

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ЗАКЛАД
«ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА»

Навчально-науковий інститут фізики, математики
та інформаційних технологій

Кафедра фізико-технічних систем та інформатики

Плотніков Євген Євгенович

**ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ РОЗРОБКИ ЕЛЕКТРОННИХ
ПОШУКОВИХ СЕРВІСІВ.**

кваліфікаційна робота
здобувача вищої освіти другого (магістерського) рівня
освітньої програми «Інформатика»
за спеціальністю 014.09 «Середня освіта. Інформатика»

Особистий підпис — _____

Науковий керівник — _____ **Козуб Ю.Г., д.т.н., доцент**
(підпис) (посада, науковий ступінь, наукове звання, ініціали, прізвище)

Зав. кафедри — _____ **Козуб Ю.Г., д.т.н., доцент**
(підпис) (посада, науковий ступінь, наукове звання, ініціали, прізвище)

РЕФЕРАТ

Пояснювальна записка до магістерської роботи складається з трьох розділів, містить 73 сторінок, 36 рисунків, 30 джерел.

Об'єкт розробки: Дослідження технологій розробки електронних пошукових

Мета роботи: створення пошукового сервісу для ліцею який допоможе в роботі для вчителів які працюють в ліцеї

В магістерській роботі виконано:

1. Проаналізовані переваги і недоліки існуючих платформ пошукових сервісів.

2. Розроблений пошуковий сервіс по базі даних для ліцею.

Отримані наступні результати: пошуковий сервіс з можливістю додавати дані до бази даних.

Практичне значення, галузь застосування роботи: додаток може бути використаний для створення на базі гімназій і ліцеїв матеріалів з різних предметних галузей.

Ключові слова: ASP.NET, ENTITY FRAMEWORK, C#, MYSQL,

Зміст

ВСТУП.....	5
1 Аналіз методів організації пошуку інформації в пошукових системах.....	6
1.1 Загальні алгоритми пошуку інформації	6
1.2 Алгоритм пошуку PageRank	7
1.3 Панель PageRank у браузері та посилальні маніпуляції	9
1.4 Принципи отримання зворотних посилань	12
1.5 Альтернативні метрики авторитетності	13
1.6 Висновок про алгоритм PageRank.....	Ошиб
1.7. Аналіз сучасних інформаційно-пошукових систем	14
2 СТВОРЕННЯ ПОШУКОВО ІНФОРМАЦІЙНОГО РЕСУРСУ НАВЧАЛЬНОГО ПРИЗНАЧЕННЯ	20
2.1. Проблеми інтеграції інформаційних ресурсів	20
2.2. Принципи інтеграції інформаційних ресурсів навчального призначення....	22
2.3. Практичні аспекти програмно-технічної реалізації механізмів пошукової інтеграції	24
2.4 Висновок про створення пошуково інформаційного ресурсу навчального призначення.....	Ошиб
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПОШУКУ В БАЗІ ДАНИХ ЛІЦЕЮ.....	29
3.1 Створення рішення в Visual studio	29
3.2 Створення структури каталогів проєкта.....	31
3.3 Налаштування конфігурації веб-додатку	32
3.4 Зміна макету проєкту.....	33
3.5 Зміна заголовка, нижнього колонтитулу та посилання меню у файлі макета	34
3.6 Передача даних з контролера на подання	35
3.7 Створення класу моделі контролера.....	37
3.8 Формування шаблону сторінки Баз даних.....	37
3.9 Огляд створеного класу контексту бази даних.....	40
3.10 Метод Details	42

3.11 Методи та подання контролера в ASP.NET Core	45
3.12 Метод Edit.....	47
3.13 Пошук в базі даних	50

ВСТУП

За століття, що минули з початку того як з'явилися книги, людство придумало декілька основних способів пошуку інформації в книгах.

І кожен із нас зустрічався з ними ще до того як з'явився доступ до мережі Інтернет. Це пошук за змістом, посиланнями та предметним покажчиком.

З'ясувалось що для пошуку потрібної інформації в Інтернеті так на різних сторінках так само використовуються ті ж самі методи , тільки вони виконуються автоматично або за допомогою спеціальних програм.

Взагалі, розробка будь-якого програмного продукту потребує великих затрат: як часу, так і ресурсів (людських та матеріальних). Тому під час розробки чогось нового, потрібно врахувати усі ризики.

Була обрана тема «Дослідження технологій розробки електронних пошукових», оскільки зараз людство все більше використовує мережу Інтернет для пошуку інформації. Тому розробникам потрібно кожного разу модернізувати свої системи для більшого і кращого пошуку інформації в мережі інтернет . Багато сайтів та компанії мають свої пошукові системи деякі з них користуються великою популярністю.

Метою даної роботи було створити пошуковий додаток для ліцею який допоможе вчителям завантажувати свої розробки уроків, а учням переглядати ці уроки навіть якщо вони вже давно вивчали якусь тему. У програмі повинна бути реєстрація користувача та можливість доступу до інформації для усіх учнів та вчителів які будуть зареєстровані в системі.

1 Аналіз методів організації пошуку інформації в пошукових системах

1.1 Загальні алгоритми пошуку інформації

На даний час відомі такі пошукові системи, як Google, Yahoo . Bing і багато інших. Алгоритми кожної системи є унікальні і вони так само важливі, як і ключові слова які в них задає користувач для пошуку.

Алгоритм пошукової системи – формула , завдяки якій система пошуку використовує для виведення даних також кожна система має свої правила [1]. Кожне з них правил визначає, чи є документ важливим для користувача, чи містить він ту інформацію, яка потрібна для користувача, а також багато інших функцій для відображення списку результатів-пошуку кожного запиту пошуку інформації.

Алгоритми пошуку відрізняються у кожної пошукової системи та мають свою класифікацію але є свої критерії, які загальні для усіх алгоритмів пошукових систем. Перший – це релевантність.

Релевантність – це алгоритм пошукової системи на потрібн інформацію яку шукає користувач системи. Це може бути сканування по ключовим словам або де використовуються ключові слова які були задані користувачем . Розташування слів є важливої частиною для забезпечення релевантності пошукої систесми . Документи з ключовими словами в документі , або рядках які можуть містися в тексті і вони будуть ранжуватися за цими ключовими словами , ніж документи, які не мають відповідний ключовиїх слів.

Кількість ключових слів також важлива для релевантності. Наприклад якщо є ключові слова часто відображаються , але вони не є результатом наповнення, документ може мати більший рейтинг[2]. Ще одним критерієм пошуку є індивідуальні факти . Друга частина за якими працюють алгоритми пошукової системи – це фактори, які відрізняють пошукову систему від інших пошукових систем . У кожної пошукової системи свої унікальні алгоритми, і

окремі фактори цих алгоритмів пояснюють, чому один і той самий запит в Google, Yahoo і Bing показує різні результати. Одним з найбільш факторів є кількість сторінок, індексованих пошуковою системою.

Кожна пошукова система може індексувати більше або частіше і відповідно від цього відображати різні пошукові результати. Деякі пошукові системи карають за спам а інші системи ігнорують це. [3] Є ще фактори які так само входять до алгоритмів пошукових система вони також індивідуальні для кожної пошукової системи. Фактор положення сторінки частота кліків і та посилання може бути показником того, наскільки документ корисний для реальних користувачів і відвідувачів, воно може призвести до того, що алгоритми різних пошукових систем можуть підняти рейтинг документу.

1.2 Алгоритм пошуку PageRank

PageRank — алгоритм ранжування, який оцінює кількість та якість посилань, що ведуть на веб-сторінки. Інженери Google Ларрі Пейдж та Сергій Брін розробили цей алгоритм у 1998 році — і це був прорив для функціонування пошуку, адже вперше пошукова система оцінювала авторитетність сторінок.

Самі інженери пояснювали, що PageRank має на меті «упорядкувати веб-простір» завдяки розстановці сил між сторінками. Алгоритм будувався з урахуванням умовного «рандомного інтернет-користувача», який переходить з однієї сторінки на іншу, клацаючи посилання. PageRank сторінки - ймовірність, з якою цей інтернет-користувач перейде на неї. Оцінка розраховувалася в межах від 0 до 10: що вона вище, то вищий авторитет сторінки.

PageRank — це спроба об'єктивно оцінювати сторінки відповідно до суб'єктивної поведінки користувачів: це природно, що чим частіше на сторінку посилаються, тим більше вона корисна для людей, які шукають інформацію.

Алгоритм також враховує авторитет джерел, що посилаються: чим вище оцінка PageRank у певної сторінки, тим більше ваги вона передасть іншій

сторінці, на яку посилається.

Давайте розглянемо, як розраховується оцінка PageRank.(Рис 1.)

$$PR(A) = (1-d) + d (PR(T1)/C(T1) + \dots + PR(Tn)/C(Tn))$$

Рис. 1 Формула PageRank

Так виглядає оригінальна формула PageRank де змінні мають такі значення:

A - аналізована сторінка

T1...Tn — сторінки, що посилаються на аналізовану

C – кількість посилань на аналізованій сторінці

d — коефіцієнт загасання, що означає ймовірність того, що користувач залишить сторінку

Виходячи з цієї формули, сторінки «роздають» свою оцінку PageRank іншим сторінкам, на які посилаються. Наприклад, джерело A з оцінкою 5 цитує B і C. Не зважаючи на інші посилання, які можуть мати B і C, ці сторінки отримують 85% від оцінки A (оцінка, помножена на демпінг-фактор), тобто 4.25 разом і по 2.125 кожна. Якщо B цитує D, оцінка D становитиме 85% від 2.125, тощо.(Рис 2.)

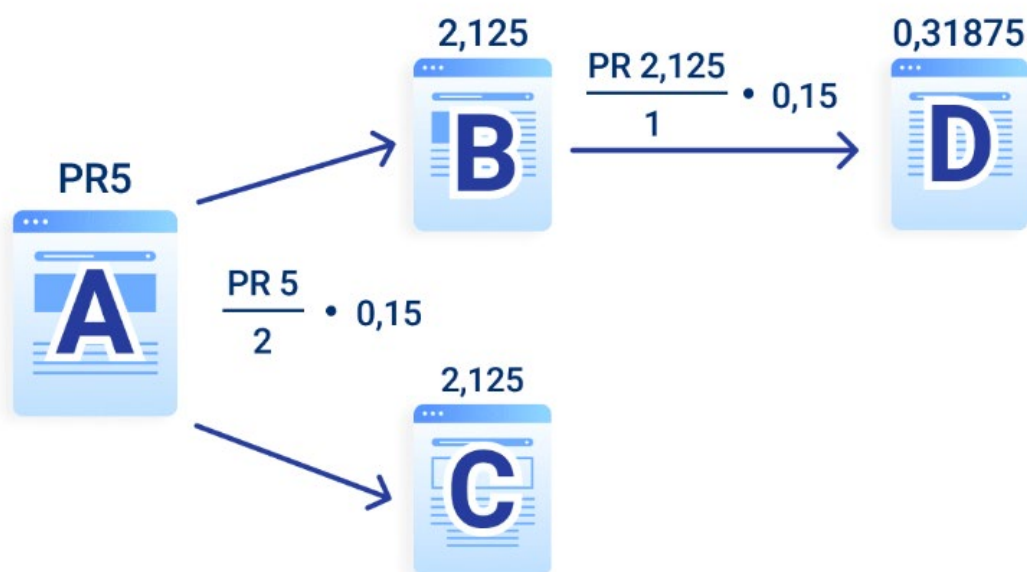


Рис. 2 Приклад розрахунку по формулі PageRank

1.3 Панель PageRank у браузері та посилальні маніпуляції

У 2000 році Google додав до браузера спеціальну панель з PageRank і можна було дізнатися оцінку будь-якого сайту. Публічний доступ до оцінки привів до маніпуляцій та штучного моделювання PageRank. Сайти намагалися отримати більше посилань зі сторінок з високою оцінкою і попит на купівлю посилань зріс аж до створення ферм посилань.

Таке розуміння алгоритму PageRank не було далекоглядним SEO-підходом, адже в отриманні беклінків важлива не тільки їх кількість і умовна якість ресурсу, що посилається, а й контекст, релевантність і природність.

Google намагався виявляти та зупиняти маніпуляції з PageRank та закрив публічно доступну панель у 2016 році. Досі існують сервіси, що вираховують оцінку PageRank і пропонують сайтам встановити бейдж із цією оцінкою — але це вже не актуально. Алгоритм досі бере участь у ранжируванні сторінок, але дізнатися реальну оцінку своєї чи зовнішньої сторінки неможливо.

Маніпулятивні практики отримання посилань були пов'язані не лише з доступністю оцінок PageRank. Спамні посилання в коментарях працювали на просування сайту — і пошукові системи розуміли, що якось це треба припинити. У 2005 році Google та інші пошукові системи ввели значення `nofollow` для атрибуту `rel`. Він каже пошуковим ботам не слідувати за посиланням, а значить і не передавати вагу посилань.

На цьому маніпулятивні техніки не вичерпали себе. Сайти почали використовувати `nofollow`, щоб на тлі посилань з цим значенням решта посилань отримувала більше ваги.

Розглянемо з прикладу. Джерело з оцінкою PageRank 5 цитує 10 сторінок, 8 із них — з `nofollow`. До впровадження цього значення кожна зі сторінок отримала $\frac{6}{10}$ оцінки (0.425 з урахуванням демпінг-фактору). А з `nofollow` лише 2 сторінки (не позначені цим значенням в атрибуті `rel`) отримують по половині оцінки (2.125 з урахуванням демпінг-фактора).

Ця схема перестала працювати в 2009 році, коли Google виправив механізм PageRank, щоб посилання з nofollow не передавали свою частину оцінки. Якщо взяти описаний вище приклад, сьогодні ті самі 2 сторінки отримують не 2.125, а 0.425, тобто PageRank розподіляється рівномірно на всі 10 посилань, але посилання з nofollow не передають цю вагу.

Навіть з nofollow посилання, що залишаються в коментарях, представляли проблему. У 2019 році Google додав нове значення атрибуту спеціально для таких посилань - UGC (user-generated content), що буквально означає "контент, створений користувачами". Зараз багато блогів та форумів автоматично маркують будь-які посилання в коментарях як UGC, а nofollow використовується для свідомо проставлених зовнішніх посилань, про авторитетність яких не хочеться сигналізувати пошуковим системам.

Новий алгоритм page PageRank був зроблений у 2004 році. В основі якого лежала модель «раціонального інтернет-користувача» та ідея про те, що потенціал переходу за посиланням впливає на якість цього посилання. Наприклад, посилання, розміщені в першій частині контенту сторінки, або посилання зі зрозумілими та інформативними анкорними текстами зазвичай більш помітні для користувачів. Тому можливість переходу за посиланням теж враховується в оцінці авторитетності сторінок.

Крім того, в 2006 році Google розробив нову систему оцінювання авторитетності: визначається кілька ресурсів з найвищим рівнем довіри (seed pages, буквально «сторінки-зерна») та з їх урахуванням оцінюються решта всіх сторінок у мережі

$$\forall s_i \neq p \in P, R_i(p) = d \sum_{q \rightarrow p} R_i(q) / q_{out} * w(q \rightarrow p)$$

Рис. 3 Формула PageRank

Нова формула PageRank(Рис 3.) виглядає так

Що означають ці змінні:

s_i — найавторитетніші сторінки (seed pages)

P - всі існуючі веб-сторінки

qout — напівступінь результату сторінки q

w - вага посилання (стандартно дорівнює 1)

Приклад авторитетної сторінки - The New York Times: цей сайт розміщує матеріали на різні теми, які цікавлять користувачів, і включає корисні вихідні посилання. Тому сторінки, на які такий ресурс посилається, також вважаються якісними.

Відповідно до оновленого алгоритму, розподіл позицій у пошуку на основі посилань відбувається у кілька етапів:

- система отримує певну кількість сторінок, відкритих до індексування
- система свідомо знає про сторінки з найвищим рівнем довіри
- система розраховує, наскільки далеко від найавторитетніших знаходяться аналізовані сторінки (за вихідними посиланнями)
- система визначає позиції в пошуку, ставлячи вище ті сторінки, які на кліках знаходяться ближче до найавторитетніших

Нова формула працює швидше, тому що не оцінює всі взаємозв'язки посилань у сукупності. Хоча оригінальний патент PageRank і втратив свою силу у 2018 році, алгоритм досі використовується Google для ранжування сайтів. У Twitter аналітик Google Джон Мюллер вказав на те, що вони використовують PageRank «серед багатьох інших сигналів».

На алгоритм PageRank впливає декілька факторів:

- кількість посилань
- атрибути посилань
- анкорні тексти
- імовірність переходу за посиланням

Тепер давайте розберемося, як грамотно вибудовувати свій профіль, щоб сигналізувати пошуковим системам про високу якість і корисність сторінок сайту.

1.4 Принципи отримання зворотних посилань

Хоча алгоритм PageRank видозмінювався, його суть залишилася незмінною з 1998 року: беклінки працюють як голоси на користь сторінок і допомагають пошукачам визначати авторитетність. Але зовнішні посилання, які ведуть певну сторінку, додають їй ваги за певних умов:

- якщо вони релевантні. Релевантність важлива у більшості SEO-процесів. Пошуковим системам не сподобається, якщо посилання між сторінками не сигналізуватимуть і семантичного зв'язку між ними. Скажімо, якщо сторінка з кулінарними рецептами отримає беклінк зі сторінки про автомобілі, таке посилання не принесе користі, навіть якщо ресурс, що посилається, дуже авторитетний
- якщо вони супроводжуються органічним анкорним текстом. Переспамлені ключовими словами анкорні тексти не принесуть користі, а малоінформативні (начебто тут) будуть менш корисні, ніж розгорнуті. Анкори повинні зрозуміло натякати на інформацію, що знаходиться на засланні.
- якщо сайти, що посилаються, досить авторитетні. Щоб отримувати корисні беклінки, потрібно перевіряти якість домену, що посилається, і конкретної сторінки, а також регулярно моніторити, чи не отримує сайт спамних посилань.
- якщо посилання доступні для пошукових роботів. Щоб передавати вагу посилань, посилання повинні бути доступні для краулерів і не закриті за допомогою robots.txt або іншого методу.
- якщо вони не ведуть на сторінки помилки сервера. Обидві сторінки — посилання та та, на яку веде посилання, повинні мати 200 код відповіді сервера, тобто бути відкритими до індексації. У випадку редиректів (3** коди) не всі з них передають ваги: хоч Google і стверджує, що всі редиректи передають PageRank, відмінні від 301 коди можуть не справлятися з цим завданням.
- якщо вони не перешкоджають пошуковим роботам слідувати за

посиланням. Як ми вже сказали, значення nofollow впливає на розподіл ваги, і беклінк з nofollow менш значущий, ніж з follow.

- якщо посилання видно на сторінці. Сховані посилання можуть призвести до штрафів пошукових систем — їх потрібно виділяти кольором або підкресленням, щоб вони були помітними, але не виділялися із загального візуального стилю сторінки.

PageRank оцінює авторитетність окремих сторінок, а не цілих сайтів, тому внутрішні посилання також важливі для оцінки, як і беклінки. За допомогою перелінкування ви можете спрямовувати вагу на сторінки сайту:

- Чим більше внутрішніх посилань ведуть на сторінку, тим вища її PageRank

- Чим більше посилань розміщено на сторінці, тим нижче PageRank, який вона передає

- Легко доступні та клікабельні посилання передають вищий PageRank

- Посилання з nofollow не передають PageRank

На відміну від внутрішніх зовнішні посилання не впливають на сторінки, на яких вони розміщені. Вони допомагають пошуковикам встановлювати зв'язки між тематично близькими ресурсами, але безпосередньо не впливають на вагу і позиції в пошуку.

1.5 Альтернативні метрики авторитетності

PageRank - перша метрика авторитетності, яка багато в чому змінила підхід до SEO. Ця оцінка досі є актуальною для Google, хоч ми й не знаємо, як саме вона впливає на позиції. Що можна сказати з точністю, те, що релевантні посилання з якісних джерел точно корисні і для нарощування авторитету, і для просування в органічному пошуку.

Інші SEO-метрики, створені для оцінки авторитетності, так само спираються на кількість і якість зворотних посилань

Метрики авторитетності, створені SEO-платформами, ґрунтуються на

посилальному профілі. SE Ranking має свої оцінки — Domain Trust і Page Trust, які оцінюють якість домену або сторінки, виходячи з кількості і якості беклінків і доменів. В інструменті «Аналіз конкурентів» ви можете переглянути оцінку Domain Trust та Page Trust для будь-якого сайту:

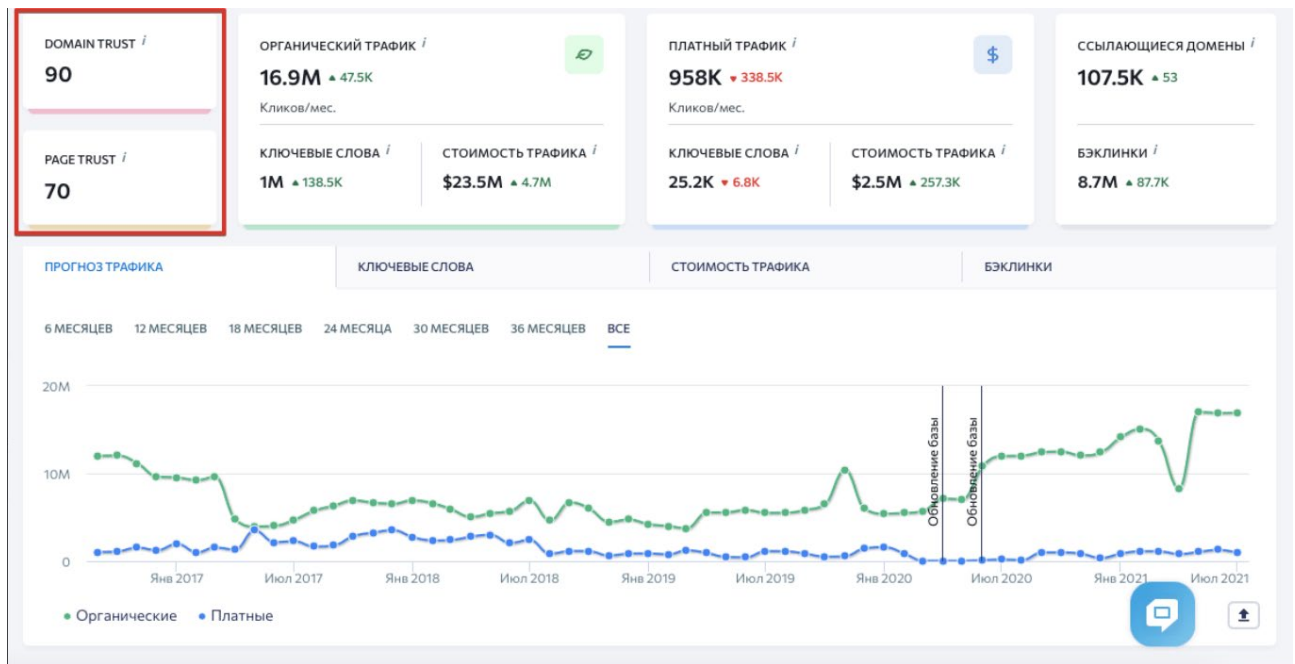


Рис 4. Інструменті Аналіз конкурентів

Ці дані також доступні в інструментах "Аналіз беклінків" та "Моніторинг беклінків", і Domain Trust окремо відображається в оглядовій частині "Аналізу сайту" поряд з іншими доменними метриками. [4]

1.7. Аналіз сучасних інформаційно-пошукових систем

Google є потужним інструментом, без якого неможливо знайти інформацію при перегляді вебсторінок. Google використовує спеціальний алгоритм для генерації результатів пошуку. Google ділиться лише загальними фактами про свій алгоритм. Це допомагає Google залишатися дуже спроможною пошуковою системою [5]

Google використовує автоматичні програми, які називаються павуки або сканерии. Так само як і інші системи пошуку, Google має індекс ключових слів.

що відрізняє Google від інших систем пошуку, як він ранжує результати пошуку, що, визначає чергу та порядок, в якому Google показує результати на сторінці пошукової системи Results Page. Google використовує алгоритм PageRank, який присвоює кожній веб-сторінці оцінку релевантності. [6]

Розглянемо на прикладі пошуку за терміном «інформатика». У міру того, більша кількість веб-сторінок посиляються на сторінку Інформатика в Вікіпедії, рейтинг сторінки Вікіпедія збільшується. Коли сторінка Вікіпедія займає перше місце, ніж інші сторінки вона відобрається одразу вгорі сторінки результатів пошуку Google. Оскільки Google розглядає на веб-сторінку як голосування, обманути систему важко, але можливість така є. [7]

Компанія Google розпочала експеримент пошуком у 2008 році. Вперше дозволивши групі тестерів почати змінювати порядок ранжування результатів пошуку. В цьому експерименті тестери можуть рекламувати або знижувати результати і адаптувати досвід пошуку так, щоб він був актуальним [8]. Розглянемо, наприкладі як Google визначає, що потрібно показувати в результатах пошуку. Алгоритм починає роботу ще до того, як користувач вводить ключові слова у поле вводу.

Спочатку відбувається сканування та індексація. Google використовує веб-сканерів та Павуків для організації інформації з вебсторінок та іншого загальнодоступного контенту в пошуковому індексі. Після цього починають працюювати алгоритми пошуку.

Система рейтингу Google сортує мільярди веб-сторінок у пошуковій індексації, щоб за секунди вивести корисні і релевантні результати. Наступним етапом є алгоритми пошуку

Системи ранжування складаються з ряду алгоритмів, які аналізують те, що шукає користувач, і яку інформацію потрібно повернути. Схема процесу ранжування пошуку інформації, якими користується Google для надання корисної інформації з Інтернету зображена на рис 5



Рис. 5 Схеми процесу пошуку та отримання інформації в пошуковій системі Google

Аналіз запиту, розуміння значення пошуку має вирішальне значення для отримання точних результатів пошуку. Тому, щоб знайти сторінки з відповідною інформацією, потрібно спочатку проаналізувати, що означають слова у пошуковому запиті. Будуються мовні моделі, щоб спробувати розшифрувати, які рядки слів потрібно шукати в індексі. Це 15 включає в себе такі кроки, як інтерпретація орфографічних помилок, ідентифікація типу запиту, застосовуючи деякі з останніх досліджень в області розуміння природної мови.

Наприклад, система синонімів допомагає пошуку зрозуміти запит, навіть якщо слово має кілька значень. 16 Пошук веб-сторінок, забезпечує пошук веб-сторінки з інформацією, що відповідає запиту. Коли виконується пошук на базовому рівні, алгоритм шукає пошукові терміни в індексі, щоб знайти відповідні сторінки. Алгоритм аналізує, як часто і де ці ключові слова з'являлися на сторінці, в заголовках або в тексті. Крім зіставлення ключових слів, алгоритм шукає докази, щоб виміряти, наскільки добрі потенційні результати пошуку дають користувачам те, що вони шукають. Коли робиться пошук, наприклад, виразу «оперативна пам'ять», користувач, ймовірно, не хоче, щоб сторінка зі словом «пам'ять» з'явилась у нього сотні разів. Алгоритм намагається з'ясувати, чи містить сторінка відповідь на запит, а не просто повторити цей запит. Таким чином, алгоритм пошуку аналізує, чи містять сторінки релевантний контент, наприклад, зображення літаків, відео або список архітекторів. Нарешті, алгоритм перевіряє, чи написана сторінка на тій же мові, що і запитання, щоб визначити пріоритетність сторінок на вибраній мові.

Ранжування корисних сторінок. Для типового запиту існують тисячі, навіть мільйони веб-сторінок з потенційно актуальною інформацією. Щоб у першу чергу оцінити кращі сторінки, використовується алгоритм для оцінки корисності цих веб-сторінок. Цей алгоритм аналізує сотні різних факторів, щоб спробувати розкрити найкращу інформацію, яку може запропонувати пошук, від свіжості контенту до частоти появи пошукових термінів, а також від того,

наскільки зручна сторінка. Щоб оцінити достовірність і авторитет у своїй тематиці, алгоритм шукає сайти, оцінює дані користувачів із схожими запитами. Якщо інші сайти за темою посилаються на сторінку, це ознака того, що інформація високої якості. В Інтернеті є багато сайтів зі спамом, які намагаються пробитися до вершини результатів пошуку за допомогою таких методів, як повторення ключових слів знову і знову або через посилання, що проходять через PageRank. Ці сайти надають дуже поганий користувацький досвід і можуть навіть зашкодити або ввести в оману користувачів [9]

Огляд контексту, такої інформації, як місце розташування користувача, історія пошуку в минулому і налаштування пошуку, допомагають алгоритму адаптувати результати так, щоб інформація була найбільш корисно і актуально для користувача в даний момент. Алгоритм використовує дані про країну і місце розташування користувача для доставки контенту відповідно його регіону. Наприклад, якщо користувач перебуває в Чикаго і шукає «матч», швидше за все, він отримає результати з американського футболу і «матчів Чикаго». Якщо шукати «матч» в Лондоні, алгоритм оцінить результати про футбол. У деяких випадках алгоритм також може персоналізувати результати, використовуючи інформацію про недавній пошуковий запит користувача. Наприклад, якщо користувач шукає «Барселона» і недавно шукав «Барселона проти Арсеналу», це може бути важливою підказкою, що йому потрібна інформація про футбольний клуб, а не про місто. Отримання кращих результатів пошуку, перш ніж алгоритм представить результати. Алгоритм оцінює, як вся відповідна інформація поєднується: чи є одна тема серед результатів пошуку чи їх багато, а також занадто багато сторінок, присвячених одній вузькій інтерпретації. Алгоритм прагне надати різноманітний набір інформації в форматах, які найбільш корисні для конкретного типу пошуку.

Завдяки більшій кількості контенту і більшого розмаїття в Інтернеті, ніж будь-коли раніше, Google має можливість пропонувати результати пошуку в широкому діапазоні форматів, щоб допомогти користувачу швидко знайти потрібну інформацію, але нерідко це важко зробити через дуже великий

діапазон та час на пошук релевантної інформації Мережа Інтернет постійно розвивається, кожену секунду публікуються сотні нових веб-сторінок. [10]

Це відображено в результатах пошуку, Google постійно аналізує інформацію у мережі для індексації нового контенту. Останнім кроком алгоритму є корисні відповіді. Залежно від запиту користувача деякі сторінки результатів швидко змінюються, а інші стають більш стабільними. Наприклад, коли користувач шукає останній рахунок спортивної гри, потрібно виконувати найсвіжіші оновлення, в той час як результати про історичну тематику можуть залишатися незмінними роками. Сьогодні Google обробляє трильйони пошукових запитів щороку. Кожен день 15% запитів, які обробляються, це ті, які ми ніколи раніше не бачили. Побудова алгоритмів пошуку, які можуть надати найбільш корисні результати для всіх цих запитів, є складним завданням, яке вимагає постійного тестування якості та інвестицій. Багато інженерів і вчених працюють над вдосконаленням алгоритмів і пошуком нових корисних способів пошуку інформації [11].

Тільки у 2016 році було покращено близько 1600 рішень у пошуку Google, який використовує наступні алгоритми пошуку інформації.

1. Відповіді з графа знань.
2. Напрямки за допомогою трафіку.
3. Прямі відповіді.
4. Вибрані фрагменти.
5. Багаті списки.
6. Відповіді, перш ніж запитати [12]

2 СТВОРЕННЯ ПОШУКОВО ІНФОРМАЦІЙНОГО РЕСУРСУ НАВЧАЛЬНОГО ПРИЗНАЧЕННЯ

2.1. Проблеми інтеграції інформаційних ресурсів

Останнім часом стає очевидним, що компетентному спеціалісту необхідно постійно працювати з інформацією, яка містить нові відомості, причому не тільки у сфері його діяльності, але й у суміжних галузях, оскільки кожен розділ педагогіки не тільки тісно пов'язаний з іншими галузями педагогічного знання, але й базується на відповідних розділах психології. Усвідомлення фундаментальної ролі інформації у суспільному розвитку, величезні обсяги наявної інформації, уявна легкодоступність необхідних матеріалів, з одного боку – полегшують завдання отримання необхідної інформації, але з іншого – призводять до того, що вона не може ефективно використовуватися у вирішенні деяких проблем. Тому сьогодні гостро стоїть завдання пошуку методів підвищення ефективності наукової, педагогічної та науково-педагогічної діяльності, які базуються, у тому числі, й на розробці прийомів пошукової інтеграції інформаційних ресурсів навчального призначення.

Проблема вирішення пошукової інтеграції не в останню чергу залежить від правильного розуміння терміну інтеграція в цілому. Отже, поняття інтеграція (лат. *integratio* – з'єднання), розглядається як фактор розвитку, явище, процес, що характеризується багатоваріантністю рішень. Філософський словник трактує інтеграцію як складову процесу розвитку, що пов'язана з об'єднанням в єдине ціле раніше різнорідних частин та елементів. При цьому процеси інтеграції можуть відтворюватися не тільки в рамках вже сформованих систем (у цьому випадку вони сприяють підвищенню рівня їх цілісності й організованості), але й виникати в абсолютно нових системах, користуючись

при цьому принципами взаємодоповнюваності їх функціонування і розвитку внутрішньосистемних зв'язків [13]

Є необхідність розглянути поняття інтеграції стосовно до педагогічних процесів. З цією метою ознайомимося з поглядом деяких вчених-педагогів на цей процес [14]

- Інтеграція – система органічно пов'язаних навчальних дисциплін, побудована за аналогією з навколишнім світом, в основу інтеграції покладена аксіома, що все в світі взаємопов'язане і не існує в «чистому вигляді» (О. Г. Гилязова);

- Інтеграція – процес зближення і зв'язку наук, що відбувається поряд з процесами диференціації та являє собою високу форму втілення міжпредметних зв'язків на якісно новому щаблі навчання (Н. С. Сердюкова); - Інтеграція як педагогічна категорія являє собою цілеспрямоване об'єднання, синтез певних навчальних дисциплін у самостійну систему 15 цільового призначення, спрямовану на забезпечення цілісності знань і умінь.

Узагальнюючи ці визначення, можна зробити висновок про те, що інтеграція – це процес, який дозволяє розкрити закономірності в педагогічних явищах і може використовуватися як засіб побудови педагогічних моделей, що дозволяє виявляти, вводити і конструювати ієрархічні зв'язки між елементами педагогічних систем.[18]

Саме таке визначення вважаємо доцільним для користування і створення моделі пошукової інтеграції інформаційних ресурсів навчального призначення. Питання розробки форм і методів інформаційного забезпечення освіти постійно досліджується науковцями. Але з появою і розвитком інтернету значно збільшилася різноманітність форм надання матеріалів педагогічної спрямованості. З'явилися бази даних не тільки з текстовою, графічною та мультимедійною інформацією, але й величезні обсяги інформації на порталах і сайтах. Але тут виникла проблема гетерогенності (різноманітності) існуючої

інформації, яка ще й зберігається в різноманітних за своєю структурою базах даних зі своїми системами управління. Забезпечення оперативності доступу до такої інформації має певні труднощі, пошук і швидке отримання результату є суттєвою проблемою.

Розростання неузгоджених структур даних призвело до виникнення цілого ряду технічних і організаційних проблем, де рішенням може стати пошукова інтеграція інформаційних ресурсів. Отже, можна виокремити ті недоліки, які потрібно вирішувати шляхом інтеграції пошукових ресурсів навчального призначення:

- Відсутність загальноприйнятого однакового формату зберігання документів у різних база даних; - Відсутність однакових способів доступу до матеріалів педагогічного призначення;
- Наявність власних специфічних пошукових атрибутів для кожної колекції;
- Відсутність єдиної тематичної систематизації/класифікації документів за рубрикаторм;
- Відсутність одноманітно структурованої довідкової, реферативної та оглядової інформації, що характеризує коло предметів і явищ освітньої сфери діяльності;
- Відсутність діючих предметно-орієнтованих баз даних, інтегрованих в загальну систему.

2.2. Принципи інтеграції інформаційних ресурсів навчального призначення

Життєдіяльність сучасних наукових, педагогічних і науковопедагогічних кадрів відбувається у величезному інформаційному просторі, яке потребує постійного підвищення рівня знань та кваліфікації.

Одне з основних місць в цьому просторі відводиться електронним освітнім ресурсам, яким, притаманні системоутворюючі ознаки: «наближеність до реального життя; доступність; здатність впливати на розвиток мислення, діяльність або на особистість в цілому» [14]

Зробимо спробу знайти певні підходи для вирішення цих проблем, адже задачі пошукової інтеграції щодо інформаційних баз даних навчального призначення надзвичайно різноманітні та багатоаспектні. Основні вимоги, що висуваються при цьому до пошукової інтеграції полягають у забезпеченні оперативного отримання, інтегрованої обробці та представлення результатів усіх доступних на сучасному рівні розвитку освіти даних.

Інтеграція інформаційних ресурсів навчального призначення досягається за допомогою:

- Створення та введення однаковості метаданих стосовно певної предметної сфери;
- Створення та застосування єдиної моделі опису інформаційного ресурсу;
- Створення і зберігання електронних копій текстових і графічних матеріалів у вигляді файлових систем;
- Створення єдиної тематичної систематизації / класифікації документів за рубрикатором.

Саме такі підходи оптимально забезпечують вирішення завдання пошукової інтеграції різноманітної і слабоструктурованої інформації за допомогою:

- Використання унікальних пошукових атрибутів для різних форм представлення інформації (текстової, графічної, мультимедіа)

- Інтеграції різнорідних інформаційних матеріалів як для всієї системи освіти так і для окремих її складових;

- Інтеграції освітніх ресурсів національного та міжнародного рівнів. До переваг використання пошукової інтеграції інформаційних ресурсів відносяться:

- Різке скорочення часових рамок пошуку-обробки даних пошукових запитів здійснюється набагато швидше;

- Підвищення рівня релевантності та пертинентності пошукових запитів;
- Можливість обробки великих (чи понад-великих) потоків інформації, яку необхідно відшукати;

- Можливість отримання різноманітної, за структурою, інформації;

- Широкий діапазон оброблених баз даних, і, як наслідок, багаторазове збільшення числа необхідних матеріалів.

2.3. Практичні аспекти програмно-технічної реалізації механізмів пошукової інтеграції

Проблеми інтеграції різнорідних інформаційних ресурсів, і як наслідок, проблеми пошукової інтеграції були розглянуті раніше в п. 2.1 і п. 2.2. Тому, одним з найбільш перспективних напрямків формування та розвитку сучасних методик пошукової інтеграції, на наш погляд, є прийоми, засновані в першу чергу на формування єдиного інформаційного простору, побудованого за принципом інтеграції всіх освітніх інформаційних ресурсів.

Такий єдиний інформаційний простір здатен забезпечити цілісність всіх даних предметної сфери і надати можливість використовувати їх в безлічі інформаційних систем. Наукова складова єдиного інформаційного простору при цьому, полягає у використанні методів структуризації інформації, яка

зберігається в різних базах даних. Практична реалізація заснована на інтеграції інформаційних систем між собою. Мета синхронізації даних – усунення розбіжностей і дублювання інформації, оскільки набір програмно-інформаційних ресурсів кожної окремо взятої бази даних може істотно відрізнятися, а завдання їх об'єднання – надто складне практично. Одне з основних питань при спробі інтеграції педагогічної інформації в єдиний простір – завдання вибору програмно-технічних засобів.

Проаналізувавши літературу з даного питання виділяємо три основні методи інтеграції інформаційних баз даних[15, 16 ,17] :

- Побудова загальної бази даних з урахуванням інтеграції всіх інформаційних ресурсів освітнього напрямку;
- Програмна реалізація обміну файлами, що містять узагальнені дані; -
Забезпечення віддаленого виклику процедур в рамках систем обміну повідомленнями для виконання дій або обміну даними. Єдиний інформаційний простір на основі інтеграції педагогічної інформації схематично може виглядати наступним чином:

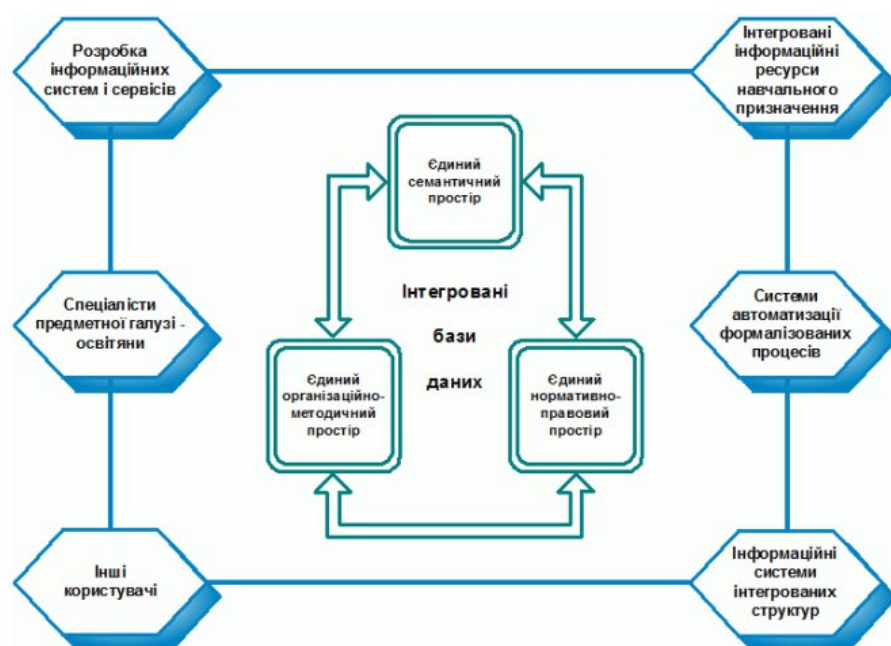


Рис.7. Єдиний інформаційний простір, створений на основі інтеграції педагогічної інформації

Базуючись на концепції побудови єдиного інформаційного простору ресурсів навчального призначення, розглянемо методи і принципи їх пошукової інтеграції. Перший крок – необхідність встановити принцип взаємодії користувача з системою інтегрованого пошуку:

- Користувач формує пошукової запит, намагаючись стисло описати характеристики (параметри) потрібних документів (ключові слова); -

Система здійснює пошук і вибірку документів, які відповідають тим чи іншим чином пошуковому образу документа і забезпечує користувачеві результат пошукового запиту;

- Користувач оцінює отриманий результат з точки зору релевантності та пертинентності.

Отже, системи інтегрованого пошуку повинні складатися з:

- засобів формування пошукового запиту;
- засобів передачі пошукового запиту пошуковому движку;
- засобів виконання пошуку; - засобів представлення результатів пошуку та роботи з ними у зручній формі.

Спираючись на принципи інтегрованого пошуку, можна говорити про створення уніфікованого інструменту для реалізації пошукових запитів, завдяки інтеграції в єдиний інформаційний простір матеріалів навчального призначення, що дозволяє абстрагуватися від проблеми розрізненості документів в базах даних.

Система інтегрованого пошуку в такому випадку, може бути реалізована у вигляді web-додатку, де чітко визначені дві частини – клієнтська і серверна, які складаються в свою чергу з функціональних модулів. Клієнтська частина надає користувачеві засоби формування пошукового запиту і перегляду результатів пошуку (роботи з ними).

Вона складається з наступних модулів:

Модуль K1 – Засоби формування пошукового запиту. Містить елементи системи генерації інтерфейсу користувача, призначеного для редагування опису пошукового образу документа.

Модуль K2 – Засоби передачі пошукового запиту користувача до серверної частини системи. Являє собою програмний шар між елементами логіки клієнтської частини системи і API-інтерфейсами, наданими серверною частиною системи; призначені для інкапсуляції формату передачі даних (пошукового запиту, результатів пошуку, параметрів функціонування і т. і.) між клієнтської і серверної частиною системи.

Модуль K3 – Засоби роботи з результатами пошуку. Являють собою набір елементів, призначених для забезпечення можливості перегляду і роботи з документами-результатами пошуку. Серверна частина реалізує засоби обробки (виконання) користувальницького пошукового запиту. Складається з наступних модулів:

Модуль C1 – здатен забезпечити роботу з документами, для яких проводиться пошук. Забезпечує перевірку прав доступу до певних колекцій, пошук документів, завантаження метаданих документів і т. і.

Модуль C2 – спрямований на роботу з колекціями, за якими проводиться пошук документів.

Модуль C3 – містить у собі пошуковий образ документа, а так само деякі параметри, що стосуються сортування результатів пошуку, їх максимальної кількості, тип відповідності документа пошуковому образу і т. і.

Модуль C4 – інкапсулює логіку пошуку, здійснює пошук за вказаним типом документів і типом пошукового запиту.

Модуль С5 – Являє собою прошарок між підсистемою пошуку і базою даних.

Модуль С6 – відповідає за доступ користувача до тієї чи іншої інформації (колекціям, документам).

Модуль С7 – містить такі параметри, як ліміт документів-результатів пошуку за замовчуванням, ступінь відповідності за замовчуванням, сортування за замовчуванням тощо.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПОШУКУ В БАЗІ ДАННИХ ЛІЦЕЮ

3.1 Створення рішення в Visual studio

Для створення ASP.Net Core MVC в IDE Visual Studio потрібно вибрати пункт меню Файл => Створити. На Рисунку 8 відображає контекстне меню для створення рішення.

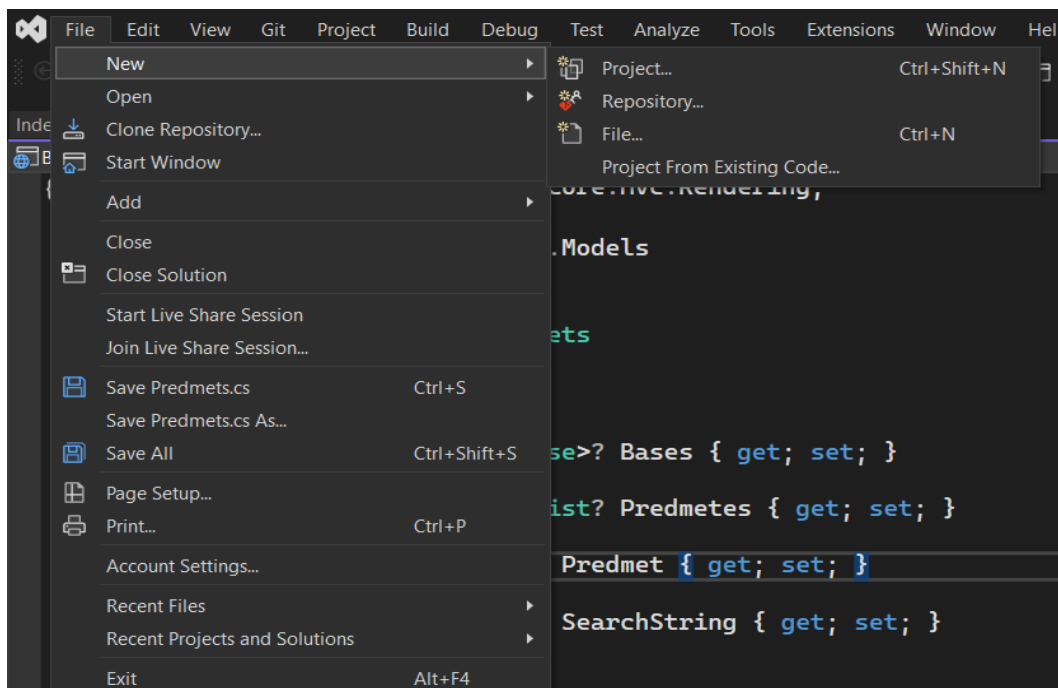


Рисунок 8 Контекстне меню для створення рішення

З запропонованого списку проєктів, необхідно обрати –ASP.Net Core, як на рисунку 9

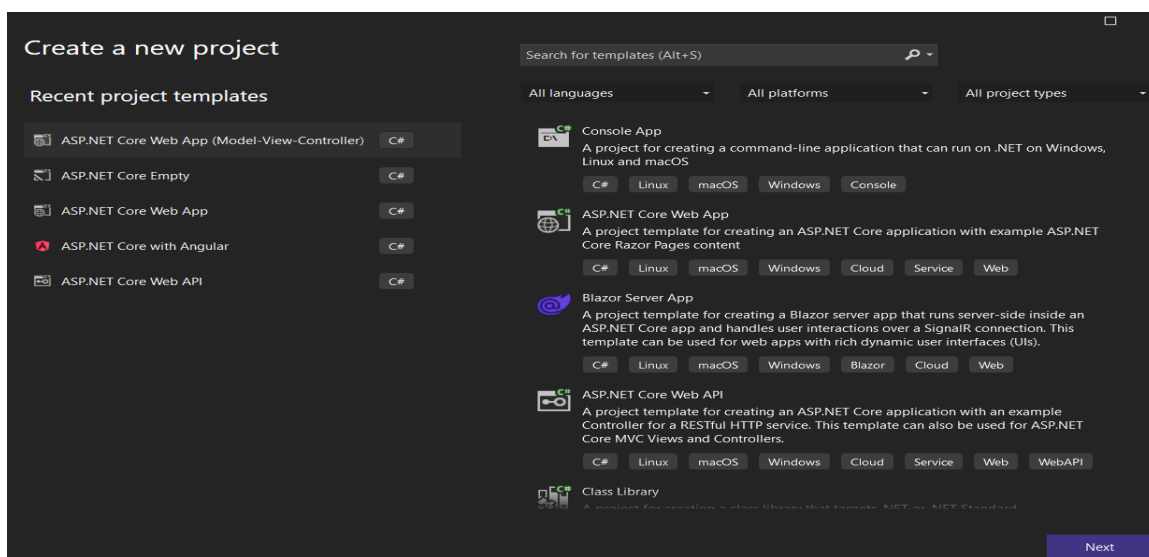


Рисунок 9 Вибір типу проєкта

У виборі проєкту обрати шаблон ASP.NET Пустий щоб можна було створити з нуля веб-додаток Рисунок 10

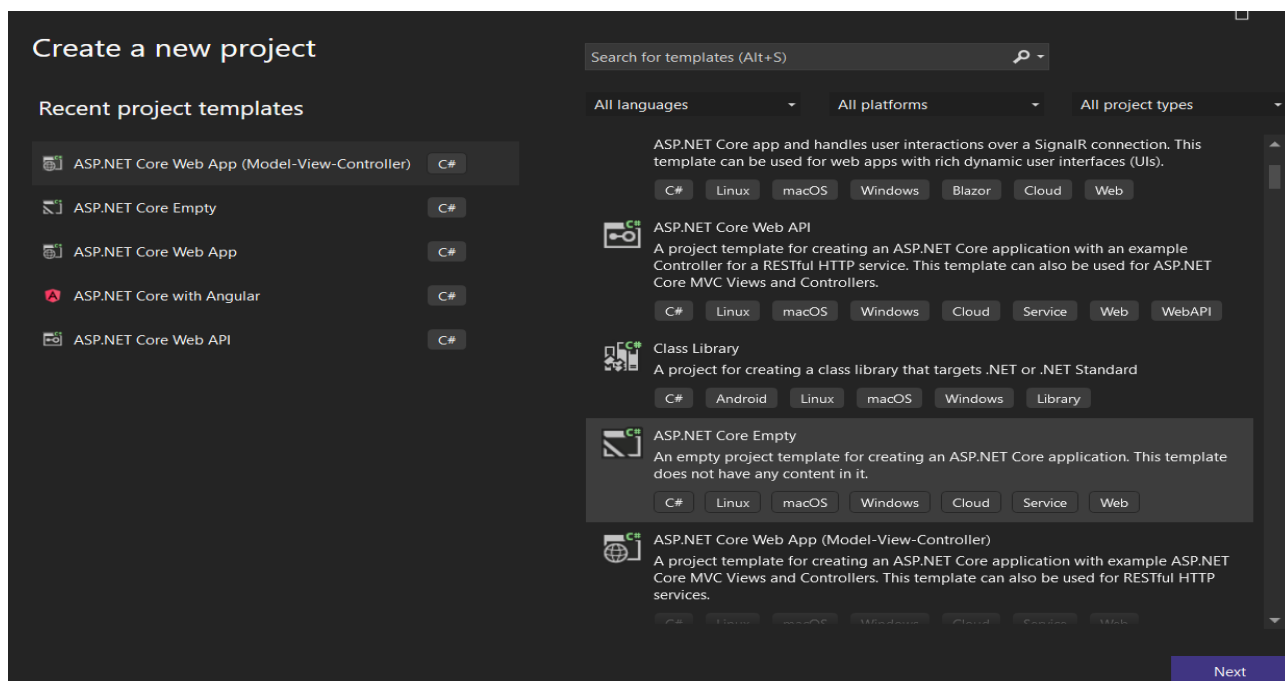


Рисунок 10 Шаблон веб-додатку ASP.NET

Таким чином серед Visual Studio створить пустий проєкт з відповідною структурою папок необхідних для початку роботи з розробки веб-додатку На Рисунку 11 відображена початкова структура папок створених після обрання макету ASP.NET Пустий

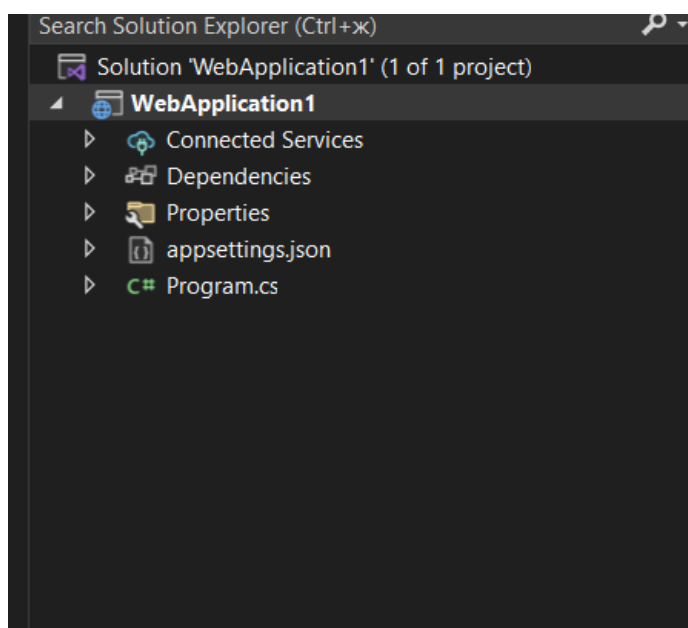


Рисунок 11 Структура папок додатку

Під час першого запуску проєкту він матиме початкову структуру яка була задана розробниками програми для того щоб можна було надалі розробляти додаток

Hello World!

Рисунок 12 Перший запуск проєкту і його відображення у браузері

3.2 Створення структури каталогів проєкта

Наступний крок заключається в створення папок які будуть містити компоненти необхідні для цього додатку) моделі контроллери і представлення

- Models - ця папка буде містити класи мо;
- Controllers - ця папка буде містити необхідні контроллери
- Views - в цій папці будуть файли RAZOR.

Структура папок для додатку показана на рисунку 13.

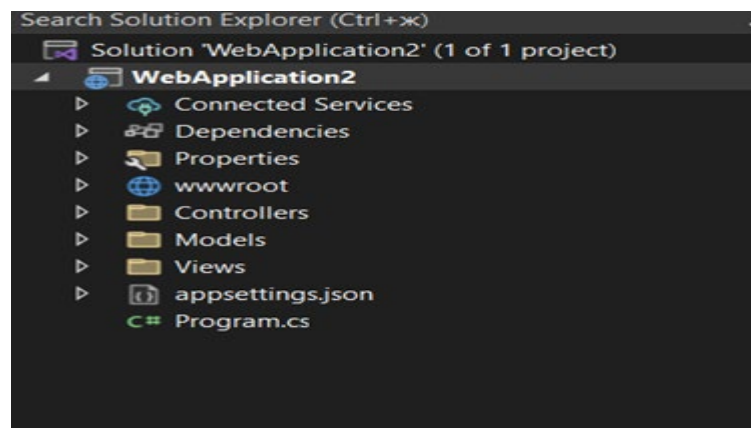


Рисунок 13 Структура каталогу проєкта

3.3 Налаштування конфігурації веб-додатку

Для роботи з архітектурою ASP.NET необхідно встановити відповідні пакети які зображені на Рисунку 14

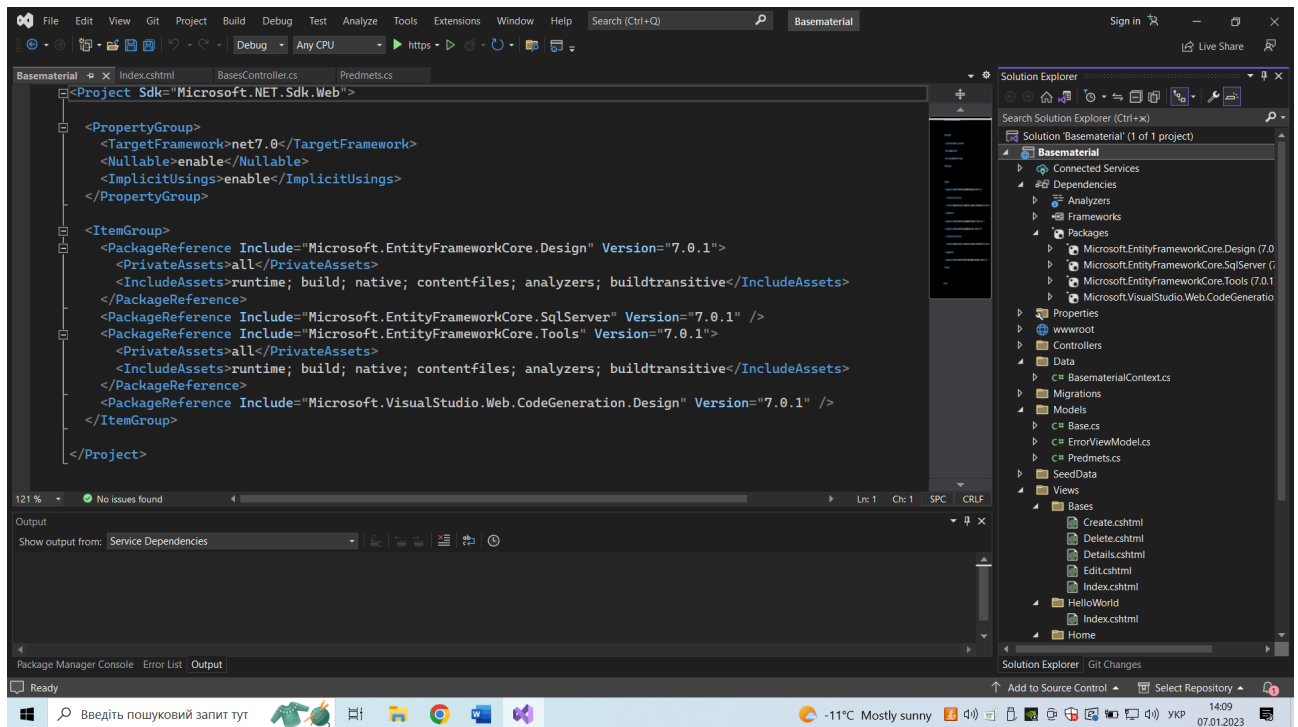


Рисунок 14 Пакети необхідні для роботи з архітектурою ASP.NET

Для того, щоб додаток правильно працював і використовував відповідні компоненти необхідно в класі конфігурації підключити залежності та маршрутизацію, як показано на Рисунку 15

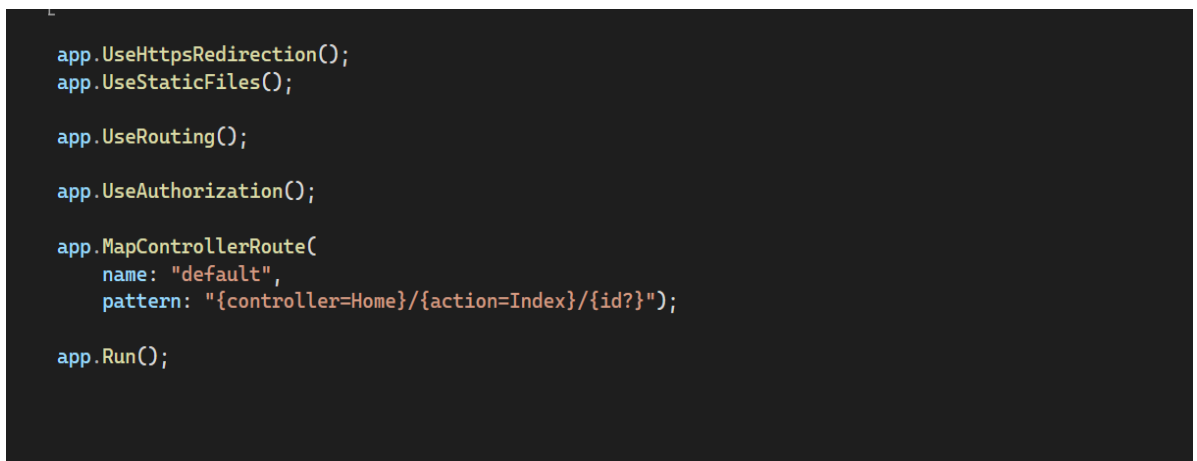


Рисунок 15 Додавання маршрутизації додатку.

3.4 Зміна макету проєкту

Обравши посилання в меню (База даних , Головна сторінка та Контактні дані). Меню на кожній сторінці мають однаковий макет. Макет меню реалізований у файлі Views/Shared/_Layout.cshtml. Рисунок 16

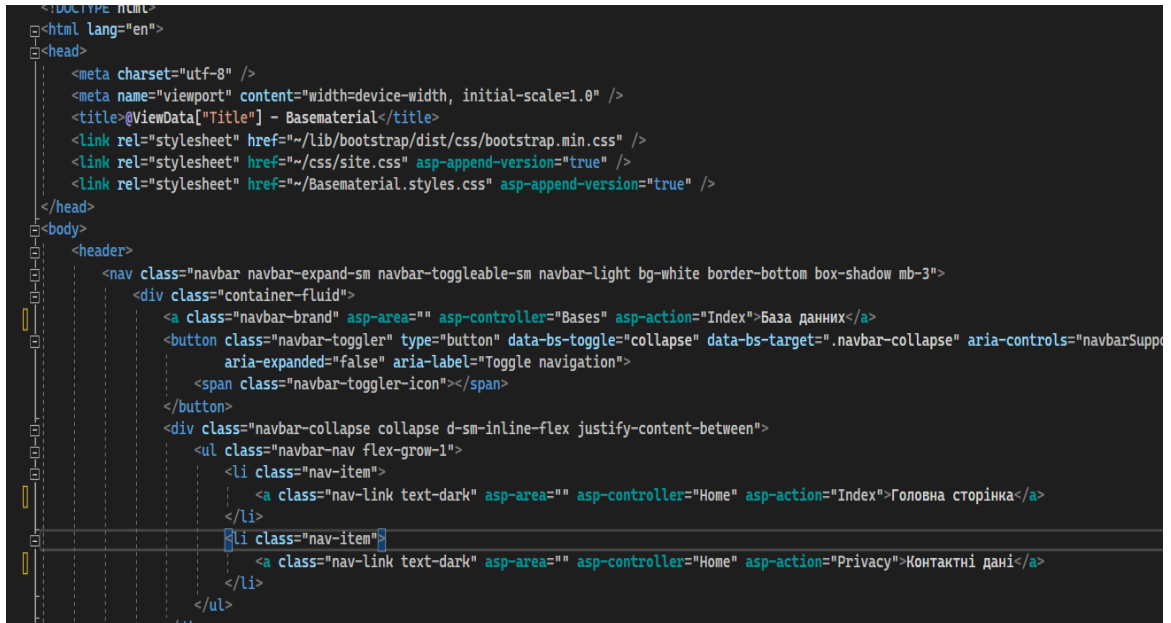


Рисунок 16 Макет меню

Шаблони макета дозволяють:

- вказати макет контейнера HTML сайту в одному місці;
- застосовувати макета контейнера HTML на кількох сторінках сайту.
- **RenderBody** — це заповнювач, в якому відображаються всі

сторінки, що створюються для певних уявлень, упаковані на сторінці макета. Якщо клацнути на посилання то відкриється , в файл розташований в папці Views/Home/Contact.cshtml. Рисунок 17

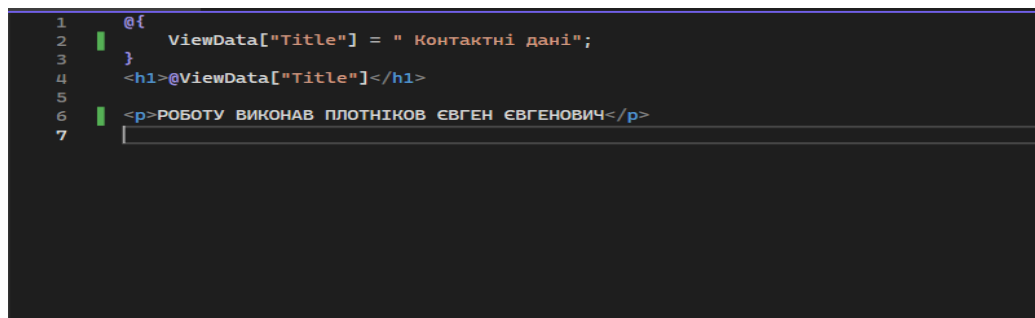


Рисунок 17 Сторінка контактні дані

3.5 Зміна заголовка, нижнього колонтитулу та посилання меню у файлі макета

Змінюємо вміст файлу Views/Shared/_Layout.cshtml Змінюючи назву відповідних вкладок і додавши наступні зміни до файлу Рисунок 18

Наведена нижче розмітка вносить наступні зміни:

- заміна трьох входжень Home на Bases;
- заміна елементів прив'язки

`Home` на

`База даних`

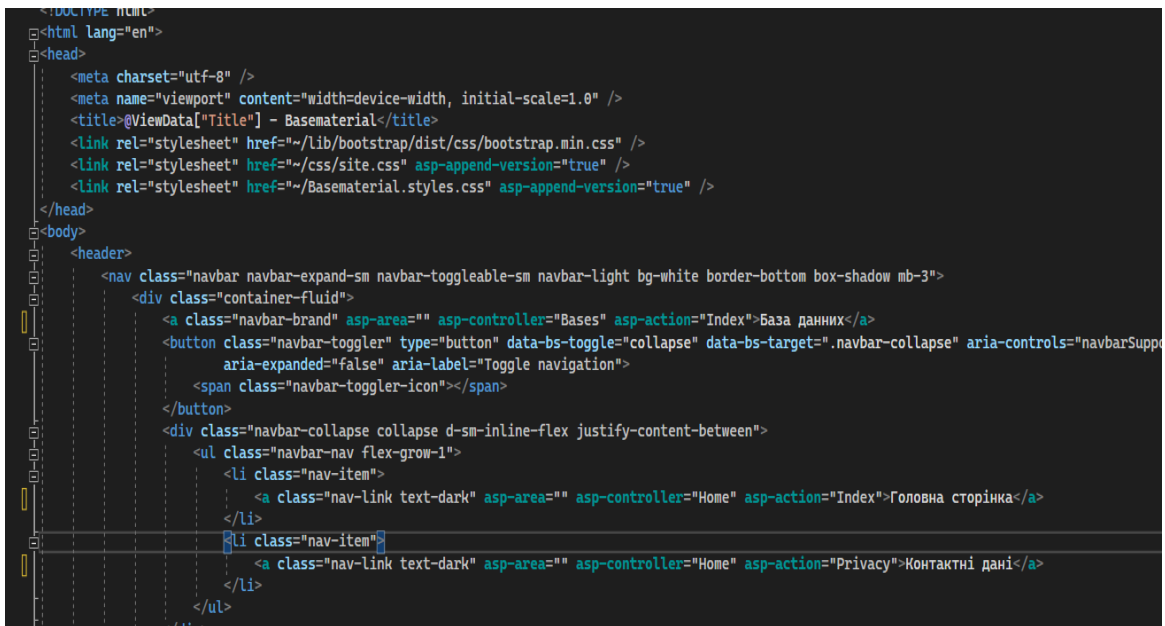


Рисунок 18 Змінений файл макету

У наведеній вище розмітці атрибут допоміжної функції тега прив'язки `asp-area=""` і значення атрибута були опущені, так як ця програма не використовує області (Areas).

На данному етапі контроллер Bases який використовується як посилання ще не реалізований і тому він ще не працює

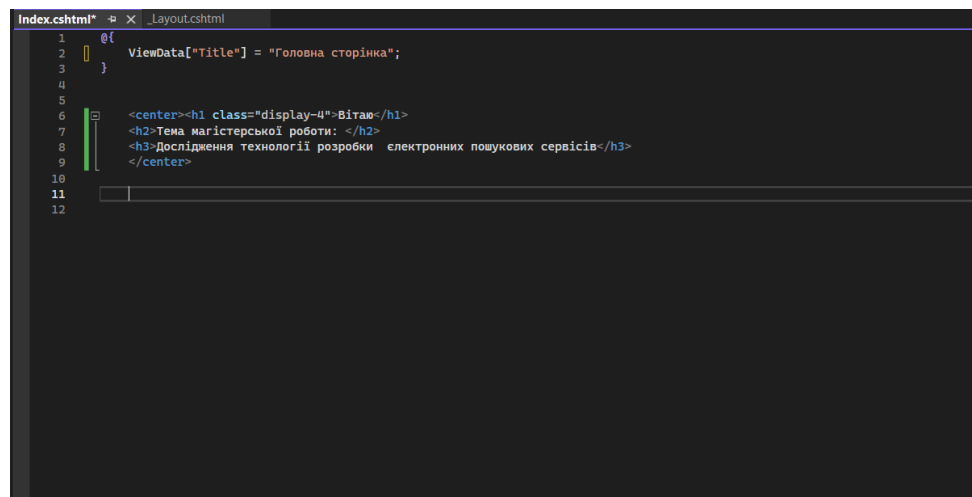
Далі переходимо до налаштування стартової сторінки в данному проєкті який реалізований у файлі Views/Home /Index.cshtml (Рисунок 19)

Зверніть увагу, що змінили такі елементи:

- Заголовок браузера.
- Основний заголовок.
- Додаткові заголовки.

Якщо в браузері зміни не відображаються, це може означати, що вміст кешований. У цьому випадку натисніть у браузері клавіші CTRL+F5 для примусового завантаження відповіді сервера. Заголовок браузера створюється за допомогою атрибута `ViewData["Title"]`, який задається в шаблоні представлення `Index.cshtml` та додаткового рядка "- Movie App", що додається до файлу макета.

Вміст шаблону представлення `Index.cshtml` поєднується з шаблоном представлення `Views/Shared/_Layout.cshtml`. В браузер надсилається одна HTML-відповідь. Шаблони макетів дозволяють легко вносити зміни до всіх сторінок у додатку.



```
1  @{
2      ViewData["Title"] = "Головна сторінка";
3  }
4
5
6  <center><h1 class="display-4">Вітаю</h1>
7      <h2>Тема магістерської роботи: </h2>
8      <h3>Дослідження технології розробки електронних пошукових сервісів</h3>
9  </center>
10
11
12
```

Рисунок 19 Змінений файл `Index.cshtml`

3.6 Передача даних з контролера на подання

Дії контролера викликаються у відповідь на запит URL-адреси. Код, що обробляє запити браузера, додається в клас контролера. Контролер отримує дані з джерела даних і визначає тип відповіді, який буде відправлено в браузер.

Контролер може використовувати шаблони уявлень для створення та форматування відповіді HTML, що надсилається браузеру.

Шаблони не повинні:

- виконувати бізнес-логіку;
- безпосередньо взаємодіяти із базою даних.

Натомість вони повинні працювати тільки з даними, які їм надає контролер. Подібний поділ сфер відповідальності дозволяє забезпечити максимальну оптимізацію коду, а також зручність його:

- очищення;
- тестування;
- обслуговування.

Замість відображення відповіді у вигляді рядка налаштуйте контролер для використання шаблону подання. Шаблон подання створює динамічний відповідь, щоб одержати якого необхідно передати відповідні дані з контролера у виставу.

Для цього контролер може помістити динамічні дані (параметри), які потрібні шаблону подання, до словника ViewData, до якого згодом буде звертатися цей шаблон за динамічними даними.

Словник ViewData є динамічним об'єктом, а це означає, що можна використовувати будь-який тип.

Об'єкт ViewData не має певних властивостей, доки не буде додано будь-який елемент.

Система прив'язки моделі MVC автоматично зіставляє іменовані параметри name і numTimes із рядка запиту з параметрами методу.

Об'єкт словника ViewData містить дані, які будуть передаватися до подання.

3.7 Створення класу моделі контролера

Обраши папку Models ми створюємо клас Base.cs.

Клас Base.cs. містить поле Id, який потрібно бази даних як первинний ключ для початку створення

Атрибут DataType для Datepublic визначає тип даних (Date).

Із цим атрибутом:

- Користувачеві не потрібно вводити відомості про час у полі дати.
- Відображається лише дата, а не час.

Після цього для наступної роботи нам потрібно підключили необхідні пакети для роботи які можна виклавши відповідними командам через консоль пакету менеджерів Nuget:

- Install-Package Microsoft.VisualStudio.Web.CodeGeneration.Design
- Install-Package Microsoft.EntityFrameworkCore.Design
- Install-Package Microsoft.EntityFrameworkCore.SqlServer

Після додавання пакетів виконуємо складання проєкту для перевірки його на помилки, які можуть виникнути після додавання пакетів у проєкт додатку.

3.8 Формування шаблону сторінки Баз даних

Для цього будемо використовувати засіб формування шаблонів, щоб створити сторінки Create, Read, Update та Delete (CRUD) для моделі бази даних.

У браузері рішень натискаємо праву кнопку миші на папці Controllers і обираємо Додати(Add) > New Scaffolded Item (Створити шаблонний елемент).

Далі У діалоговому вікні Додати новий елемент із шаблоном обравши Контролер MVC з уявленнями, використовуючи команду Додати Entity Framework> Рисунок 20.

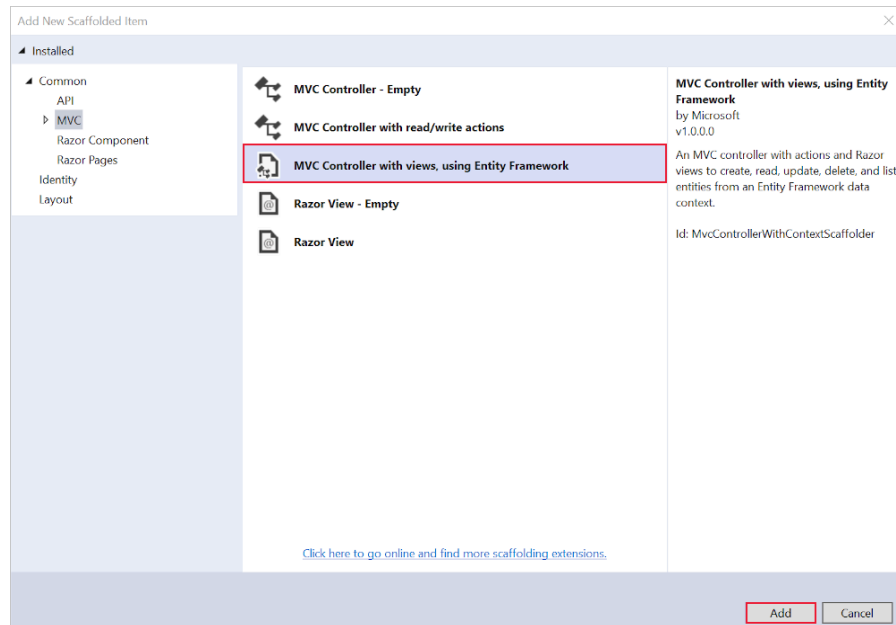


Рисунок 20 Додавання контролеру MVC

У діалоговому вікні додавання контролера MVC з поданнями з використанням Entity Framework і були виконані наступні дії

- у списку було обрано клас моделі (Base.Models) .
- у рядку Клас контексту даних натиснути на (+).
- у діалоговому вікні Додавання контексту даних автоматично створюється ім'я класу BasematerialContext.
- обираємо додати.
- ім'я контролера залишаємо стандартним і обираємо додати (Рисунок 21)

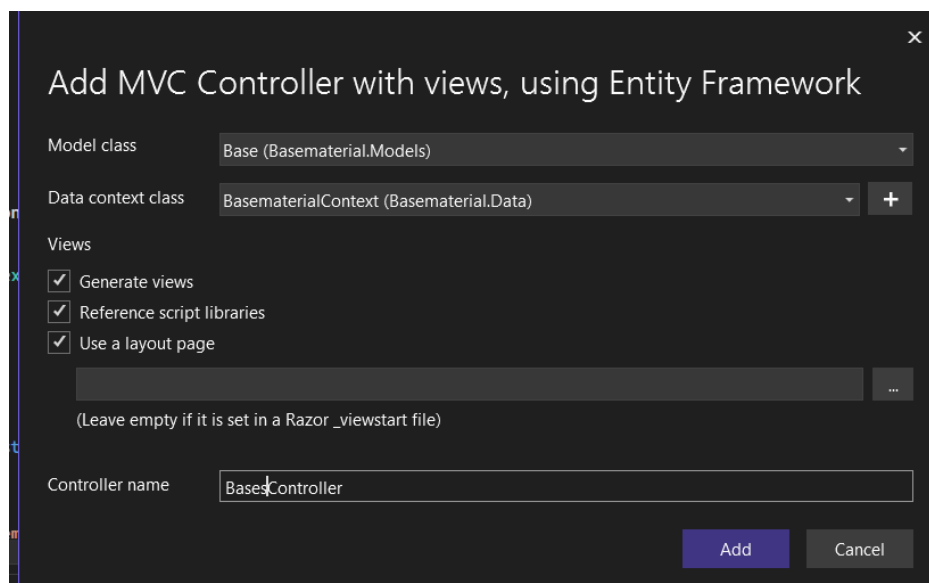


Рисунок 21 Налаштування MVC контролера

При формуванні шаблонів виконується автоматичне створення таких об'єктів:

- бази даних `Controllers/BaseController.cs`;
- файли представлення Razor для сторінок `Create`, `Delete`, `Details`, `Edit` та `Index`: `Views/Base/*.cshtml`;
- клас контексту бази даних: `Data/BasematerialContext.cs`.

Автоматичне створення цих файлів та його оновлення називається формуванням шаблонів.

Сформовані сторінки ще не можна використовувати, оскільки база даних не існує.

Запуск програми та вибір посилання База даних призводить до появи повідомлення про помилку Не вдається відкрити базу даних бо вона ще не створенна.

Після цього виконуємо збірку програму для перевірки її на помилки.

Наступним кроком буде міграція створених даних в базу даних

Для того щоб це зробити потрібно виконати такий алгоритм дії.

У меню Сервіс послідовно обрати пункти Диспетчер пакетів NuGet>Консоль диспетчера пакетів.

Після цього ввести у консолі диспетчера пакетів такі команди

- `Add-Migration InitialCreate`
- `Update-Database`

А тепер розглянемо що кожна з цих команд виконала.

1. `Add-Migration InitialCreate`. Створює міграційний файл `Migrations/{timestamp}_InitialCreate.cs`.

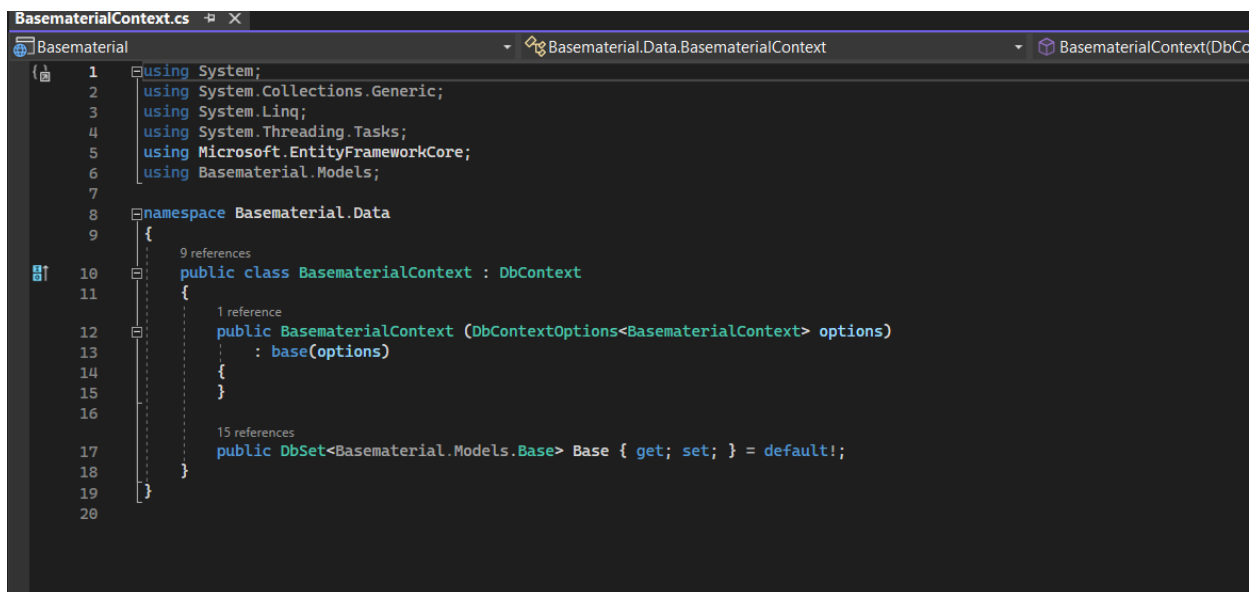
Аргумент `InitialCreate` – це ім'я міграції. Можна використовувати будь-яке ім'я, але за згодою вибирається ім'я, яке описує міграцію. Так як це перша міграція, створений клас містить код створення схеми бази даних. Схема бази даних створюється на основі моделі, вказаної в класі `BaseMaterialContext`.

2. Update-Database. Оновлює базу даних до останньої міграції, створеної попередньою командою. Ця команда виконує метод Up у файлі Migrations/{time-stamp}_InitialCreate.cs, який створює базу даних.

3.9 Огляд створеного класу контексту бази даних

При EF Core використання доступу до даних здійснюється за допомогою моделі. Модель складається з класів сутностей та об'єкта контексту, що представляє сеанс взаємодії з базою даних. Об'єкт контексту дозволяє виконувати запити та зберігати дані. Контекст бази даних успадковується від Microsoft.EntityFrameworkCore.DbContext і визначає сутності, які потрібно включити до моделі даних.

Під час формування шаблонів створюється клас контексту бази даних Data/BasematerialContext.cs (Рисунок 22):



```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.EntityFrameworkCore;
6  using Basematerial.Models;
7
8  namespace Basematerial.Data
9  {
10     public class BasematerialContext : DbContext
11     {
12         public BasematerialContext(DbContextOptions<BasematerialContext> options)
13             : base(options)
14         {
15         }
16
17         public DbSet<Basematerial.Models.Base> Base { get; set; } = default!;
18     }
19 }

```

Рисунок 22 Файл BasematerialContext.cs

Також під час оновлення бази даних було оновленні залежності в файлах.

ASP.NET Core підтримує використання залежностей. Служби, такі як контекст бази даних, реєструються за допомогою Program.cs.

Ці служби надаються компонентам, які вимагають їх через параметри

конструктора.

У файлі `Controllers/BaseController.cs` цей конструктор застосовує використання залежностей для впровадження контексту бази даних `BasematerialContext` в контролер.

Контекст бази даних використовується в кожному методі створення, читання, оновлення та видалення в контролері.

При формуванні шаблонів був створений наступний код:

```
builder.Services.AddDbContext<BasematerialContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("BasematerialContext")) ?? throw new InvalidOperationException("Connection string 'BasematerialContext' not found."));
```

Також під час формування шаблону у файл `appsettings.json` було додано такий рядок підключення - `"ConnectionStrings": {"BasematerialContext": "Server=(localdb)\\mssqllocaldb;Database=Basematerial.Data;Trusted_Connection=True;MultipleActiveResultSets=true"`

Під час локальної розробки система конфігурації ASP.NET Core зчитує ключ `ConnectionString` із файлу `appsettings.json`.

Розглянувши клас `InitialCreate` який розміщений в папці `Migrations/{timestamp}_InitialCreate.cs`. (Рисунок 23)

```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "Base",
        columns: table => new
        {
            Id = table.Column<int>(type: "int", nullable: false)
                .Annotation("SqlServer:Identity", "1, 1"),
            Title = table.Column<string>(type: "nvarchar(max)", nullable: true),
            Datepublic = table.Column<DateTime>(type: "datetime2", nullable: false),
            Predmet = table.Column<string>(type: "nvarchar(max)", nullable: true),
            Class = table.Column<decimal>(type: "int", nullable: false)
        },
        constraints: table =>
        {

```

Рисунок 23 Файл `InitialCreate`

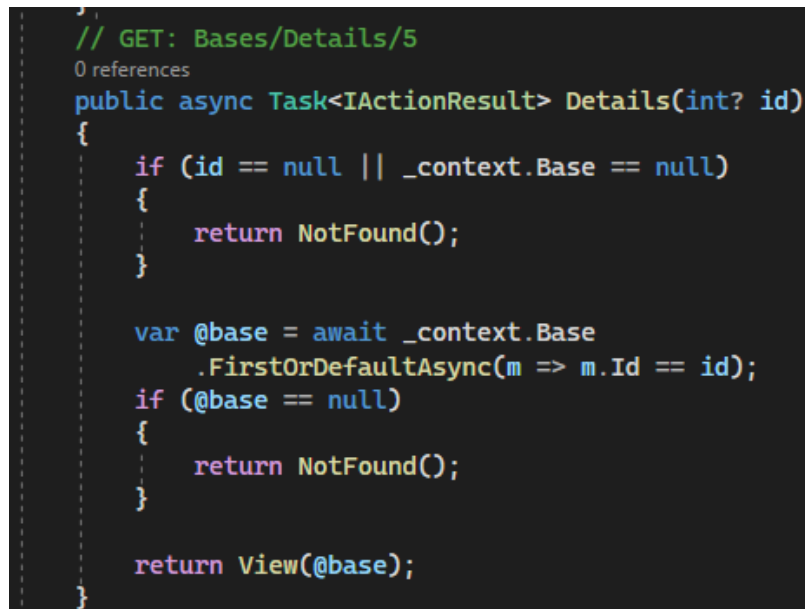
У наведеному вище коді: метод `InitialCreate.Up` створює таблицю `Base` і налаштовує `Id` як первинний ключ; метод `InitialCreate.Down` скасовує зміни схеми, внесені під час міграції `Up`.

3.10 Метод Details

Словник ViewData є динамічним об'єктом, який реалізує зручний механізм пізнього зв'язування передачі інформації в представлення.

Модель підтримує передачу строго типізованих об'єктів моделі у виставу. Такий підхід забезпечує перевірку коду під час компіляції. Механізм формування шаблонів передає строго типізовану модель класу BaseController і представлення.

Для початку розглянемо цей метод розміщений в Controllers/BaseController.cs: (Рисунок 24)



```
// GET: Bases/Details/5
0 references
public async Task<IActionResult> Details(int? id)
{
    if (id == null || _context.Base == null)
    {
        return NotFound();
    }

    var @base = await _context.Base
        .FirstOrDefaultAsync(m => m.Id == id);
    if (@base == null)
    {
        return NotFound();
    }

    return View(@base);
}
```

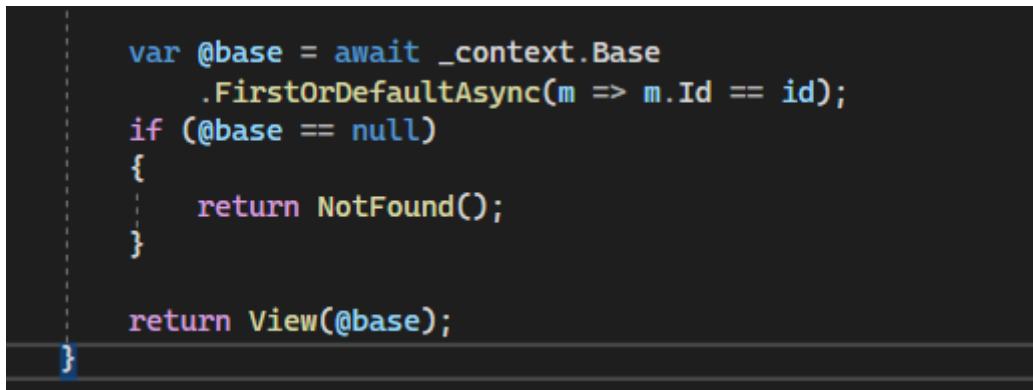
Рисунок 24 Метод Details в контролері

Параметр id зазвичай передається як дані маршруту. Наприклад, [https://localhost:7258 /Base/details/1](https://localhost:7258/Base/details/1) задає:

- контролер до контролера base перший сегмент URL-адреси;
- дія для details, другий сегмент URL-адреси;
- значення 1 для ID, останній сегмент URL-адреси.

Параметр id можна передати за допомогою рядка ось такого рядку запиту, <https://localhost:7258/Base/details?id=1>.

Параметр `id` визначається як тип, що допускає значення `NULL` (`int?`), якщо не вказано значення `id`. Лямбда-вираз передається у метод `FirstOrDefaultAsync` для вибору записів з бази, які відповідають даним маршруту або значенням рядка запиту. (Рисунок 25)



```
var @base = await _context.Base
    .FirstOrDefaultAsync(m => m.Id == id);
if (@base == null)
{
    return NotFound();
}

return View(@base);
```

Рисунок 25 Передача рядку запиту

Продовжуємо розглядати Метод розміщений `Views/Base/Details.cshtml`.

Оператор `@model` на початку файлу представлення задає тип об'єкта, який очікується представленням. Під час створення контролера `base` було включено наступний оператор `@model`:

Ця директива `@model` забезпечує доступ до даних, який контролер передав. Об'єкт `Model` є строго типізованим. У поданні `Details.cshtml` код передає кожне поле даних до допоміжних функцій `HTML.DisplayNameFor` і `DisplayFor` зі строго типізованим об'єктом `Model`.

Методи `Create` та `Edit` та уявлення також передають об'єкт моделі `Base`.

Розглянувши метод `Index.cshtml` у контролері `Base`. Зверніть увагу, як у коді створюється об'єкт `List` під час виклику методу `View`. Код передає список `Base` з методу дії `Index` на подання.

Код повертає інформацію про проблему , `Base` якщо властивість контексту даних має значення `NULL`.

Ця директива `@model` забезпечує доступ до даних, який контролер передав з використанням строго типізованого об'єкта `Model`. У поданні

Index.cshtml код циклічно перебирає дані з використанням оператора foreach для строго типізованого об'єкта Model. Оскільки об'єкт Model є строго типізованим (як і об'єкт IEnumerable<Base>), кожен елемент циклу отримує тип Base. Крім інших переваг, компілятор перевіряє типи, у коді. (Рисунок 26)

```
@foreach (var item in Model.Bases!)
{
<tr>
<td>
@Html.DisplayFor(modelItem => item.Title)
</td>
<td>
@Html.DisplayFor(modelItem => item.Datepublic)
</td>
<td>
@Html.DisplayFor(modelItem => item.Predmet)
</td>
<td>
@Html.DisplayFor(modelItem => item.Class)
</td>
<td>
<a asp-action="Edit" asp-route-id="@item.Id">Редагувати</a> |
<a asp-action="Details" asp-route-id="@item.Id">Детальніше</a> |
<a asp-action="Delete" asp-route-id="@item.Id">Видалити</a>
</td>
</tr>
</tbody>
</table>
```

Рисунок 26 Оператор foreach у файлі Index.cshtml

Далі Створюємо клас SeedData у папці SeedData. для заповнення бази даних з наступним кодом який зображений на Рисунку 27:

```
1 public static void Initialize(IServiceProvider serviceProvider)
2 {
3     using (var context = new BasematerialContext(
4         serviceProvider.GetRequiredService<
5             DbContextOptions<BasematerialContext>>())
6     {
7         // Look for any movies.
8         if (context.Base.Any())
9         {
10             return; // DB has been seeded
11         }
12         context.Base.AddRange(
13             new Base
14             {
15                 Title = "Поняття мови розмітки гіпертексту. Етапи створення веб-сайтів",
16                 Datepublic = DateTime.Parse("09-09-2022"),
17                 Predmet = "Інформатика",
18                 Class = 8
19             },
20             new Base
21             {
22                 Title = "Сервіси розміщення аудіо та відео файлів в Інтернеті",
23                 Datepublic = DateTime.Parse("09-09-2022"),
24                 Predmet = "Інформатика",
25                 Class = 8
26             },
27             new Base
28             {
29                 Title = "Розвиток фінансових відносин в Україні",
30                 Datepublic = DateTime.Parse("09-09-2022"),
31                 Predmet = "Інформатика",
32                 Class = 8
33             },
34             new Base
35             {
36                 Title = "Розвиток фінансових відносин в Україні",
37                 Datepublic = DateTime.Parse("09-09-2022"),
38                 Predmet = "Інформатика",
39                 Class = 8
40             }
41         )
42     }
43 }
```

Рисунок 27 Клас SeedData

Далі додаємо ініціалізатора заповнення до файлу Program.cs з наступним кодом. Рисунок 28

```
var app = builder.Build();
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;

    SeedData.Initialize(services);
}
```

Рисунок 28 Ініціалізатор заповнення

Після додавання ініціалізатора в базі будуть додані дані які були внесені раніше у файлі SeedData

Title	Date public)	Predmet	Class	
Поняття мови розмітки гіпертексту. Етапи створення веб-сайтів	09.09.2022	Інформатика	8	Редагувати Детальніше Видалити
Сервіси розміщення аудіо та відео файлів в Інтернеті	09.09.2022	Інформатика	8	Редагувати Детальніше Видалити
Розвиток фінансових відносин в Україні	09.09.2022	Інформатика	8	Редагувати Детальніше Видалити
Побудова аудіо- та відеоряду	09.09.2022	Інформатика	8	Редагувати Детальніше Видалити

Риснок 29 Результат виконання після додавання ініціалізатор

3.11 Методи та подання контролера в ASP.NET Core

Додаток для роботи з базою майже готовий але потрібно ще зхмінити відображення Date public щоб воно відображалось як два слово а не як одне.

Для цього відкриваємо файл Base.cs і додаємо до цього файлу ось такі рядки :

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
```

```
[Display(Name = "Date public")]
```

```
DataType(DataType.Date)]
```

Атрибут `Display` визначає ім'я поля, що відображається (у цьому випадку "Date public" замість " Datepublic "). Атрибут `DataType` визначає тип даних (`Date`), тому інформація про час, що зберігається в полі, не відображається.

Щоб `Entity Framework Core` коректно зіставила `Class` з даним у базі даних потрібно вказати тип даних типу `int`

Тепер переходимо до зміни відображення посилань `Edit` (Редагувати), `Details` (Детальніше) та `Delete` (Видалити) ці елементи створюються допоміжною функцією тегів прив'язки `Core` у файлі `Views/Bases/Index.cshtml`.

```
<td>
    <a asp-action="Edit" asp-route-id="@item.Id">Редагувати</a> |
    <a asp-action="Details" asp-route-id="@item.Id">Детальніше</a> |
    <a asp-action="Delete" asp-route-id="@item.Id">Видалити</a>
</td>
```

Рисунок 30 Елементи прив'язки

Допоміжні функції тегів дозволяють серверному коду брати участь у створенні та відображенні HTML-елементів у файлах `Razor`. У наведеному вище коді `AnchorTagHelper` динамічно створює значення атрибуту HTML `href` на підставі методу дії контролера та ідентифікатора маршруту.

Для вивчення створеної розмітки можна використати функцію перегляду вихідного коду у браузері або засобу розробника.

Формат для маршрутизації був задан у файлі `Program.cs` за допомогою команди: `pattern: "{controller=Home}/{action=Index}/{id?}"`.

`ASP.NET Core` перетворює `https://localhost:7258/Base/Edit/4` на запит методу дії `Edit` контролера `Base` з параметром `Id`. (Методи контролера також називаються методами дії.)

Допоміжні функції тегів є однією з найпопулярніших нових можливостей `ASP.NET Core`.

3.12 Метод Edit

Розглянемо метод дії Edit. У наступному коді демонструється метод HTTP GET Edit, який виконує вибірку фільмів та заповнює форму редагування, створену файлом Edit.cshtmlRazor.

Нижче показано метод HTTP POST Edit, який є власником переданих значень даних:

```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Edit(int id, [Bind("Id,Title,Datepublic,Predmet,Class")] Base @base)
{
    if (id != @base.Id)
    {
        return NotFound();
    }
}
```

Рисунок 31 Метод HTTP POST Edit

Атрибут [Bind] є одним із способів захисту від надмірної передачі даних. Властивості необхідно включати лише в той атрибут [Bind], який потрібно змінити.

ViewModels реалізує альтернативний підхід до захисту від надмірної передачі даних.

Зверніть увагу на другий спосіб дії Edit, якому передують атрибути [HttpPost].

Атрибут HttpPost вказує на те, що цей метод Edit може викликатись лише для запитів POST. Можна застосувати атрибут [HttpGet] до першого методу редагування, але це необов'язково.

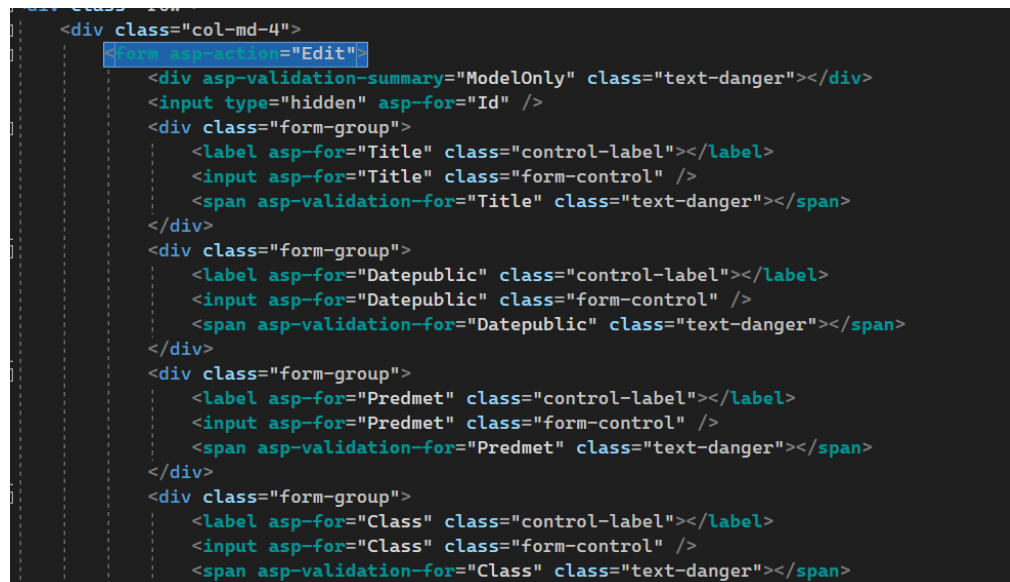
Атрибут ValidateAntiForgeryToken використовується для захисту від підробки запитів та використовується спільно з відповідним маркером безпеки, який створюється у файлі представлення редагування (Views/Bases/Edit.cshtml).

У файлі шаблону редагування для створення маркера захисту від підробки використовується допоміжна функція Form : <form asp-action="Edit">.

Допоміжна функція тега Form створює прихований маркер захисту від підробки, який повинен відповідати [ValidateAntiForgeryToken] аналогічному маркеру безпеки методом Edit контролера Bases.

Метод HttpGet Edit приймає параметр бази ID, виконує пошук даних з використанням методу FindAsync платформи Entity Framework та повертає вибрані дані на подання редагування. Якщо дані не вдається знайти, повертається помилка NotFound (HTTP 404).

Якщо у вигляді редагування створено систему формування шаблонів, вона перевіряє клас Base і створює код для відображення елементів <label> та <input> для кожної властивості класу. Ось так виглядає створення редагування, системою формування шаблонів Visual Studio(Рисунок 32).



```

<div class="col-md-4">
    <form asp-action="Edit">
        <div asp-validation-summary="ModelOnly" class="text-danger"></div>
        <input type="hidden" asp-for="Id" />
        <div class="form-group">
            <label asp-for="Title" class="control-label"></label>
            <input asp-for="Title" class="form-control" />
            <span asp-validation-for="Title" class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="Datepublic" class="control-label"></label>
            <input asp-for="Datepublic" class="form-control" />
            <span asp-validation-for="Datepublic" class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="Predmet" class="control-label"></label>
            <input asp-for="Predmet" class="form-control" />
            <span asp-validation-for="Predmet" class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="Class" class="control-label"></label>
            <input asp-for="Class" class="form-control" />
            <span asp-validation-for="Class" class="text-danger"></span>
        </div>
    </form>
</div>

```

Рисунок 32 Відредагований шаблон Edit

На початку файлу шаблону міститься оператор @model Basematerial.Models.Base Який вказує, що у поданні потрібна модель шаблону з типом Base.

Для оптимізації розмітки HTML є код що використовує кілька способів допоміжних функцій тегів. Допоміжна функція тега Label відображає ім'я поля ("Title", "Datepublic", "Predmet" і "Class").

Допоміжна функція тега Input відображає елемент HTML <input>. Допоміжна функція тега Validation відображає будь-які повідомлення

перевірки, пов'язані із зазначеною властивістю.

Елементи `<input>` знаходяться в елементі HTML `<form>`, атрибут `action` якого задає передачу даних за URL-адресою `/Movies/Edit/id`. Дані форми передаватимуться на сервер при натисканні кнопки `Save`. В останньому рядку перед елементом, що закриває `</form>`, відображається прихований маркер XSRF, створений допоміжною функцією тега `Form`.

Атрибут `[ValidateAntiForgeryToken]` перевіряє прихований маркер безпеки XSRF, створений генератором маркерів у допоміжній функції тега `Form`

Система моделі прив'язки приймає передані значення форми та створює об'єкт `Base`, який передається як параметр `base`. Властивість `ModelState.IsValid` перевіряє, чи можна використовувати передані у формі дані для зміни (редагування або оновлення) об'єкта `Base`. Допустимі дані зберігаються. Оновлені (змінені) дані фільму зберігаються в базі даних через виклик методу `SaveChangesAsync` в контексті бази даних.

Після збереження даних код перенаправляє користувача на метод дії `Index` класу `BasesController`, який відображає дані з урахуванням щойно внесених змін.

Перед відправкою форми на сервер за клієнта перевіряється виконання всіх правил перевірки для полів.

При виявленні помилок перевірки відображається повідомлення про помилку, а форма не надсилається. Якщо JavaScript вимкнено, перевірка на стороні клієнта не виконується.

Проте, сервер виявить передані неприпустимі значення, внаслідок чого значення форми будуть відображені повторно з повідомленнями про помилки. Допоміжна функція тега `Validation` у шаблоні `Views/Bases/Edit.cshtml` забезпечує відображення відповідних повідомлень про помилку.

3.13 Пошук в базі даних

На початку для додавання пошуку в нашу базу даних ми змінимо метод `Index` можливості пошуку, які дозволяють виконувати пошук даних за предметом та темою.

Оновимо код який знаходиться у файлі `Controllers/BasesController.cs`, додавши наступний код:

```
public async Task<IActionResult> Index(string Predmet, string searchString)
{
    if (_context.Base == null)
    {
        return Problem("Entity set 'Basematerial.Context' is null.");
    }

    IQueryable<string> genreQuery = from m in _context.Base
                                    orderby m.Predmet
                                    select m.Predmet;

    var bases = from m in _context.Base
                select m;

    if (!string.IsNullOrEmpty(searchString))
    {
        bases = bases.Where(s => s.Title!.Contains(searchString));
    }
    if (!string.IsNullOrEmpty(Predmet))
    {
        bases = bases.Where(x => x.Predmet == Predmet);
    }
    var PredmetVM = new Predmets
    {
        Predmetes = new SelectList(await genreQuery.Distinct().ToListAsync()),
        Bases = await bases.ToListAsync()
    };

    return View(PredmetVM);
}
```

Рисунок 33 Оновлений код `Index`

Наступний рядок у методі `Index` action створює запит LINQ для вибору даних `var bases = from m in _context.Base select m;`

Цей запит визначається лише в цій точці і не виконується для бази даних.

Якщо параметр `searchString` містить рядок, запиту даних змінюється для фільтрації за значенням у рядку пошуку. `if (!string.IsNullOrEmpty(searchString)) {bases = bases.Where(s => s.Title!.Contains(searchString));}`

Наведений вище код `s => s.Title! Contains(searchString)` являє собою лямбда-вираз.

Лямбда-вирази використовуються в запитах LINQ на основі методів як аргументи стандартних методів операторів запиту, таких як метод Where або Contains (використовується в наведеному вище коді).

Запити LINQ не виконуються, якщо вони визначаються або змінюються шляхом виклику методу, наприклад, Where, Contains або OrderBy.

Натомість виконання запиту відкладається. Це означає, що обчислення виразу відкладається до тих пір, поки не буде виконано ітерацію його реалізованого значення або поки не буде викликаний метод ToListAsync.

Метод Contains виконується у базі даних, а не в коді C#, Реєстр символів запиту враховується залежно від параметрів бази даних та сортування.

У SQL Server метод Contains зіставляється у якому реєстр символів не враховується. У SQLite за параметрами сортування за умовчанням реєстр символів враховується.

Тепер відкриваємо файл Index і додаємо розмітку тега <form> для відображення рядка пошуку по двом категоріям по темі і по назві предмета.

```
<<form asp-controller="Bases" asp-action="Index" method="get">
  <p>
    <select asp-for="Predmet" asp-items="Model.Predmetes">
      <option value="">Усі предмети</option>
    </select>
    В
    Пошук: <input type="text" asp-for="SearchString" />
    <input type="submit" value="Знайти" />
  </p>
</form>
```

Рисунок 34 Розмітка тега форма у файлі Index

Тег HTML <form> використовує допоміжну функцію тега Form, щоб при відправленні форми рядок фільтра передавався в дію Index контролера Bases.

Тепер переходимо для додавання пошуку по предмету для цього ми вже додали рядок пошуку. Тепер додамо клас для здійснення пошуку по предмету Predmets.cs в Models.

```

using Microsoft.AspNetCore.Mvc.Rendering;

namespace Basematerial.Models
{
    4 references
    public class Predmets
    {
        6 references
        public List<Base>? Bases { get; set; }
        2 references
        public SelectList? Predmetes { get; set; }
        1 reference
        public string? Predmet { get; set; }
        1 reference
        public string? SearchString { get; set; }
    }
}

```

Рисунок 35 Клас Predmet.cs

Модель предметів міститиме:

- список предметів
- об'єкт SelectList зі списком предметів.
- у цьому списку користувач може вибрати предмет.
- об'єкт Predmet, що містить вибраний предмет.
- SearchString містить текст, який користувачі вводять у поле пошуку.

Також тепер змінимо метод Index у файлі BasesController для того щоб мати можливість здійснювати пошук за предметами.

```

public async Task<IActionResult> Index(string Predmet, string searchString)
{
    if (_context.Base == null)
    {
        return Problem("Entity set 'Basematerial.Context' is null.");
    }

    IQueryable<string> genreQuery = from m in _context.Base
                                    orderby m.Predmet
                                    select m.Predmet;
    var bases = from m in _context.Base
                select m;

    if (!string.IsNullOrEmpty(searchString))
    {
        bases = bases.Where(s => s.Title!.Contains(searchString));
    }
    if (!string.IsNullOrEmpty(Predmet))
    {
        bases = bases.Where(x => x.Predmet == Predmet);
    }
    var PredmetVM = new Predmets
    {
        Predmetes = new SelectList(await genreQuery.Distinct().ToListAsync()),
        Bases = await bases.ToListAsync()
    };

    return View(PredmetVM);
}

```

Рисунок 36 Змінений метод Index для пошуку по предметам.

Ось це код що міститься в файлі BasesController метода Index визначає запит LINQ, який отримує всі предмети з бази даних.

```
IQueryable<string> predmetQuery = from m in _context.Base  
orderby m.Predmet  
select m.Predmet;
```

Об'єкт SelectList зі списком предмет створюється шляхом проектування окремих предметів (це необхідно, щоб виключити предмети, що повторюються).

Коли користувач шукає елемент, значення пошуку зберігається в полі пошуку.

ВИСНОВКИ

У даній магістерській роботі було розглянуто сучасні пошукові системи, які використовуються для пошуку великої кількості інформації у мережі інтернет . У першому розділі було розглянуто методи пошуку інформації в мережі Інтернет з використанням пошукових систем, таких як система Google, яка обробляє понад 60 тисяч пошукових запитів за секунду.

Роль посилань стала основною для механізму ранжування Google. Хоча алгоритм PageRank змінювався і, здавалося б, втрачав свою значущість, посилання завжди були і, швидше за все, залишаться одним з головних факторів ранжирування. У 2016 році аналітики Google вказали, що два визначальних фактори - це контент і посилання, а в 2020 Джон Мюллер підтвердив, що PageRank досі використовується при розподілі позицій.

Можна бути впевненими: робота над посилальним профілем та грамотна внутрішня перелінковка залишаються пріоритетними SEO-завданнями. Регулярно перевіряйте, щоб структура вашого сайту дозволяла легко переходити між сторінками, та займайтеся лінкбیلдингом, фокусуючись на авторитетних та релевантних вашому сайту ресурсах.

У другому розділі на було розглянуто створення пошуково інформаційного ресурсу навчального призначення В умовах зростання потоків інформації та розвитку інформаційних-комунікаційних та технологій віртуальної реальності , і вирішення завдань впровадження різних ресурсів, на яких може міститися інформації та забезпечення цілісності збереження інфомрації та покращення механізмів вдосконалення механізмів набуває актуальності та значущості.

Єдиний інформаційний простір, сформований з урахуванням специфічних особливостей предметних галузей педагогіки, визначається як системоутворююча основа при підготовці, плануванні та реалізації моделей інтеграції. Розглянута система пошукового впровадження інформаційних ресурсів навчального призначення, яка побудована на принципах впровадження

різних ресурсів, надає можливість покращити освітню діяльність користувачів з урахуванням за рахунок використання і оновлення інформації.

У третьому розділі описано розроблене програмне забезпечення для створення пошуково-інформаційної системи бази даних матеріалів для ліцею.

Описані методи які були використані для написання програмного забезпечення. Деякі з методів були розглянуті детальніше. Також було організовано пошук в цій базі даних матеріалів. З використанням двох видів пошуку по предметам та по темам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zong Woo Geem. Recent Advances in Harmony Search Algorithm Studies in Computation Intelligence URL
https://link.springer.com/chapter/10.1007/978-3-642-04317-8_6
2. Schütze, Hinrich, Christopher D. Manning, Raghavan, Prabhakar: Introduction to information retrieval, Prabhakar// Cambridge, UK: Cambridge University Press. URL <https://nlp.stanford.edu/IR-book/information-retrieval-book.html>
3. Автоматизовані інформаційно-пошукові мови URL
https://stud.com.ua/34387/informatika/informatsiyno_poshukova_mova
4. Алгоритм пошуку URL <https://seranking.com/ru/blog/pagerank/>
5. The real impact of Google's RankBrain on search traffic. URL
<https://www.startupgrind.com/blog/the-real-impact-of-googles-rankbrain-on-search-traffic/>
6. The Anatomy of a Large-Scale Hypertextual Web Search Engine. URL
<https://snap.stanford.edu/class/cs224w-readings/Brin98Anatomy.pdf>
7. Google's Tensor Processing Unit could advance Moore's Law. URL:
<https://www.pcworld.com/article/3072256/googles-tensor-processingunit-said-to-advance-moores-law-seven-years-into-the-future.html>.
8. Google: RankBrain. URL
<https://searchengineland.com/library/google/google-rankbrain>
9. What are malicious websites?. URL <https://us.norton.com/internetsecurity-malware-what-are-malicious-websites.html>
10. Best Search Engines in The World. URL
<https://www.inspire.scot/blog/2016/11/11/top-12-best-searchengines-in-the-world238>
11. What is Google Team Drive.- URL
<https://www.systoolsgroup.com/google-drive/team-drive.html>.
12. 8 major Google algorithm updates, explained URL
<https://searchengineland.com/8-major-googlealgorithm-updates-explained-282627>.

13. Философский словарь. Под ред. И. Т. Фролова. М: Из-во Современник. 2009 г. 846 с.
- 14 Белов В.В., Терехов А.А., Чистякова В.И. Повышение пертинентности поиска в современных информационных средах. М: НТИ «Горячая линия – Телеком», 2012. 158 с
- 15 Ладыженский Г. Интеграция приложений такая, как она есть.
<http://citforum.ru/gazeta/50/>,
- 16 Минашкин С. А. Математическое и программное обеспечение интеллектуальных поисковых систем на основе использования мультиагентной архитектуры. Автореферат диссертации.
- 17 . Татьянушкин Д. В. Технология обработки информации студентами в высшей школе: этапы, методы, приёмы. Вестник Волжского университета им. В. Н. Татищева № 3[13] / 2013, С. 110–117.
18. Пузанкова Е.Н., Бочкова Н.В. Современная педагогическая интеграция, ее характеристики режим доступа Электронный, научный информационноаналитический журнал «Образование и общество»
19. . Використання Web API URL
https://professorweb.ru/my/ASP_NET/mvc/level8/8_2.php
20. Путь ASP.NET Core. <https://habr.com/ru/post/312226/>.
21. Entity Framework Core documentation.: URL
<https://docs.microsoft.com/en-us/ef/core/>
22. Введение в Entity Framework Core. URL
<https://metanit.com/sharp/entityframeworkcore/1.1.php>. –
23. Microsoft Visual Studio URL [https://open-](https://open-file.ru/programs/microsoftvisual-studio)
[file.ru/programs/microsoftvisual-studio](https://open-file.ru/programs/microsoftvisual-studio).
24. Руководство по ASP.NET MVC 5 URL <https://metanit.com/sharp/mvc5>.
25. Entity Framework 6 URL: <https://docs.microsoft.com/ru-ru/ef/ef6>
26. SQL <https://www.w3schools.com/sql/>
- 27 Entity Framework 7 URL: <https://docs.microsoft.com/ru-ru/ef/ef7>
28. <https://dotnet.microsoft.com/en-us/apps/aspnet>

29. Створення бази даних ASP. NET

<https://metanit.com/sharp/mvc5/5.2.php>

30. Entity framework <https://habr.com/ru/post/282844/>

ДОДАТОК ЛІСТИНГ ПРОГРАМИ

BASE controller

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using Basematerial.Data;
using Basematerial.Models;

namespace Basematerial.Controllers
{
    public class BasesController : Controller
    {
        private readonly BasematerialContext _context;

        public BasesController(BasematerialContext context)
        {
            _context = context;
        }

        // GET: Bases
        public async Task<IActionResult> Index(string Predmet, string searchString)
        {
            if (_context.Base == null)
            {
                return Problem("Entity set 'Basematerial.Context' is null.");
            }

            IQueryable<string> predmetQuery = from m in _context.Base
                                              orderby m.Predmet
                                              select m.Predmet;
            var bases = from m in _context.Base
                       select m;

            if (!string.IsNullOrEmpty(searchString))
            {
                bases = bases.Where(s => s.Title!.Contains(searchString));
            }
        }
    }
}
```

```

        if (!string.IsNullOrEmpty(Predmet))
        {
            bases = bases.Where(x => x.Predmet == Predmet);
        }
        var PredmetVM = new Predmets
        {
            Predmetes = new SelectList(await
predmetQuery.Distinct().ToListAsync()),
            Bases = await bases.ToListAsync()
        };

        return View(PredmetVM);
    }
    // GET: Bases/Details/5
    public async Task<IActionResult> Details(int? id)
    {
        if (id == null || _context.Base == null)
        {
            return NotFound();
        }

        var @base = await _context.Base
            .FirstOrDefaultAsync(m => m.Id == id);
        if (@base == null)
        {
            return NotFound();
        }

        return View(@base);
    }

    // GET: Bases/Create
    public IActionResult Create()
    {
        return View();
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
Create([Bind("Id,Title,Datepublic,Predmet,Class")] Base @base)
    {
        if (ModelState.IsValid)

```

```

    {
        _context.Add(@base);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(@base);
}

// GET: Bases/Edit/5
public async Task<IActionResult> Edit(int? id)
{
    if (id == null || _context.Base == null)
    {
        return NotFound();
    }

    var @base = await _context.Base.FindAsync(id);
    if (@base == null)
    {
        return NotFound();
    }
    return View(@base);
}

// POST: Bases/Edit/5

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id,
[Bind("Id,Title,Datepublic,Predmet,Class")] Base @base)
{
    if (id != @base.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(@base);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)

```

```

    {
        if (!BaseExists(@base.Id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }
    return RedirectToAction(nameof(Index));
}
return View(@base);
}

// GET: Bases/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    if (id == null || _context.Base == null)
    {
        return NotFound();
    }

    var @base = await _context.Base
        .FirstOrDefaultAsync(m => m.Id == id);
    if (@base == null)
    {
        return NotFound();
    }

    return View(@base);
}

// POST: Bases/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    if (_context.Base == null)
    {
        return Problem("Entity set 'BasematerialContext.Base' is null.");
    }
    var @base = await _context.Base.FindAsync(id);
    if (@base != null)

```

```

    {
        _context.Base.Remove(@base);
    }

    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}

private bool BaseExists(int id)
{
    return (_context.Base?.Any(e => e.Id == id)).GetValueOrDefault();
}
}
}

```

Create

@model Basematerial.Models.Base

```

@{
    ViewData["Title"] = "Create";
}

```

<h1>Create</h1>

<h4>Base</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Create">

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

<div class="form-group">

<label asp-for="Title" class="control-label"></label>

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

</div>

<div class="form-group">

<label asp-for="Datepublic" class="control-label"></label>

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

</div>

<div class="form-group">

<label asp-for="Predmet" class="control-label"></label>

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

```

        <span asp-validation-for="Predmet" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Class" class="control-label"></label>
        <input asp-for="Class" class="form-control" />
        <span asp-validation-for="Class" class="text-danger"></span>
    </div>
    <div class="form-group">
        <input type="submit" value="Create" class="btn btn-primary" />
    </div>
</form>
</div>
</div>

<div>
    <a asp-action="Index">Back to List</a>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```



```
Edit
@model Basematerial.Models.Base
```

```
@{
    ViewData["Title"] = "Edit";
}
```

```
<h1>Edit</h1>
```

```
<h4>Base</h4>
```

```
<hr />
```

```
<div class="row">
```

```
    <div class="col-md-4">
```

```
        <form asp-action="Edit">
```

```
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
```

```
            <input type="hidden" asp-for="Id" />
```

```
            <div class="form-group">
```

```
                <label asp-for="Title" class="control-label"></label>
```

```
                <input asp-for="Title" class="form-control" />
```

```
                <span asp-validation-for="Title" class="text-danger"></span>
```

```
            </div>
```

```
            <div class="form-group">
```

```
                <label asp-for="Datepublic" class="control-label"></label>
```

```
                <input asp-for="Datepublic" class="form-control" />
```

```
                <span asp-validation-for="Datepublic" class="text-danger"></span>
```

```
            </div>
```

```
            <div class="form-group">
```

```
                <label asp-for="Predmet" class="control-label"></label>
```

```
                <input asp-for="Predmet" class="form-control" />
```

```
                <span asp-validation-for="Predmet" class="text-danger"></span>
```

```
            </div>
```

```
            <div class="form-group">
```

```
                <label asp-for="Class" class="control-label"></label>
```

```
                <input asp-for="Class" class="form-control" />
```

```
                <span asp-validation-for="Class" class="text-danger"></span>
```

```
            </div>
```

```
            <div class="form-group">
```

```
                <input type="submit" value="Save" class="btn btn-primary" />
```

```
            </div>
```

```
        </form>
```

```
    </div>
```

```
</div>
```

```
<div>
```

```
<a asp-action="Index">Back to List</a>  
</div>
```

```
@section Scripts {  
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}  
}
```

Details

@model Basematerial.Models.Base

```
@{
    ViewData["Title"] = "Details";
}
```

```
<h1>Details</h1>
```

```
<div>
```

```
    <h4>Base</h4>
```

```
    <hr />
```

```
    <dl class="row">
```

```
        <dt class="col-sm-2">
```

```
            @Html.DisplayNameFor(model => model.Title)
```

```
        </dt>
```

```
        <dd class="col-sm-10">
```

```
            @Html.DisplayFor(model => model.Title)
```

```
        </dd>
```

```
        <dt class="col-sm-2">
```

```
            @Html.DisplayNameFor(model => model.Datepublic)
```

```
        </dt>
```

```
        <dd class="col-sm-10">
```

```
            @Html.DisplayFor(model => model.Datepublic)
```

```
        </dd>
```

```
        <dt class="col-sm-2">
```

```
            @Html.DisplayNameFor(model => model.Predmet)
```

```
        </dt>
```

```
        <dd class="col-sm-10">
```

```
            @Html.DisplayFor(model => model.Predmet)
```

```
        </dd>
```

```
        <dt class="col-sm-2">
```

```
            @Html.DisplayNameFor(model => model.Class)
```

```
        </dt>
```

```
        <dd class="col-sm-10">
```

```
            @Html.DisplayFor(model => model.Class)
```

```
        </dd>
```

```
    </dl>
```

```
</div>
```

```
<div>
```

```
    <a asp-action="Edit" asp-route-id="@Model?.Id">Edit</a> |
```

```
    <a asp-action="Index">Back to List</a>
```

```
</div>
```

Delete

@model Basematerial.Models.Base

```
@{
    ViewData["Title"] = "Delete";
}
```

<h1>Delete</h1>

<h3>Are you sure you want to delete this?</h3>

<div>

<h4>Base</h4>

<hr />

<dl class="row">

<dt class="col-sm-2">

@Html.DisplayNameFor(model => model.Title)

</dt>

<dd class="col-sm-10">

@Html.DisplayFor(model => model.Title)

</dd>

<dt class="col-sm-2">

@Html.DisplayNameFor(model => model.Datepublic)

</dt>

<dd class="col-sm-10">

@Html.DisplayFor(model => model.Datepublic)

</dd>

<dt class="col-sm-2">

@Html.DisplayNameFor(model => model.Predmet)

</dt>

<dd class="col-sm-10">

@Html.DisplayFor(model => model.Predmet)

</dd>

<dt class="col-sm-2">

@Html.DisplayNameFor(model => model.Class)

</dt>

<dd class="col-sm-10">

@Html.DisplayFor(model => model.Class)

</dd>

</dl>

<form asp-action="Delete">

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

<input type="submit" value="Delete" class="btn btn-danger" /> |

<a asp-action="Index">Back to List

```

</form>
</div>

Index.cshtml\
@model Basematerial.Models.Predmets

@{
    ViewData["Title"] = "Матеріали";
}

<h1>База матеріалів</h1>

<p>
    <a asp-action="Create">Додати матеріал</a>
</p>
<<form asp-controller="Bases" asp-action="Index" method="get">
    <p>

        <select asp-for="Predmet" asp-items="Model.Predmetes">
            <option value="">Усі предмети</option>
        </select>
        В
        Пошук: <input type="text" asp-for="SearchString" />
        <input type="submit" value="Знайти" />
    </p>
</form>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Bases![0].Title)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Bases![0].Datepublic)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Bases![0].Predmet)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Bases![0].Class)
            </th>
            <th></th>
        </tr>
    </thead>

```

```

<tbody>
  @foreach (var item in Model.Bases!)
  {
    <tr>
      <td>
        @Html.DisplayFor(modelItem => item.Title)
      </td>
      <td>
        @Html.DisplayFor(modelItem => item.Datepublic)
      </td>
      <td>
        @Html.DisplayFor(modelItem => item.Predmet)
      </td>
      <td>
        @Html.DisplayFor(modelItem => item.Class)
      </td>
      <td>
        <a asp-action="Edit" asp-route-id="@item.Id">Редагувати</a> |
        <a asp-action="Details" asp-route-id="@item.Id">Детальніше</a> |
        <a asp-action="Delete" asp-route-id="@item.Id">Видалити</a>
      </td>
    </tr>
  }
</tbody>
</table>

```

Base.cs

```

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Basematerial.Models
{
    public class Base
    {
        public int Id { get; set; }
        public string? Title { get; set; }
        [Display(Name = "Date public")]
        [DataType(DataType.Date)]
        public DateTime Datepublic { get; set; } //realse date
        public string? Predmet { get; set; } //genre

        public int Class { get; set; } //price
    }
}

```

```

    }
}
Home.cs

```

```

using Basematerial.Models;
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;

namespace Basematerial.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        public IActionResult Index()
        {
            return View();
        }

        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 });
        }
    }
}

```

Base context

```

using System;
using System.Collections.Generic;
using System.Linq;

```

```

using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using Basematerial.Models;

namespace Basematerial.Data
{
    public class BasematerialContext : DbContext
    {
        public BasematerialContext (DbContextOptions<BasematerialContext> options)
            : base(options)
        {
        }

        public DbSet<Basematerial.Models.Base> Base { get; set; } = default!;
    }
}

```

Initial create

```

using System;
using Microsoft.EntityFrameworkCore.Migrations;

```

#nullable disable

```

namespace Basematerial.Migrations
{
    /// <inheritdoc />
    public partial class InitialCreate : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Base",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    Title = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    Datepublic = table.Column<DateTime>(type: "datetime2", nullable:
false),
                    Predmet = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    Class = table.Column<decimal>(type: "int", nullable: false)
                },
                constraints: table =>

```



```
        {
            table.PrimaryKey("PK_Base", x => x.Id);
        });
    }

    /// <inheritdoc />
    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "Base");
    }
}
```