<div **asp-validation-summary**="All" class="text-danger"></div>

<div class="form-group">

<label **asp-for**="Input.Email"></label>

<input **asp-for**="Input.Email" class="form-control" />

<span **asp-validation-for**="Input.Email" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Input.Password"></label>

<input **asp-for**="Input.Password" class="form-control" />

<span **asp-validation-for**="Input.Password" class="text-danger"></span>

</div>

<div class="form-group">

<div class="checkbox">

<label **asp-for**="Input.RememberMe">

<input **asp-for**="Input.RememberMe" />

@Html.DisplayNameFor(m => m.Input.RememberMe)

</label>

</div>

</div>

<div class="form-group">

<button type="submit" class="btn btn-primary btn-block">Log in</button>

</div>

</form>

</section>

</div>

@section Scripts {

<**partial** **name**="\_ValidationScriptsPartial" />

}

Також в файлі який відповідає за конфігурування IdentityHostingStartup.cs, додав певні параметри для входу та реєстрації. А також підключився до бази даних. Все це виглядає ось так

using System;

using IMDBanalog.Areas.Identity.Data;

using IMDBanalog.Data;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.UI;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

[assembly: HostingStartup(typeof(IMDBanalog.Areas.Identity.IdentityHostingStartup))]

namespace IMDBanalog.Areas.Identity

{

public class IdentityHostingStartup : IHostingStartup

{

public void Configure(IWebHostBuilder builder)

{

builder.ConfigureServices((context, services) => {

services.AddDbContext<AuthSystemDBContext>(options =>

options.UseSqlServer(

context.Configuration.GetConnectionString("AuthSystemDBContextConnection")));

services.AddDefaultIdentity<ApplicationUser>(options => {

options.SignIn.RequireConfirmedAccount = false;

options.SignIn.RequireConfirmedEmail = false;

options.Password.RequireUppercase = false;

options.Password.RequireLowercase = false;

options.Password.RequireNonAlphanumeric = false;

})

.AddRoles<IdentityRole>()

.AddEntityFrameworkStores<AuthSystemDBContext>();

});

}

}

}

В результаті структура частини, що відповідала за реєстрацію та авторизацію користувачів отримала ось такий вигляд.

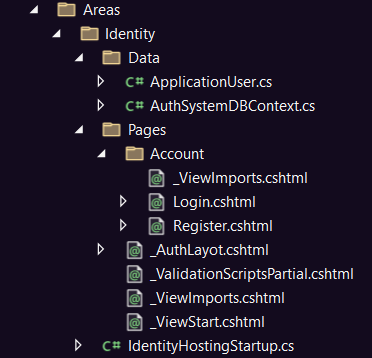


Рис 6. Структура частини, що відповідає за реєстрацію та авторизацію.

Також необхідно прописати у файлі Startup.cs, в частині Configure напступні команди

app.UseAuthentication();

app.UseAuthorization();

Після цього створив моделі (Comment.cs, Film.cs) для нашої бази даних, інтерфейси, які описують всі дії над нашими моделями IComment.cs, IFilm.cs, а також класи які наслідують наші інтерфейси CommentRepository.cs, FilmRepository.cs.

Модель Film.cs має наступний вигляд

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Domain.Entities

{

public class Film

{

public int Id {get;set;}

public string Name { get; set; }

public string ReleaseDate { get; set; }

public string Director { get; set; }

public string Genres { get; set; }

public string Rating { get; set; }

public int Runtime { get; set; }

public int Metacritics { get; set; }

public int RottenTomatoes { get; set; }

public string image { get; set; }

public string Summary { get; set; }

public virtual ICollection<Comment> UserRate { get; set; }

}

}

В ній прописані всі поля нашої таблиці такі як назва, дата релізу, режисер, жанри, віковий рейтинг, тривалість, оцінки від критиків з двох найпопулярніших порталів, опис а також список коментарів користувачів.

Інтерфейс IFilm.cs є переліком всіх операцій які можливі над елементами моделі фільмів, і має такий вигляд

using IMDBanalog.Domain.Entities;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Domain.Repositories.Interface

{

public interface IFilm

{

public List<Film> GetItems();

public Film GetItem(int id);

public void CreateItem(Film item);

public void DeleteIteam(int id);

public void UpdateItem(Film item);

}

}

Тобто допомагає отримати колекцію моделей з моделі, або окремий елемент, а також створити, видалити та редагувати елемент. Реалізація цих функцій знаходиться у файлі FilmRepository.cs

using IMDBanalog.Domain.Entities;

using IMDBanalog.Domain.Repositories.Interface;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Domain.Repositories

{

public class FilmRepository : IFilm

{

private readonly AppDbContext dbContext;

public FilmRepository()

{

dbContext = new AppDbContext();

}

public List<Film> GetItems()

{

return dbContext.Set<Film>().ToList();

}

public Film GetItem(int id)

{

return dbContext.Set<Film>().Find(id);

}

public void CreateItem (Film item )

{

dbContext.Set<Film>().Add(item);

dbContext.SaveChanges();

}

public void DeleteIteam(int id)

{

Film item = dbContext.Set<Film>().Find(id);

dbContext.Set<Film>().Remove(item);

dbContext.SaveChanges();

}

public void UpdateItem(Film item)

{

dbContext.Entry(item).State = EntityState.Modified;

dbContext.SaveChanges();

}

}

}

Все це справедливо і для моделі коментарів , яка містить такі поля як ід, назву користувача який залишає коментар, оцінку яку він залишив, сам коментар та фільм на який власне залишений цей коментар

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Domain.Entities

{

public class Comment

{

public int Id { get; set; }

public string User { get; set; }

public int Rate { get; set; }

public string Comm { get; set; }

public virtual Film Film { get; set; }

}

}

Інтерфейс який описує всі можливі дії над моделлю коментаря, описує такі ж функції як і для фільму, реалізація також схожа окрім частини зі створенням, вона має ось такий вигляд

public void Create(string name, int rate, Film film)

{

film.UserRate ??= new List<Comment>();

film.UserRate.Add(new Comment()

{

User = name,

Rate = rate

});

dbContext.Set<Film>().Update(film);

dbContext.SaveChanges();

}

Також налаштував контекст бази даних. В якому було прописано під’єднання до бази даних. А також описав елементи цього контексту

using IMDBanalog.Domain.Entities;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Domain

{

public class AppDbContext : DbContext

{

public AppDbContext() : base() {}

public DbSet<Film> Films { get; set; }

public DbSet<Comment> Comments { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Film>().HasMany(x => x.UserRate);

base.OnModelCreating(modelBuilder);

}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer("Server=WIN-266ANC3OJ4P;Database=IMDBanalog;Trusted\_Connection=True;MultipleActiveResultSets=true");

optionsBuilder.EnableSensitiveDataLogging();

}

}

}

В кінцевому результаті структура частини мого проекту, яка відповідала за моделі даних та всі операції над цими моделями, мала ось такий вигляд

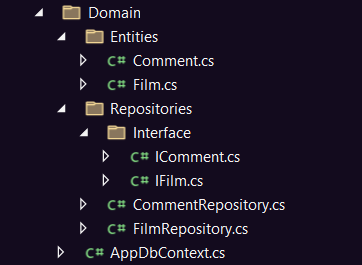


Рис 7. Частина програми, що відповідає за моделі та контекст даних

Для створення таблиць в базі даних, та зміни схем даних я використовував спеціальний механізм Entity Framework Core – міграції . В процесі розробки мною було створено декілька міграцій, адже в часто вносилися зміни у схеми моделей баз даних

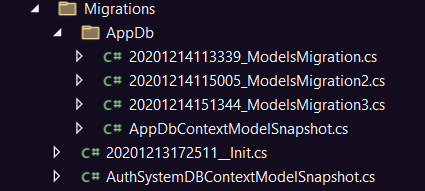


Рис 8. Повний список міграцій

Після цього було додано ролі, в тому числі і роль адміністратора, та присвоєно її одному з існуючих користувачів в базі даних. Фрагмент коду, який знаходиться у файлі Startup.cs та відповідає за створення ролі адміністратора та присвоєння її певному користувачу має ось такий вигляд.

private async Task CreateUserRoles(IServiceProvider serviceProvider)

{

var RoleManager = serviceProvider.GetRequiredService<RoleManager<IdentityRole>>();

var UserManager = serviceProvider.GetRequiredService<UserManager<ApplicationUser>>();

IdentityResult roleResult;

var roleCheck = await RoleManager.RoleExistsAsync("Admin");

if (!roleCheck)

{

roleResult = await RoleManager.CreateAsync(new IdentityRole("Admin"));

}

ApplicationUser user = await UserManager.FindByEmailAsync("cwewqe@gmail.com");

var User = new ApplicationUser();

await UserManager.AddToRoleAsync(user, "Admin");

}

Потім було створено «панель адміністратора» для створення, редагування та видалення елементів наших фільмів в базі даних, за допомогою графічного інтерфейсу. Для цього необхідно було створити відповідний контролер та представлення. Повний список представлень, має такий вигляд

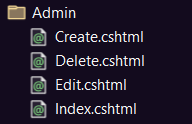


Рис 9. Всі представлення для панелі адміністратора

Реалізація кожного з представлень схожа, створюється форма з відповідними полями, та передається відповідна модель і залежно від операції заповнюються поля. Для прикладу наведу код представлення для операції Create.

@model IMDBanalog.Domain.Entities.Film

@{

ViewData["Title"] = "Create";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<style>

\* {

margin-bottom: 5px;

}

</style>

<form method="post" **asp-action**="Create">

<div class="border p-3">

<div class="form-group row">

<h2 class="text-info pl-3">Add Film</h2>

</div>

<div class="row">

<div class="col-8">

<div class="form-group row">

<div class="col-4">

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

</div>

<div class="col-8">

<input **asp-for**="Name" class="form-control" />

<span **asp-validation-for**="Name" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="ReleaseDate"></label>

</div>

<div class="col-8">

<input **asp-for**="ReleaseDate" class="form-control" />

<span **asp-validation-for**="ReleaseDate" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Director"></label>

</div>

<div class="col-8">

<input **asp-for**="Director" class="form-control" />

<span **asp-validation-for**="Director" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Genres"></label>

</div>

<div class="col-8">

<input **asp-for**="Genres" class="form-control" />

<span **asp-validation-for**="Genres" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Rating"></label>

</div>

<div class="col-8">

<input **asp-for**="Rating" class="form-control" />

<span **asp-validation-for**="Rating" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Runtime"></label>

</div>

<div class="col-8">

<input **asp-for**="Runtime" class="form-control" />

<span **asp-validation-for**="Runtime" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Metacritics"></label>

</div>

<div class="col-8">

<input **asp-for**="Metacritics" class="form-control" />

<span **asp-validation-for**="Metacritics" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="RottenTomatoes"></label>

</div>

<div class="col-8">

<input **asp-for**="RottenTomatoes" class="form-control" />

<span **asp-validation-for**="RottenTomatoes" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="image"></label>

</div>

<div class="col-8">

<input **asp-for**="image" class="form-control" />

<span **asp-validation-for**="image" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Summary"></label>

</div>

<div class="col-8">

<input **asp-for**="Summary" class="form-control" />

<span **asp-validation-for**="Summary" class="text-danger"></span>

</div>

</div>

<div class="form-group col">

<div class="col-8 offset-4 row">

<div class="col">

<input type="submit" class="btn btn-info w-100" value="Create" />

</div>

<div class="col">

<a **asp-action**="Index" class="btn btn-success w-100"><i class="fas fa-sign-out-alt"></i> Back</a>

</div>

</div>

</div>

</div>

<div class="col-4">

</div>

</div>

</div>

</form>

@section Scripts{

@{

<**partial** **name**="\_ValidationScriptsPartial" />

}

}

Для створення зв’язку між представленням та базою даних було створено відповідний контролер . В контролері було описано поведінку при різних типах запитів(POST, GET) на різних типах представлень(Index, Edit,Create, Delete). Також були створенні представлення, які графічно відображали дані та давали можливість їх змінити. Контролер має ось такий вигляд.

using IMDBanalog.Domain;

using IMDBanalog.Domain.Entities;

using IMDBanalog.Domain.Repositories.Interface;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Controllers

{

[Authorize(Roles = "Admin")]

public class AdminController : Controller

{

private readonly IFilm \_allfilms;

private readonly IComment \_allcomment;

public AdminController(IFilm allfilms, IComment allcoment)

{

\_allcomment = allcoment;

\_allfilms = allfilms;

}

public IActionResult Index()

{

return View(\_allfilms.GetItems().ToList());

}

public IActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult Create(Film obj)

{

if (ModelState.IsValid)

{

\_allfilms.CreateItem(obj);

return RedirectToAction("Index");

}

return View(obj);

}

public IActionResult Edit(int id)

{

var obj = \_allfilms.GetItem(id);

if (obj == null)

{

return NotFound();

}

return View(obj);

}

//POST - EDIT

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult Edit(Film obj)

{

if (ModelState.IsValid)

{

\_allfilms.UpdateItem(obj);

return RedirectToAction("Index");

}

return View(obj);

}

public IActionResult Delete(int id)

{

var obj= \_allfilms.GetItem(id);

return View(obj);

}

//POST - DELETE

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult DeletePost(int id)

{

\_allfilms.DeleteIteam(id);

return RedirectToAction("Index");

}

}

}

В результаті наша панель адміністратора отримала ось такий вигляд.

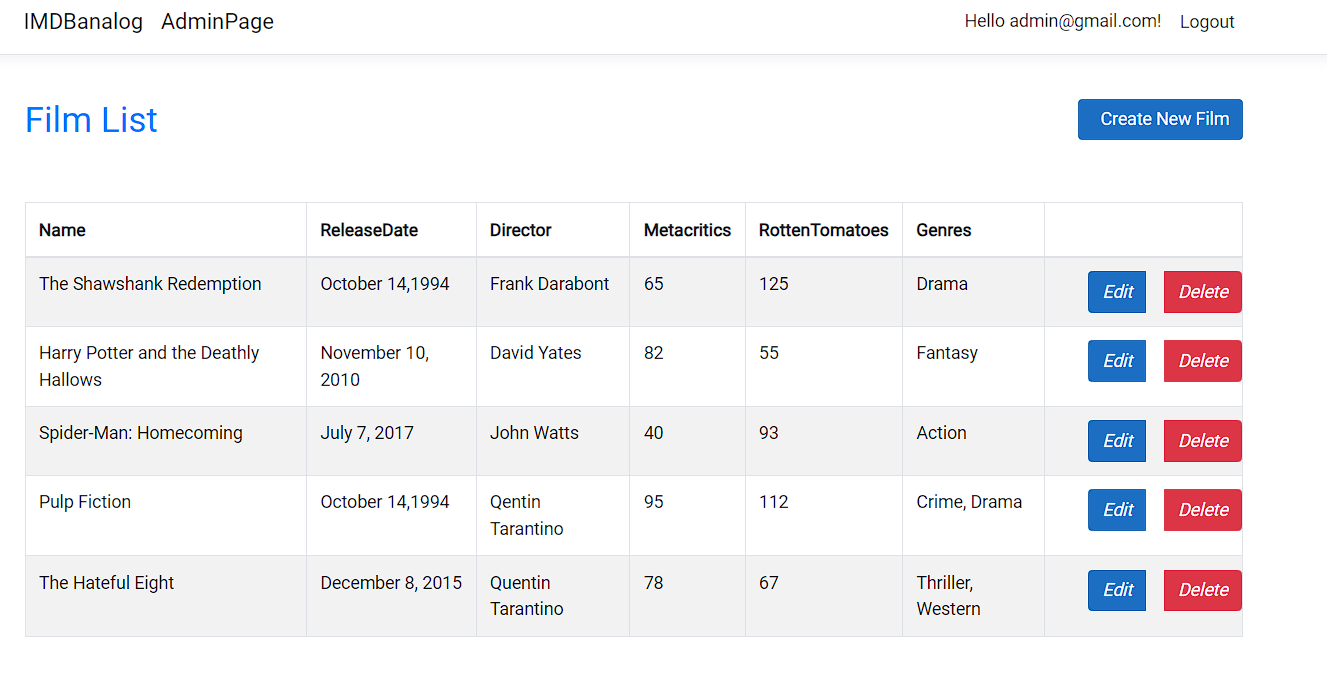


Рис 10. Вигляд панелі адміністратора

Після цього я перейшов до створення контролеру та представлення головної сторінки. В їх створенні не було ніяких особливих нюансів, просто необхідно було передати до контролеру контекст нашої бази даних, і у представленні за просто діставати необхідні дані з моделей. Зазначу, що представлення розроблялося з допомогою Bootstrap, тому є повністю адаптивним, тобто зберігає коректний вигляд при зміні розміру екрану. Також додам, що представлення створюється з розширенням cshtml, це дозволяє в Нtml код додавати вставки С# коду, за допомогою Razor. Це має такий вигляд.

@foreach (var item in Model)

{

int rating;

if (item.UserRate.Count == 0) { rating = 0; }

else{ rating = (int)Math.Round((item.UserRate?.Select(x => x.Rate).Average() ?? 0) \* 10); }

<div class="col-sm-12 border rounded" style="margin:10px 0px 10px 0px; ">

<div class="film" style="display:flex;">

<img src="@item.image" alt="Alternate Text" style="height:250px; margin:10px 10px 10px 0px" />

<div class="film-context " style="width:100%; margin:10px 10px 10px 0px">

<div class="rating" style="display:flex; justify-content:space-between;width:100% ">

<h3> <a **asp-controller**="Film" **asp-route-Id**="@item.Id">@item.Name</a></h3>

<div>

<label><img src="/img/metacritic-icon.svg" alt="Alternate Text" style="width:20px;" /> @item.Metacritics</label>

<label><img src="/img/rotten tomato.png" alt="Alternate Text" style="width:20px;" /> @item.RottenTomatoes</label>

<label><img src="~/img/tomatouser.png" alt="Alternate Text" style="width:20px;" /> @rating</label>

</div></div>

<div class="film-text">

<p class="text-secondary">@item.ReleaseDate, Rating: @item.Rating</p>

<p>Director: @item.Director</p>

<p>Genres: @item.Genres , @item.Runtime min </p>

<p>Summary: @item.Summary</p>

</div></div></div></div>

}

Після налаштування головної сторінки було створено контролер та представлення для сторінок фільмів. В це представлення ми передаємо список моделей фільмів, та виводимо за допомогою циклу foreach всі елементи даної моделі, стилізуючи їх. Також було додано певні елементи для стилізації в результаті головна сторінка отримала такий вигляд

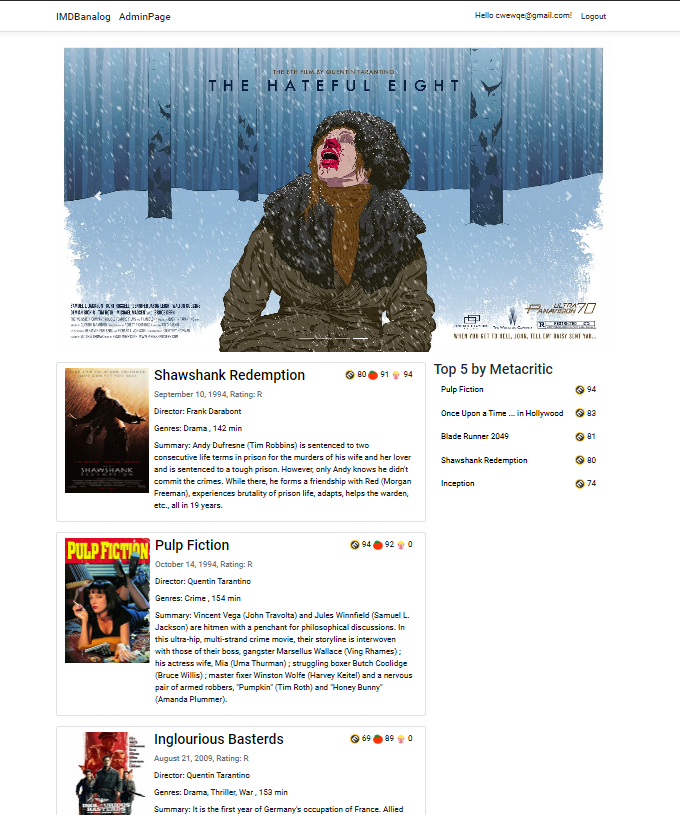


Рис 11.Головна сторінка веб-додатку

На ній ми бачимо слайдер з постерів головних фільмів предсталених на ресурсі, список самих фільмів, з відомостями про них, а також колонку з найкращими фільмами за версією Metacritic з посиланнями на трейлери.

Контролер так само пов’язує нашу модель даних з представленням. У випадку фільмів у представлення ми масив елементів типу Film. Та має ось такий вигляд

using IMDBanalog.Domain;

using IMDBanalog.Domain.Entities;

using IMDBanalog.Domain.Repositories.Interface;

using IMDBanalog.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Logging;

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Controllers

{

[Authorize]

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

private readonly IFilm \_allfilms;

private readonly AppDbContext \_appDbContext;

public HomeController(ILogger<HomeController> logger, IFilm allfilms, AppDbContext appDbContext)

{

\_logger = logger;

\_allfilms = allfilms;

\_appDbContext = appDbContext;

}

public IActionResult Index()

{

var films = \_appDbContext.Films.Include(x=>x.UserRate).ToList();

return View(films);

}

public IActionResult Privacy()

{

return View();

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

Після цього я прийнявся за створення представлення для фільмів. В нього ми передаємо лише модель фільму, а не масив моделей. І заповнюємо поля з необхідної моделі . Код представлення фільмів має такий вигляд

@model Film

@using System.Security.Claims;

@{

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

}

<style>

@@import url('https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,500;1,400;1,500&display=swap');

@@import url('https://fonts.googleapis.com/css2?family=Playfair+Display:ital,wght@1,600&display=swap');

\* {

font-family: 'Roboto', sans-serif;

}

.rate img{

margin-left:4px;

}

</style>

@{

var rating = Math.Round((Model.UserRate?.Select(x => x.Rate).Average() ??0)\*10);

}

<div class="info" style="display:flex; justify-content:space-between">

<div class="text">

<p style="font-size:50px; text-align:center; font-family: 'Playfair Display', serif; margin-bottom:2px">@Model?.Name</p>

<p class="font-italic" style="color:grey;"> @Model.ReleaseDate | Rating: @Model.Rating | @Model.Genres | @Model.Runtime min | Director: @Model.Director</p>

</div>

<div style="padding-top: 20px ">

<label><img src="/img/metacritic-icon.svg" alt="Alternate Text" style="width:50px;" /> @Model.Metacritics</label>

<label><img src="/img/rotten tomato.png" alt="Alternate Text" style="width:50px;" /> @Model.RottenTomatoes </label>

<label><img src="~/img/tomatouser.png" alt="Alternate Text" style="width:50px;" /> @rating</label>

</div>

</div>

<div class="main\_ph">

<img src="/img/@Model.Id/main.png" alt="Alternate Text" style="width:100%;" />

</div>

<div class="container">

<div class="row" style="margin-top: 30px;">

<div class="col-sm-12 border">

<p style="font-family: 'Playfair Display', serif; font-size: 27px">@Model.Summary</p>

</div>

<div class=" rate" style="display:flex; justify-content:flex-end; width:100%;margin-top:5px;">

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(1)" onclick="rateCheck(1)" />

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(2)" onclick="rateCheck(2)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(3)" onclick="rateCheck(3)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(4)" onclick="rateCheck(4)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(5)" onclick="rateCheck(5)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(6)" onclick="rateCheck(6)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(7)" onclick="rateCheck(7)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(8)" onclick="rateCheck(8)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(9)" onclick="rateCheck(9)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(10)" onclick="rateCheck(10)"/>

<form **asp-action**="Create" method="post" style="display:flex ; height: 26px; margin-left:5px">

<input type="text" id="userName" name="name" value="@User.Identity.Name" style="display:none; height:1px" /><br />

<input type="number" id="rate" name="rate" style="display:none " /><br />

<input type="number" id="modelId" name="id" value="@Model.Id" style="display:none; height:1px" /><br />

<input class="btn-dark" type="submit" value="Rate" style="border:1px; border-radius:5px; width:100px"/>

</form>

</div>

<div style="display:flex; justify-content:space-between; width:100%; margin-top:20px">

<img src="/img/@Model.Id/1.jpg" class="img-thumbnail" style="max-height:200px;" alt="Alternate Text" />

<img src="/img/@Model.Id/2.jpg" class="img-thumbnail" style="max-height:200px;" alt="Alternate Text" />

<img src="/img/@Model.Id/3.png" class="img-thumbnail" style="max-height:200px;" alt="Alternate Text" />

</div>

@{

var userName = User.Identity.Name;

var a = userName;

}

</div>

</div>

@{

var list\_comm = Model.UserRate.Select(x => x.Comm).ToList();

}

<script>

var checked = false;

var tempimage = 0;

var tmp = 0;

function check( n)

{

console.log(n)

tmp = n;

checked = false;

for (let i = 0; i < n; i++) {

let k = $(".rate-star")[i];

$(".rate").find(k).attr('src', '/img/blackstar.svg')

}

checked = true;

}

document.onmouseover = function (event) {

if (event.target.className != "rate-star" && checked == true) {

for (let i = 0; i < tempimage; i++) {

let k = $(".rate-star")[i];

$(".rate").find(k).attr('src', '/img/blackstar.svg')

}

for (let n = tempimage; n < 10; n++ && checked == true) {

let k = $(".rate-star")[n];

$(".rate").find(k).attr('src', '/img/star.svg')

}

}

else {

for (let i = 0; i < tmp; i++ && checked == true) {

let k = $(".rate-star")[i];

$(".rate").find(k).attr('src', '/img/blackstar.svg')

}

for (let n = tmp; n < 10; n++ && checked == true) {

let k = $(".rate-star")[n];

$(".rate").find(k).attr('src', '/img/star.svg')

}

}

}

function rateCheck(n) {

checked = true;

tempimage = n;

let item = document.getElementById('rate')

for (let i = 0; i < n; i++) {

let k = $(".rate-star")[i];

$(".rate").find(k).attr('src', '/img/blackstar.svg')

}

item.value = n;

}

</script>

Для користувача це представлення має ось такий вигляд. Як видно воно містить ряд фото, опис фільму, певні відомості про фільм. А також рейтинг фільмів як від критиків з ряду порталів так і користувачів. Також користувач може і сам оцінити відповідний фільм. Обравши оцінку, та натичнувший відповідну кнопку.

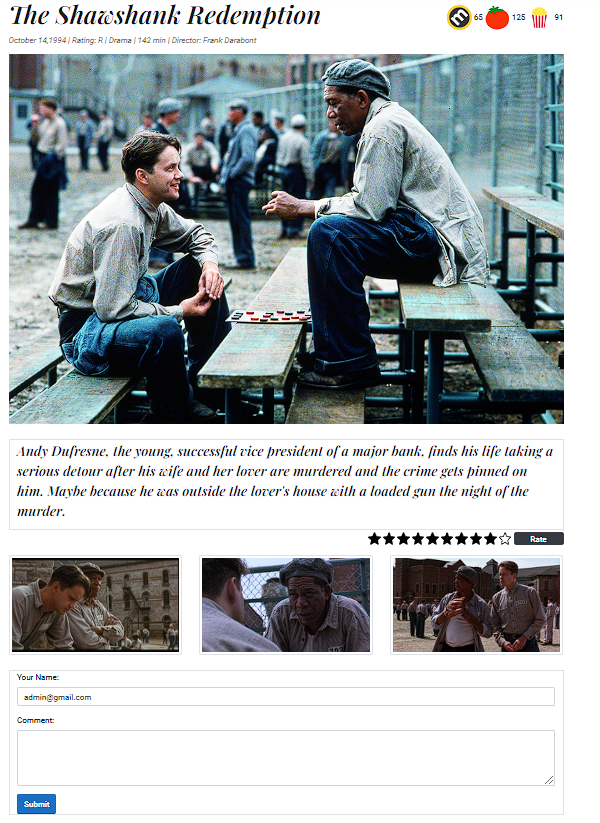


Рис 12. Сторінка фільму

Контролер який пов’язує представлення сторінки з моделлю бази даних має такий вигляд

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Controllers

{

public class FilmController : Controller

{

private readonly IFilm \_allfilms;

private readonly IComment \_allcomment;

private readonly AppDbContext \_appDbContext;

public FilmController(IFilm allfilms, IComment allcoment,AppDbContext appDbContext)

{

\_allcomment = allcoment;

\_allfilms = allfilms;

\_appDbContext = appDbContext;

}

// GET: FilmController

public ActionResult Index(int id)

{

var film = \_appDbContext.Films.Include(f=>f.UserRate).FirstOrDefault(f=>f.Id ==id);

if(film!=null)

{

}

return View(film);

}

[HttpPost]

[ValidateAntiForgeryToken]

// GET: FilmController/Details/5

public ActionResult Details(int id)

{

return View();

}

// GET: FilmController/Create

public ActionResult Create()

{

return View();

}

// POST: FilmController/Create

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create(string name, int rate, int id )

{

Film film = \_allfilms.GetItem(id);

\_allcomment.Create(name, rate, film);

return RedirectToAction("Index",film);

}

}

}

Головною його функцією є передача моделі при запиті GET, та створення коментаря та додавання його до бази даних, при запиті POST.

**3.2 Можливі доопрацювання**

Як я вже зазначав вище, можливості сайту можна значно збільшити. Як у «глибину» та і в «ширину», а також можливо додати певні мінорні зміни(додати paging та сортування за одним з типів оцінок). Що стосується «глибини» можливо додавати такі поля як режисер та жанр(або й навіть віковий рейтинг) не як стрічкову, а як або об’єкти певної моделі, або як масиви цих об’єктів(у випадку жанрів), і згодом можна буде зробити як пошук по даним полям, так і окремі представлення де будуть лише фільми певного режисера, або жанру. Також можливо додати можливість додавати не просто рейтинг, а й текстові коментарі, так окреме представлення для них. Також додати можливість показувати, вже оцінені фільми та фільми, які хочеш переглянути. І це тільки перше, що приходить в голову.

Що стосується розширення функціоналу, можливо додати інші категорії продуктів, та в них реалізувати схожий функціонал, як і з фільмами. Це може бути будь- що, правда логічно було б, аби вони поєднувалися хоч якоюсь концепцією. Можливо також дати користувачу створювати користувачу свій продукт, певного типу, та додавати його на сторінку після дозволю адміністратора.

Як видно функціонал можна розширювати та поглиблювати доволі сильно. На даний момент я б назвав проект базою від якої можна відштовхуватися, та послідовно додавати різні елементи. Тому можливих доопрацювань є безліч. Багато цілком логічних з них я навів вище

**Висновок**

Під час роботи над додатком мені довелося ознайомитися з рядом певних технологій пов’язаних мовою програмування С# та платформою .Net. Дослідив їх особливості в тому числі фреймворків ASP.NET Core та Еntity Framework Core . А також з таким шаблоном програмування як MVC. Оцінив переваги та недоліки даного стеку технологій. Можу сказати, що він надає розробнику дуже багато можливостей для розробки веб-додатків. Дає можливості чітко структурувати всі необхідні дані. Архітектура MVC, яка до речі використовується не тільки при створенні веб-додатків, інтуїтивно зрозуміла, і хоч в процесі розробки в мене виникали труднощі з розумінням, але з часом все стало просто та зрозуміло.

На основі всіх перелічених технологій а також СУБД Microsoft SQL, мною було створено веб-додаток як для перегляду оцінок продукції так і для власного оцінювання продукції. Він має необхідний базовий функціонал та великий простір для подальшого поглиблення та розширення функціоналу.

В підсумку можу сказати розробка за допомогою всіх цих компонент проста та зрозуміла, але підходить не для кожного проекту, адже має свої мінуси, які потрібно враховувати на етапі планування.

**Список використаної літератури**

**.**

**1.Метаніт – режим:** [**https://metanit.com/**](https://metanit.com/)

**2.Хабр - режим доступу:** [**https://habr.com/**](https://habr.com/)

**3.Портал Stackoverflow – Режим доступу:** [**https://stackoverflow.com/**](https://stackoverflow.com/)

4. **Аdam Freeman Pro ASP.NET Core MVC 2 - 2019. – 1010с.**

5. **Adam Freeman** **Pro Entity Framework Core 2 for ASP.NET Core MVC - 2019. – 626с.**

**6. Jeffrey Richter CLR via C# (Developer Reference) – 2013. – 896с.**

**7. Andrew Troelsen Pro C# 8 with .NET Core 3 – 2020. – 1262с.**

**8. Herbert Schildt The Complete Reference C# 4.0 – 2011. – 1056с.**

**Додаток**

**Додаток 1**

Program. Cs

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog

{

public class Program

{

public static void Main(string[] args)

{

CreateHostBuilder(args).Build().Run();

}

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder =>

{

webBuilder.UseStartup<Startup>();

});

}

}

**Додаток 2**

Startup.cs

using IMDBanalog.Areas.Identity.Data;

using IMDBanalog.Domain;

using IMDBanalog.Domain.Repositories;

using IMDBanalog.Domain.Repositories.Interface;

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.HttpsPolicy;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog

{

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.AddControllersWithViews();

services.AddRazorPages();

services.AddDbContext<AppDbContext>();

services.AddTransient<IFilm, FilmRepository>();

services.AddTransient<IComment, CommentRepository>();

services.AddHttpContextAccessor();

}

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

public void Configure(IApplicationBuilder app, IWebHostEnvironment env, IServiceProvider service)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

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

// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

endpoints.MapRazorPages();

});

CreateUserRoles(service).Wait();

}

private async Task CreateUserRoles(IServiceProvider serviceProvider)

{

var RoleManager = serviceProvider.GetRequiredService<RoleManager<IdentityRole>>();

var UserManager = serviceProvider.GetRequiredService<UserManager<ApplicationUser>>();

IdentityResult roleResult;

var roleCheck = await RoleManager.RoleExistsAsync("Admin");

if (!roleCheck)

{

roleResult = await RoleManager.CreateAsync(new IdentityRole("Admin"));

}

ApplicationUser user = await UserManager.FindByEmailAsync("cwewqe@gmail.com");

var User = new ApplicationUser();

await UserManager.AddToRoleAsync(user, "Admin");

}

}

}

**Додаток 3**

ApplicationUser.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Identity;

namespace IMDBanalog.Areas.Identity.Data

{

public class ApplicationUser : IdentityUser

{

[PersonalData]

[Column(TypeName = "varchar(100)")]

public string FirstName { get; set; }

[PersonalData]

[Column(TypeName = "varchar(100)")]

public string LastName { get; set; }

}

}

AuthSystemDBContext.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using IMDBanalog.Areas.Identity.Data;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

namespace IMDBanalog.Data

{

public class AuthSystemDBContext : IdentityDbContext<ApplicationUser>

{

public AuthSystemDBContext(DbContextOptions<AuthSystemDBContext> options)

: base(options)

{

}

protected override void OnModelCreating(ModelBuilder builder)

{

base.OnModelCreating(builder);

}

}

}

**Додаток 4**

IdentityHostingStartup.cs

using System;

using IMDBanalog.Areas.Identity.Data;

using IMDBanalog.Data;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.UI;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

[assembly: HostingStartup(typeof(IMDBanalog.Areas.Identity.IdentityHostingStartup))]

namespace IMDBanalog.Areas.Identity

{

public class IdentityHostingStartup : IHostingStartup

{

public void Configure(IWebHostBuilder builder)

{

builder.ConfigureServices((context, services) => {

services.AddDbContext<AuthSystemDBContext>(options =>

options.UseSqlServer(

context.Configuration.GetConnectionString("AuthSystemDBContextConnection")));

services.AddDefaultIdentity<ApplicationUser>(options => {

options.SignIn.RequireConfirmedAccount = false;

options.SignIn.RequireConfirmedEmail = false;

options.Password.RequireUppercase = false;

options.Password.RequireLowercase = false;

options.Password.RequireNonAlphanumeric = false;

})

.AddRoles<IdentityRole>()

.AddEntityFrameworkStores<AuthSystemDBContext>();

});

}

}

}

**Додаток 5**

Login.cshtml

@page

@model LoginModel

@{

ViewData["Title"] = "Log in";

Layout = "~/Areas/Identity/Pages/\_AuthLayot.cshtml";

}

<div class="col-md-10 offset-md-1">

<section>

<div class="login-form-icon">

<img src="/img/login.png" alt="Alternate Text" class="img-thumbnail"style="width:100px;" />

</div>

<form id="account" method="post">

<div **asp-validation-summary**="All" class="text-danger"></div>

<div class="form-group">

<label **asp-for**="Input.Email"></label>

<input **asp-for**="Input.Email" class="form-control" />

<span **asp-validation-for**="Input.Email" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="Input.Password"></label>

<input **asp-for**="Input.Password" class="form-control" />

<span **asp-validation-for**="Input.Password" class="text-danger"></span>

</div>

<div class="form-group">

<div class="checkbox">

<label **asp-for**="Input.RememberMe">

<input **asp-for**="Input.RememberMe" />

@Html.DisplayNameFor(m => m.Input.RememberMe)

</label>

</div>

</div>

<div class="form-group">

<button type="submit" class="btn btn-primary btn-block">Log in</button>

</div>

</form>

</section>

</div>

@section Scripts {

<**partial** **name**="\_ValidationScriptsPartial" />

}

**Додаток 6**

Register.cshtml

@page

@model RegisterModel

@{

ViewData["Title"] = "Register";

Layout = "~/Areas/Identity/Pages/\_AuthLayot.cshtml";

}

<style>

form{

margin:0px 20px 10px 20px;

}

</style>

<form **asp-route-returnUrl**="@Model.ReturnUrl" method="post">

<div **asp-validation-summary**="All" class="text-danger"></div>

<div class="row">

<div class="col-md-6">

<div class="form-group">

<label **asp-for**="Input.FirstName"></label>

<input **asp-for**="Input.FirstName" class="form-control" />

<span **asp-validation-for**="Input.FirstName" class="text-danger"></span>

</div>

</div>

<div class="col-md-6">

<div class="form-group">

<label **asp-for**="Input.LastName"></label>

<input **asp-for**="Input.LastName" class="form-control" />

<span **asp-validation-for**="Input.LastName" class="text-danger"></span>

</div>

</div>

</div>

<div class="form-group">

<label **asp-for**="Input.Email"></label>

<input **asp-for**="Input.Email" class="form-control" />

<span **asp-validation-for**="Input.Email" class="text-danger"></span>

</div>

<div class="row">

<div class="col-md-6">

<div class="form-group">

<label **asp-for**="Input.Password"></label>

<input **asp-for**="Input.Password" class="form-control" />

<span **asp-validation-for**="Input.Password" class="text-danger"></span>

</div>

</div>

<div class="col-md-6">

<div class="form-group">

<label **asp-for**="Input.ConfirmPassword"></label>

<input **asp-for**="Input.ConfirmPassword" class="form-control" />

<span **asp-validation-for**="Input.ConfirmPassword" class="text-danger"></span>

</div>

</div>

</div>

<button type="submit" class="btn btn-primary btn-block">Register</button>

</form>

@section Scripts {

<**partial** **name**="\_ValidationScriptsPartial" />

}

**Додаток 7**

Comment.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Domain.Entities

{

public class Comment

{

public int Id { get; set; }

public string User { get; set; }

public int Rate { get; set; }

public virtual Film Film { get; set; }

}

}

**Додаток 7**

Film.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Domain.Entities

{

public class Film

{

public int Id {get;set;}

public string Name { get; set; }

public string ReleaseDate { get; set; }

public string Director { get; set; }

public string Genres { get; set; }

public string Rating { get; set; }

public int Runtime { get; set; }

public int Metacritics { get; set; }

public int RottenTomatoes { get; set; }

public string image { get; set; }

public string Summary { get; set; }

public virtual ICollection<Comment> UserRate { get; set; }

}

}

**Додаток 8**

IComment.cs

using IMDBanalog.Domain.Entities;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Domain.Repositories.Interface

{

public interface IComment

{

public List<Comment> GetItems();

public Comment GetItem(int id);

public void Create(string name, int rate, Film film);

public void Delete(int id);

public void UpdateItem(Comment item);

}

}

**Додаток 9**

IFilm.cs

using IMDBanalog.Domain.Entities;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Domain.Repositories.Interface

{

public interface IFilm

{

public List<Film> GetItems();

public Film GetItem(int id);

public void CreateItem(Film item);

public void DeleteIteam(int id);

public void UpdateItem(Film item);

}

}

**Додаток 10**

CommentRepository.cs

using IMDBanalog.Domain.Entities;

using IMDBanalog.Domain.Repositories.Interface;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Domain.Repositories

{

public class CommentRepository : IComment

{

private readonly AppDbContext dbContext;

public CommentRepository()

{

dbContext = new AppDbContext();

}

public List<Comment> GetItems()

{

return dbContext.Set<Comment>().ToList();

}

public Comment GetItem(int id)

{

return dbContext.Set<Comment>().Find(id);

}

public void Create(string name, int rate, Film film)

{

film.UserRate ??= new List<Comment>();

film.UserRate.Add(new Comment()

{

User = name,

Rate = rate

});

dbContext.Set<Film>().Update(film);

dbContext.SaveChanges();

}

public void Delete(int id)

{

var item = dbContext.Set<Comment>().Find(id);

dbContext.Set<Comment>().Remove(item);

dbContext.SaveChanges();

}

public void UpdateItem(Comment item)

{

dbContext.Entry(item).State = EntityState.Modified;

dbContext.SaveChanges();

}

}

}

**Додаток 11**

FilmRepository.cs

using IMDBanalog.Domain.Entities;

using IMDBanalog.Domain.Repositories.Interface;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Domain.Repositories

{

public class FilmRepository : IFilm

{

private readonly AppDbContext dbContext;

public FilmRepository()

{

dbContext = new AppDbContext();

}

public List<Film> GetItems()

{

return dbContext.Set<Film>().ToList();

}

public Film GetItem(int id)

{

return dbContext.Set<Film>().Find(id);

}

public void CreateItem (Film item )

{

dbContext.Set<Film>().Add(item);

dbContext.SaveChanges();

}

public void DeleteIteam(int id)

{

Film item = dbContext.Set<Film>().Find(id);

dbContext.Set<Film>().Remove(item);

dbContext.SaveChanges();

}

public void UpdateItem(Film item)

{

dbContext.Entry(item).State = EntityState.Modified;

dbContext.SaveChanges();

}

}

}

**Додаток 12**

AppDbContext.cs

using IMDBanalog.Domain.Entities;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Domain

{

public class AppDbContext : DbContext

{

public AppDbContext() : base() {}

public DbSet<Film> Films { get; set; }

public DbSet<Comment> Comments { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Film>().HasMany(x => x.UserRate);

base.OnModelCreating(modelBuilder);

}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer("Server=WIN-266ANC3OJ4P;Database=IMDBanalog;Trusted\_Connection=True;MultipleActiveResultSets=true");

optionsBuilder.EnableSensitiveDataLogging();

}

}

}

**Додаток 13**

AdminController.cs

using IMDBanalog.Domain;

using IMDBanalog.Domain.Entities;

using IMDBanalog.Domain.Repositories.Interface;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Controllers

{

[Authorize(Roles = "Admin")]

public class AdminController : Controller

{

private readonly IFilm \_allfilms;

private readonly IComment \_allcomment;

public AdminController(IFilm allfilms, IComment allcoment)

{

\_allcomment = allcoment;

\_allfilms = allfilms;

}

public IActionResult Index()

{

return View(\_allfilms.GetItems().ToList());

}

public IActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult Create(Film obj)

{

if (ModelState.IsValid)

{

\_allfilms.CreateItem(obj);

return RedirectToAction("Index");

}

return View(obj);

}

public IActionResult Edit(int id)

{

var obj = \_allfilms.GetItem(id);

if (obj == null)

{

return NotFound();

}

return View(obj);

}

//POST - EDIT

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult Edit(Film obj)

{

if (ModelState.IsValid)

{

\_allfilms.UpdateItem(obj);

return RedirectToAction("Index");

}

return View(obj);

}

public IActionResult Delete(int id)

{

var obj= \_allfilms.GetItem(id);

return View(obj);

}

//POST - DELETE

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult DeletePost(int id)

{

\_allfilms.DeleteIteam(id);

return RedirectToAction("Index");

}

}

}

**Додаток 14**

FilmController.cs

using IMDBanalog.Domain;

using IMDBanalog.Domain.Entities;

using IMDBanalog.Domain.Repositories.Interface;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Controllers

{

public class FilmController : Controller

{

private readonly IFilm \_allfilms;

private readonly IComment \_allcomment;

private readonly AppDbContext \_appDbContext;

public FilmController(IFilm allfilms, IComment allcoment,AppDbContext appDbContext)

{

\_allcomment = allcoment;

\_allfilms = allfilms;

\_appDbContext = appDbContext;

}

// GET: FilmController

public ActionResult Index(int id)

{

var film = \_appDbContext.Films.Include(f=>f.UserRate).FirstOrDefault(f=>f.Id ==id);

if(film!=null)

{

}

return View(film);

}

[HttpPost]

[ValidateAntiForgeryToken]

// GET: FilmController/Details/5

public ActionResult Details(int id)

{

return View();

}

// GET: FilmController/Create

public ActionResult Create()

{

return View();

}

// POST: FilmController/Create

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create(string name, int rate, int id )

{

Film film = \_allfilms.GetItem(id);

\_allcomment.Create(name, rate, film);

return RedirectToAction("Index",film);

}

// GET: FilmController/Edit/5

public ActionResult Edit(int id)

{

return View();

}

// POST: FilmController/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit(int id, IFormCollection collection)

{

try

{

return RedirectToAction(nameof(Index));

}

catch

{

return View();

}

}

// GET: FilmController/Delete/5

public ActionResult Delete(int id)

{

return View();

}

// POST: FilmController/Delete/5

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Delete(int id, IFormCollection collection)

{

try

{

return RedirectToAction(nameof(Index));

}

catch

{

return View();

}

}

}

}

**Додаток 15**

HomeController.cs

using IMDBanalog.Domain;

using IMDBanalog.Domain.Entities;

using IMDBanalog.Domain.Repositories.Interface;

using IMDBanalog.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Logging;

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Threading.Tasks;

namespace IMDBanalog.Controllers

{

[Authorize]

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

private readonly IFilm \_allfilms;

private readonly AppDbContext \_appDbContext;

public HomeController(ILogger<HomeController> logger, IFilm allfilms, AppDbContext appDbContext)

{

\_logger = logger;

\_allfilms = allfilms;

\_appDbContext = appDbContext;

}

public IActionResult Index()

{

var films = \_appDbContext.Films.Include(x=>x.UserRate).ToList();

return View(films);

}

public IActionResult Privacy()

{

return View();

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

**Додаток 16**

Admin > Create.schtml

@model IMDBanalog.Domain.Entities.Film

@{

ViewData["Title"] = "Create";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<style>

\* {

margin-bottom: 5px;

}

</style>

<form method="post" **asp-action**="Create">

<div class="border p-3">

<div class="form-group row">

<h2 class="text-info pl-3">Add Film</h2>

</div>

<div class="row">

<div class="col-8">

<div class="form-group row">

<div class="col-4">

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

</div>

<div class="col-8">

<input **asp-for**="Name" class="form-control" />

<span **asp-validation-for**="Name" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="ReleaseDate"></label>

</div>

<div class="col-8">

<input **asp-for**="ReleaseDate" class="form-control" />

<span **asp-validation-for**="ReleaseDate" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Director"></label>

</div>

<div class="col-8">

<input **asp-for**="Director" class="form-control" />

<span **asp-validation-for**="Director" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Genres"></label>

</div>

<div class="col-8">

<input **asp-for**="Genres" class="form-control" />

<span **asp-validation-for**="Genres" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Rating"></label>

</div>

<div class="col-8">

<input **asp-for**="Rating" class="form-control" />

<span **asp-validation-for**="Rating" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Runtime"></label>

</div>

<div class="col-8">

<input **asp-for**="Runtime" class="form-control" />

<span **asp-validation-for**="Runtime" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Metacritics"></label>

</div>

<div class="col-8">

<input **asp-for**="Metacritics" class="form-control" />

<span **asp-validation-for**="Metacritics" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="RottenTomatoes"></label>

</div>

<div class="col-8">

<input **asp-for**="RottenTomatoes" class="form-control" />

<span **asp-validation-for**="RottenTomatoes" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="image"></label>

</div>

<div class="col-8">

<input **asp-for**="image" class="form-control" />

<span **asp-validation-for**="image" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Summary"></label>

</div>

<div class="col-8">

<input **asp-for**="Summary" class="form-control" />

<span **asp-validation-for**="Summary" class="text-danger"></span>

</div>

</div>

<div class="form-group col">

<div class="col-8 offset-4 row">

<div class="col">

<input type="submit" class="btn btn-info w-100" value="Create" />

</div>

<div class="col">

<a **asp-action**="Index" class="btn btn-success w-100"><i class="fas fa-sign-out-alt"></i> Back</a>

</div>

</div>

</div>

</div>

<div class="col-4">

</div>

</div>

</div>

</form>

@section Scripts{

@{

<**partial** **name**="\_ValidationScriptsPartial" />

}

}

**Додаток 17**

Admin>Delete.cshtml

@model IMDBanalog.Domain.Entities.Film

@{

ViewData["Title"] = "Create";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<style>

\* {

margin-bottom: 5px;

}

</style>

<form method="post" **asp-action**="DeletePost">

<div class="border p-3">

<input **asp-for**="Id" hidden />

<div class="form-group row">

<h2 class="text-info pl-3">Delete Film</h2>

</div>

<div class="row">

<div class="col-8">

<div class="form-group row">

<div class="col-4">

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

</div>

<div class="col-8">

<input **asp-for**="Name" class="form-control" />

<span **asp-validation-for**="Name" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="ReleaseDate"></label>

</div>

<div class="col-8">

<input **asp-for**="ReleaseDate" class="form-control" />

<span **asp-validation-for**="ReleaseDate" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Director"></label>

</div>

<div class="col-8">

<input **asp-for**="Director" class="form-control" />

<span **asp-validation-for**="Director" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Genres"></label>

</div>

<div class="col-8">

<input **asp-for**="Genres" class="form-control" />

<span **asp-validation-for**="Genres" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Rating"></label>

</div>

<div class="col-8">

<input **asp-for**="Rating" class="form-control" />

<span **asp-validation-for**="Rating" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Runtime"></label>

</div>

<div class="col-8">

<input **asp-for**="Runtime" class="form-control" />

<span **asp-validation-for**="Runtime" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Metacritics"></label>

</div>

<div class="col-8">

<input **asp-for**="Metacritics" class="form-control" />

<span **asp-validation-for**="Metacritics" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="RottenTomatoes"></label>

</div>

<div class="col-8">

<input **asp-for**="RottenTomatoes" class="form-control" />

<span **asp-validation-for**="RottenTomatoes" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="image"></label>

</div>

<div class="col-8">

<input **asp-for**="image" class="form-control" />

<span **asp-validation-for**="image" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Summary"></label>

</div>

<div class="col-8">

<input **asp-for**="Summary" class="form-control" />

<span **asp-validation-for**="Summary" class="text-danger"></span>

</div>

</div>

<div class="form-group col">

<div class="col-8 offset-4 row">

<div class="col">

<input type="submit" class="btn btn-info w-100" value="Delete" />

</div>

<div class="col">

<a **asp-action**="Index" class="btn btn-success w-100"><i class="fas fa-sign-out-alt"></i> Back</a>

</div>

</div>

</div>

</div>

<div class="col-4">

</div>

</div>

</div>

</form>

@section Scripts{

@{

<**partial** **name**="\_ValidationScriptsPartial" />

}

}

**Додаток 18**

Admin>Edit.cshtml

@model IMDBanalog.Domain.Entities.Film

@{

ViewData["Title"] = "Create";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<style>

\* {

margin-bottom: 5px;

}

</style>

<form method="post" >

<div class="border p-3">

<input **asp-for**="Id" hidden />

<div class="form-group row">

<h2 class="text-info pl-3">Edit Film</h2>

</div>

<div class="row">

<div class="col-8">

<div class="form-group row">

<div class="col-4">

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

</div>

<div class="col-8">

<input **asp-for**="Name" class="form-control" />

<span **asp-validation-for**="Name" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="ReleaseDate"></label>

</div>

<div class="col-8">

<input **asp-for**="ReleaseDate" class="form-control" />

<span **asp-validation-for**="ReleaseDate" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Director"></label>

</div>

<div class="col-8">

<input **asp-for**="Director" class="form-control" />

<span **asp-validation-for**="Director" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Genres"></label>

</div>

<div class="col-8">

<input **asp-for**="Genres" class="form-control" />

<span **asp-validation-for**="Genres" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Rating"></label>

</div>

<div class="col-8">

<input **asp-for**="Rating" class="form-control" />

<span **asp-validation-for**="Rating" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Runtime"></label>

</div>

<div class="col-8">

<input **asp-for**="Runtime" class="form-control" />

<span **asp-validation-for**="Runtime" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Metacritics"></label>

</div>

<div class="col-8">

<input **asp-for**="Metacritics" class="form-control" />

<span **asp-validation-for**="Metacritics" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="RottenTomatoes"></label>

</div>

<div class="col-8">

<input **asp-for**="RottenTomatoes" class="form-control" />

<span **asp-validation-for**="RottenTomatoes" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="image"></label>

</div>

<div class="col-8">

<input **asp-for**="image" class="form-control" />

<span **asp-validation-for**="image" class="text-danger"></span>

</div>

<div class="col-4">

<label **asp-for**="Summary"></label>

</div>

<div class="col-8">

<input **asp-for**="Summary" class="form-control" />

<span **asp-validation-for**="Summary" class="text-danger"></span>

</div>

</div>

<div class="form-group col">

<div class="col-8 offset-4 row">

<div class="col">

<input type="submit" class="btn btn-info w-100" value="Update" />

</div>

<div class="col">

<a **asp-action**="Index" class="btn btn-success w-100"><i class="fas fa-sign-out-alt"></i> Back</a>

</div>

</div>

</div>

</div>

<div class="col-4">

</div>

</div>

</div>

</form>

@section Scripts{

@{

<**partial** **name**="\_ValidationScriptsPartial" />

}

}

**Додаток 19**

Admin>Index.cshtml

@model IEnumerable<Film>

<div class="container p-3">

<div class="row pt-4">

<div class="col-6">

<h2 class="text-primary">Film List</h2>

</div>

<div class="col-6 text-right">

<a **asp-controller**="Admin" **asp-action**="Create" class="btn btn-primary">

<i ></i> &nbsp; Create New Film

</a>

</div>

</div>

<br /><br />

@if (Model.Count() > 0)

{

<table class="table table-bordered table-striped" style="width:100%">

<thead>

<tr>

<th>

Name

</th>

<th>

ReleaseDate

</th>

<th>

Director

</th>

<th>

Metacritics

</th>

<th>

RottenTomatoes

</th>

<th>

Genres

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var obj in Model)

{

<tr>

<td>@obj.Name</td>

<td>@obj.ReleaseDate</td>

<td>@obj.Director</td>

<td>@obj.Metacritics</td>

<td>@obj.RottenTomatoes</td>

<td>@obj.Genres</td>

<td class="text-center">

<div class="w-75 btn-group" role="group">

<a **asp-controller**="Admin" **asp-route-Id**="@obj.Id" **asp-action**="Edit" class="btn btn-primary mx-2">

<i>Edit</i>

</a>

<a **asp-controller**="Admin" **asp-route-Id**="@obj.Id" **asp-action**="Delete" class="btn btn-danger mx-2">

<i> Delete</i>

</a>

</div>

</td>

</tr> }

</tbody>

</table> }

else

{

<p> No category exists.</p>}

</div>

**Додаток 20**

Film > Index.cshtml

@model Film

@using System.Security.Claims;

@{

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

}

<style>

@@import url('https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,500;1,400;1,500&display=swap');

@@import url('https://fonts.googleapis.com/css2?family=Playfair+Display:ital,wght@1,600&display=swap');

\* {

font-family: 'Roboto', sans-serif;

}

.rate img{

margin-left:4px;

}

</style>

@{

var rating = Math.Round((Model.UserRate?.Select(x => x.Rate).Average() ??0)\*10);

}

<div class="info" style="display:flex; justify-content:space-between">

<div class="text">

<p style="font-size:50px; text-align:center; font-family: 'Playfair Display', serif; margin-bottom:2px">@Model?.Name</p>

<p class="font-italic" style="color:grey;"> @Model.ReleaseDate | Rating: @Model.Rating | @Model.Genres | @Model.Runtime min | Director: @Model.Director</p>

</div>

<div style="padding-top: 20px ">

<label><img src="/img/metacritic-icon.svg" alt="Alternate Text" style="width:50px;" /> @Model.Metacritics</label>

<label><img src="/img/rotten tomato.png" alt="Alternate Text" style="width:50px;" /> @Model.RottenTomatoes </label>

<label><img src="~/img/tomatouser.png" alt="Alternate Text" style="width:50px;" /> @rating</label>

</div>

</div>

<div class="main\_ph">

<img src="/img/@Model.Id/main.png" alt="Alternate Text" style="width:100%;" />

</div>

<div class="container">

<div class="row" style="margin-top: 30px;">

<div class="col-sm-12 border">

<p style="font-family: 'Playfair Display', serif; font-size: 27px">@Model.Summary</p>

</div>

<div class=" rate" style="display:flex; justify-content:flex-end; width:100%;margin-top:5px;">

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(1)" onclick="rateCheck(1)" />

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(2)" onclick="rateCheck(2)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(3)" onclick="rateCheck(3)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(4)" onclick="rateCheck(4)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(5)" onclick="rateCheck(5)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(6)" onclick="rateCheck(6)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(7)" onclick="rateCheck(7)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(8)" onclick="rateCheck(8)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(9)" onclick="rateCheck(9)"/>

<img class="rate-star" src="/img/star.svg" alt="Alternate Text" style="height:25px" onmouseover="check(10)" onclick="rateCheck(10)"/>

<form **asp-action**="Create" method="post" style="display:flex ; height: 26px; margin-left:5px">

<input type="text" id="userName" name="name" value="@User.Identity.Name" style="display:none; height:1px" /><br />

<input type="number" id="rate" name="rate" style="display:none " /><br />

<input type="number" id="modelId" name="id" value="@Model.Id" style="display:none; height:1px" /><br />

<input class="btn-dark" type="submit" value="Rate" style="border:1px; border-radius:5px; width:100px"/>

</form>

</div>

<div style="display:flex; justify-content:space-between; width:100%; margin-top:20px">

<img src="/img/@Model.Id/1.jpg" class="img-thumbnail" style="max-height:200px;" alt="Alternate Text" />

<img src="/img/@Model.Id/2.jpg" class="img-thumbnail" style="max-height:200px;" alt="Alternate Text" />

<img src="/img/@Model.Id/3.png" class="img-thumbnail" style="max-height:200px;" alt="Alternate Text" />

</div>

@{

var userName = User.Identity.Name;

var a = userName;

}

</div>

</div>

<script>

var checked = false;

var tempimage = 0;

var tmp = 0;

function check( n)

{

console.log(n)

tmp = n;

checked = false;

for (let i = 0; i < n; i++) {

let k = $(".rate-star")[i];

$(".rate").find(k).attr('src', '/img/blackstar.svg')

}

checked = true;

}

document.onmouseover = function (event) {

if (event.target.className != "rate-star" && checked == true) {

for (let i = 0; i < tempimage; i++) {

let k = $(".rate-star")[i];

$(".rate").find(k).attr('src', '/img/blackstar.svg')

}

for (let n = tempimage; n < 10; n++ && checked == true) {

let k = $(".rate-star")[n];

$(".rate").find(k).attr('src', '/img/star.svg')

}

}

else {

for (let i = 0; i < tmp; i++ && checked == true) {

let k = $(".rate-star")[i];

$(".rate").find(k).attr('src', '/img/blackstar.svg')

}

for (let n = tmp; n < 10; n++ && checked == true) {

let k = $(".rate-star")[n];

$(".rate").find(k).attr('src', '/img/star.svg')

}

}

}

function rateCheck(n) {

checked = true;

tempimage = n;

let item = document.getElementById('rate')

for (let i = 0; i < n; i++) {

let k = $(".rate-star")[i];

$(".rate").find(k).attr('src', '/img/blackstar.svg')

}

item.value = n;

}

</script>

**Додаток 21**

Home>Index.cshtml

@model IEnumerable<Film>

@{

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

}

<style>

@@import url('https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,500;1,400;1,500&display=swap');

\* {

font-family: 'Roboto', sans-serif;

}

.film-text p{

margin-bottom:10px;

}

.top :hover {

border-radius:5px;

background: #cecece;

font-weight: bold;

transition: all 0.3s ease 0s;

}

</style>

<div id="carouselExampleIndicators" class="carousel slide" data-ride="carousel">

<ol class="carousel-indicators">

<li data-target="#carouselExampleIndicators" data-slide-to="0" class="active"></li>

<li data-target="#carouselExampleIndicators" data-slide-to="1"></li>

<li data-target="#carouselExampleIndicators" data-slide-to="2"></li>

<li data-target="#carouselExampleIndicators" data-slide-to="3"></li>

</ol>

<div class="carousel-inner">

<div class="carousel-item active">

<img class="d-block w-100" src="/img/1.png" alt="First slide">

</div>

<div class="carousel-item">

<img class="d-block w-100" src="/img/2.jpg" alt="Second slide">

</div>

<div class="carousel-item">

<img class="d-block w-100" src="/img/3.jpg" alt="Third slide">

</div>

<div class="carousel-item">

<img class="d-block w-100" src="/img/4.jpg" alt="Third slide">

</div>

</div>

<a class="carousel-control-prev" href="#carouselExampleIndicators" role="button" data-slide="prev">

<span class="carousel-control-prev-icon" aria-hidden="true"></span>

<span class="sr-only">Previous</span>

</a>

<a class="carousel-control-next" href="#carouselExampleIndicators" role="button" data-slide="next">

<span class="carousel-control-next-icon" aria-hidden="true"></span>

<span class="sr-only">Next</span>

</a>

</div>

<div class="container" style="margin-top:10px;">

<div class="row" style="display:flex">

<div class="col-sm-8 ">

<div class="row">

@foreach (var item in Model)

{

int rating;

if (item.UserRate.Count == 0) { rating = 0; }

else{ rating = (int)Math.Round((item.UserRate?.Select(x => x.Rate).Average() ?? 0) \* 10); }

<div class="col-sm-12 border rounded" style="margin:10px 0px 10px 0px; ">

<div class="film" style="display:flex;">

<img src="@item.image" alt="Alternate Text" style="height:250px; margin:10px 10px 10px 0px" />

<div class="film-context " style="width:100%; margin:10px 10px 10px 0px">

<div class="rating" style="display:flex; justify-content:space-between;width:100% ">

<h3> <a **asp-controller**="Film" **asp-route-Id**="@item.Id">@item.Name</a></h3>

<div>

<label><img src="/img/metacritic-icon.svg" alt="Alternate Text" style="width:20px;" /> @item.Metacritics</label>

<label><img src="/img/rotten tomato.png" alt="Alternate Text" style="width:20px;" /> @item.RottenTomatoes</label>

<label><img src="~/img/tomatouser.png" alt="Alternate Text" style="width:20px;" /> @rating</label>

</div>

</div>

<div class="film-text">

<p class="text-secondary">@item.ReleaseDate, Rating: @item.Rating</p>

<p>Director: @item.Director</p>

<p>Genres: @item.Genres , @item.Runtime min </p>

<p>Summary: @item.Summary</p>

</div>

</div>

</div>

</div>

}

</div>

</div>

<div class="col-sm-4 " style="margin:10px 0px 10px 0px; ">

<h3>Top 5 by Metacritic </h3>

@foreach (var obj in Model.OrderByDescending(s => s.Metacritics).Take(5))

{

<div class=" top">

<div class="col-sm-12" style="margin:10px 0px 15px 0px; ">

<div class="sider-bar-info" style="display:flex; justify-content:space-between">

<a **asp-controller**="Admin" **asp-route-Id**="@obj.Id" **asp-action**="Edit">@obj.Name</a>

<label><img src="/img/metacritic-icon.svg" alt="Alternate Text" style="width:20px;" /> @obj.Metacritics</label>

</div>

</div>

</div>

}

</div>

</div>

</div>