Міністерство освіти і науки України
Національний університет «Одеська політехніка»

Навчально-науковий
інститут комп'ютерних систем

КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни «Сучасні мови програмування»
для здобувачів першого (бакалаврського) рівня вищої освіти
за спеціальністю
124 Системний аналіз
(частина 2)

Одеса – 2025

Міністерство освіти і науки України
Національний університет «Одеська політехніка»

Навчально-науковий
інститут комп'ютерних систем

КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни «Сучасні мови програмування»
для здобувачів першого (бакалаврського) рівня вищої освіти
за спеціальністю
124 Системний аналіз
(частина 2)

Затверджено
на засіданні кафедри прикладної математики та інформаційних технологій
Протокол № _ від _____

Одеса – 2025

Конспект лекцій з дисципліни «Сучасні мови програмування» для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 124 – Системний аналіз (освітньо-професійна програма: «Системний аналіз та наука про дані») (частина 2) / Укл.: К. О. Трифонова. − Одеса, 2025. − 29 с.

Укладач: Трифонова К. О., ст. викл.

ЗМІСТ

Вступ

Навчальна дисципліна «Сучасні мови програмування» є необхідною дисципліною для підвищення рівня теоретичних і прикладних знань, що формують фахівця в галузі інформаційних технологій, які сприяють утворенню у здобувачів поглиблених вмінь та навичок проектування та розробки системи управління доступом для веб-застосувань із застосуванням сучасних парадигм програмування, тобто підготовці спеціалістів для створення, впровадження та підтримки професійно-орієнтованих комп'ютерних технологій у професійній діяльності.

Дисципліна «Сучасні мови програмування» відповідає освітньо-професійній програмі, навчальному плану підготовки фахівців першого (бакалаврського) рівня вищої освіти спеціальності 124 – Системний аналіз і є складовою циклу дисциплін професійної підготовки обов'язкової частини навчального плану.

Дисципліну викладають впродовж сьомого семестру першого (бакалаврського) рівня. У процесі навчання передбачено лекції, лабораторні заняття.

Згідно навчального плану передбачено підсумковий контроль у вигляді заліку.

Робочу програму навчальної дисципліни укладено згідно з вимогами кредитно-модульної системи організації навчального процесу. Програма визначає обсяги компетентностей, які повинен опанувати здобувач відповідно до освітньо-професійної програми, алгоритму вивчення навчального матеріалу дисципліни «Сучасні мови програмування», необхідне методичне забезпечення, складові та технологію оцінювання навчальних досягнень.

Предмет навчальної дисципліни «Сучасні мови програмування» – сучасні технології проектування та розробки системи управління доступом для веб-застосувань.

Мета вивчення навчальної дисципліни «Сучасні мови програмування» – забезпечити формування поглибленої системи теоретичних і практичних знань у галузі проектування та розробки системи управління доступом для веб-застосувань, пов'язаною з наданням можливості санкціонованого та попередженням несанкціонованого доступу до даних, для реалізації здатності використання інформаційно-комунікаційних технологій, сучасних методів і моделей інформаційної безпеки та/або кібербезпеки.

Завдання вивчення дисципліни:

– формування теоретичних основ та практичних навичок для проектування та розробки системи автентифікації, перевірки справжності користувача шляхом порівняння за допомогою певної унікальної інформації для веб-застосувань;

– формування теоретичних основ та практичних навичок для проектування та розробки системи авторизації, перевірки та визначення повноважень на виконання певних дій для веб-застосувань;

– формування теоретичних основ та практичних навичок для проектування та розробки системи конфіденційності, гарантування захисту важливих даних, що передаються між клієнтом та сервером або зберігаються для веб-застосувань;

– формування теоретичних основ та практичних навичок для проектування та розробки системи цілісності, гарантування внаслідок неавторизованого втручання не змінності важливих даних, що передаються між клієнтом та сервером для веб-застосувань.

Стратегічні цілі дисципліни – націлити майбутніх фахівців на творче застосування отриманих знань у подальшій професійній підготовці та їх наступній практичній діяльності.

Лекція 9. Робота з відношеннями

1. Доступ до зв'язаних даних безпосередньо
   1.1. Підвищення зв'язаних даних
   1.2. Доступ до зв'язаних даних із використанням параметра типу
2. Укомплектування відношення між даними
   2.1. Запитування зв'язаних даних у відношенні «один до багатьох»
3. Робота зі зв'язаними даними у відношенні «один до багатьох»
   3.1. Оновлення зв'язаних об'єктів
   3.2. Створення нових зв'язаних об'єктів
   3.3. Зміна відношень

У цьому розділі буде показано, яким чином отримувати доступ до зв'язаних даних безпосередньо і як потім укомплектувати відношення, щоб можна було переміщатися в обох напрямках між об'єктами у відношенні. Розділ сконцентрований на відношеннях «один до багатьох», які найлегше всього створювати, але які вимагають уваги при виконанні запитів чи оновленні БД. У наступному розділі розглядаються інші типи відношень між даними, які підтримує інфраструктура Entity Framework Core.

1. Доступ до зв'язаних даних безпосередньо

У більшості застосувань, принаймні, певні зв'язані дані мають власний життєвий цикл і робочі потоки, які користувачі повинні виконувати. Наприклад, у застосуванні DataApp легко представити, що адміністратору може знадобитися внести зміну, що відображає зміну адреси. Зробити це складно, оскільки отримувати доступ до даних у БД можна, тільки починаючи з об'єкта Product і слідуючи за його навігаційними властивостями. Скажімо, якщо потрібно оновити об'єкт ContactLocation, то доведеться почати з пошуку об'єкта Product, який зв'язаний з об'єктом Supplier, що підлягає модифікації, запросити знайдений об'єкт Product і пройти за навігаційними властивостями для отримання об'єктів Supplier та ContactDetails. Лише тоді з'явиться можливість доступу до об'єкту ContactLocation, який потрібно модифікувати.

Для того щоб уникнути проблеми такого роду можна отримувати доступ до зв'язаних даних безпосередньо. У наступних розділах будуть представлені два підходи для доступу до зв'язаних даних разом з поясненнями, коли кожен із них має використовуватися.

1.1. Підвищення зв'язаних даних

Для даних, які відіграють важливу роль у застосуванні, на кшталт тих, що мають власні інструменти управління та життєвий цикл, найкращий підхід передбачає їх підвищення, щоб до них можна було звертатися через властивість DbSet<T>, визначену в класі контекста.

Це найбільш руйнівний підхід, оскільки він вимагає створення та застосування нової міграції, але він ставить зв'язані дані в першокласне становище в рамках застосування та дотримується угод, до яких звикло багато розробників.

1.2. Доступ до зв'язаних даних із використанням параметра типу

Альтернативою підвищенню даних є застосування набору методів, які надаються класом контексту БД та дозволяють вказувати тип даних як параметр типу. Вчиняти так корисно при роботі з даними, для яких потрібний не регулярний або обмежений доступ через специфічні операції і підвищення до властивості DbSet<T> не має підстав. У таблиці описані методи DbContext, які приймають параметр типу і можуть використовуватися для доступу до даних без необхідності в їх підвищенні.

Таблиця. Методи DbContext із параметрами типу

Ім'я	Опис
`Set<T>()`	Повертає об'єкт DbSet, який застосовується для запитування БД
`Find<T>(key)`	Запитує у БД об'єкт типу T, який має вказаний ключ
`Add<T>(newObject)`	Додає до БД новий об'єкт типу T
`Update<T>(changedObject)`	Оновлює об'єкт типу T
`Remove<T>(dataObject)`	Видаляє з БД об'єкт типу T

Перевага методів, наведених у таблиці, полягає в тому, що вони можуть використовуватися для створення узагальненого сховища, за допомогою якого можна надати доступ до специфічного типу, коли сховище конфігуровано в якості служби у файлі Program.cs.

2. Укомплектування відношення між даними

Підвищення зв'язаних даних полегшило доступ до них, але роботу з ними можна продовжити вдосконалювати. На даний момент відношення між класами відбуваються лише в одному напрямку. Наприклад, можна почати з об'єкта Product і пройти за навігаційною властивістю, щоб отримати зв'язані дані Supplier, але почати з об'єкта Supplier і рухатися в протилежному напрямку не можна. Для усунення зазначеного недоліку інфраструктура Entity Framework Core дозволяє визначати властивості, які уможливлюють навігацію в іншому напрямку, що відомо як укомплектування відношення.

Існує ряд різних типів відношень, які можна створювати між класами, але коли вперше визначається навігаційна властивість, інфраструктура Entity Framework Core передбачає, що потрібно створити відношення «один до багатьох», і навігаційна властивість додається до класу, що знаходиться на стороні «багато» відношення.

Укомплектування відношення «один до багатьох» виконується легко і вимагає додавання навігаційної властивості до класу, що знаходиться на боці «один» відношення. Така властивість називається зворотною.

2.1. Запитування зв'язаних даних у відношенні «один до багатьох»

Після укомплектування відношення можна починати з об'єкта будь-якого типу у відношенні та рухатися в обох напрямках. У прикладі застосування це означає, що можна почати з об'єкта Supplier і пройти за властивістю Products, щоб переміститься на набір зв'язаних об'єктів Product за умови включення зв'язаних даних у запит. Включити зв'язані дані в укомплектованому відношенні «один до багатьох» можна декількома способами, які пояснюються у наступних розділах.

Запитування всіх зв'язаних даних

Якщо необхідний повний набір зв'язаних даних, тоді можна застосувати метод Include() або ThenInclude() для розширення запиту шляхом вибору навігаційної властивості.

Запитування з використанням явного завантаження

Застосування методу Include() для слідування по навігаційній властивості у відношенні «один до багатьох» не пропонує якогось способу фільтрації зв'язаних даних, оскільки всі вони вилучаються з БД. Якщо потрібна більша вибірковість, тоді інфраструктура Entity Framework Core надає два прийоми фільтрації даних, що запитуються. Перший прийом називається явним завантаженням.

Запитування із використанням засобу виправлення

Інфраструктура Entity Framework Core підтримує засіб, що називається виправленням (fixing up), коли дані, витягнуті з об'єкта контексту БД, кешуються та застосовуються для заповнення навігаційних властивостей об'єктів, що створюються у наступних запитах. При обережному використанні цей засіб дозволяє створювати складні запити, які отримують зв'язані дані більш ефективно, ніж явне завантаження або слідування за навігаційною властивістю із застосуванням методу Include().

3. Робота зі зв'язаними даними у відношенні «один до багатьох»

Однією з найважливіших характеристик укомплектованих відношень є те, що операції можуть виконуватися з використанням навігаційних властивостей на обох сторонах відношення.

3.1. Оновлення зв'язаних об'єктів

У ASP.NET MVC з Entity Framework Core редагування зв’язаних об’єктів здійснюється через навігаційні властивості, передаючи з форми дані як для головної сутності, так і для пов’язаних з нею об’єктів. Для цього використовується спеціальне представлення, яке формує коректну іменовану колекцію елементів input, що дозволяє зв’язувачу моделей MVC відновити об’єкт Supplier разом із колекцією Product. Після надсилання форми MVC автоматично створює нові екземпляри об’єктів, але Entity Framework Core не відстежує їхній стан, тому всі властивості сутностей повинні бути явно передані. Отриманий об’єкт з повністю заповненими даними передається в репозиторій, де виконується оновлення пов’язаних записів у базі даних.

3.2. Створення нових зв'язаних об'єктів

Створення нових зв’язаних об’єктів у Entity Framework Core може виконуватися через ту саму навігаційну властивість, що й оновлення існуючих даних. Якщо під час обробки форми EF Core отримує об’єкти Product з нульовим значенням Id, він автоматично додає їх до бази даних як нові та пов’язує з відповідним об’єктом Supplier. Важливо передавати у формі повну колекцію пов’язаних об’єктів, оскільки EF Core вважає її повним набором і може розірвати зв’язки з відсутніми елементами. Таким чином, додавання нового Product реалізується як оновлення Supplier, що дозволяє зберегти цілісність зв’язків і не потребує окремої операції створення в контролері.

3.3. Зміна відношень

Зміна відношень між зв’язаними об’єктами в Entity Framework Core є складнішою операцією, ніж створення чи оновлення, особливо для обов’язкових зв’язків, оскільки база даних контролює цілісність даних. Для цього користувачеві надається форма, яка дозволяє вибрати нового постачальника для кожного товару, при цьому всі незмінені дані передаються прихованими полями. Об’єкти, створені зв’язувачем моделей MVC, не відстежуються EF Core, тому їх не можна напряму використовувати для оновлення БД. Замість цього зміни визначаються логічно, після чого переносяться на сутності, завантажені та відстежувані Entity Framework Core, що й забезпечує коректне оновлення відношень між Product і Supplier.

Лекція 10. Типи відношень

1. Укомплектування відношення «один до одного»
   1.1. Визначення навігаційної властивості
   1.2. Вибір залежного сутнісного класу
   1.3. Створення та застосування міграції
2. Робота з відношеннями «один до одного»

Відношення «один до багатьох», описане в попередньому розділі, не є єдиним типом відношень, що підтримуються інфраструктурою Entity Framework Core. У цьому розділі буде показано, як визначати відношення «один до одного» і «багато до багатьох», продемонстровано, яким чином запитувати зв'язані дані при використанні таких відношень, і пояснено, як ними управляти в БД.

1. Укомплектування відношення «один до одного»

У відношенні «один до одного» об'єкт одного типу пов'язаний з єдиним об'єктом іншого типу. Для завершення відношення «один до одного» потрібні два кроки, які пояснюються в наступних розділах.

1.1. Визначення навігаційної властивості

Щоб Entity Framework Core створив відношення «один до одного», потрібно визначити обидві навігаційні властивості в пов’язаних класах. Якщо навігаційна властивість задана лише з одного боку, EF Core автоматично вважає, що це відношення «один до багатьох».

1.2. Вибір залежного сутнісного класу

Під час налаштування відношення «один до одного» в Entity Framework Core необхідно визначити, яка сутність є залежною, тобто в таблиці якої зберігатиметься зовнішній ключ. Залежною вважається та сутність, що містить стовпець зовнішнього ключа, тоді як інша є головною.

На відміну від зв’язку «один до багатьох», де залежна сторона визначається автоматично, у відношенні one-to-one це рішення потрібно прийняти явно, додавши властивість зовнішнього ключа. Тип цієї властивості визначає характер зв’язку: не-nullable тип створює обов’язкове відношення, а nullable тип – необов’язкове.

1.3. Створення та застосування міграції

Після визначення навігаційних властивостей і зовнішнього ключа модель даних готова до створення міграції. Під час її генерації Entity Framework Core аналізує зміни в моделі та формує набір операцій для оновлення структури бази даних. Якщо тип відношення змінюється, міграція переносить зовнішній ключ у таблицю залежної сутності та коригує існуючі обмеження. Для відношення «один до одного» додатково створюється унікальний індекс на стовпці зовнішнього ключа, що забезпечує унікальність зв’язку та гарантує, що кожному об’єкту може відповідати не більше одного пов’язаного об’єкта.

2. Робота з відношеннями «один до одного»

Після того, як відношення укомплектовано, виконання операцій над зв'язаними даними стає прямолінійним, як описано в наступних розділах.

2.1. Запитування зв'язаних даних у відношенні «один до одного»

Запитування даних «один до одного». Для завантаження пов’язаних об’єктів використовують метод Include(), який дозволяє отримати разом основну та залежну сутність. Запит можна починати з будь-якої сторони відношення, а отримані об’єкти передаються до представлення для зручного відображення.

Оновлення бази даних

Після зміни моделі старі дані можуть не відповідати новій схемі. Під час розробки простіше видалити базу та застосувати міграції заново. При запуску застосування база заповнюється початковими даними, і контролер відображає їх у браузері.

2.2. Створення та оновлення зв'язаних об'єктів

У відношенні «один до одного» створення та оновлення даних може виконуватися через будь-яку з навігаційних властивостей, оскільки обидві сутності взаємно пов’язані. Дані для створення або редагування передаються з форми до контролера, де на основі значення ідентифікатора визначається, чи потрібно створити новий об’єкт, чи оновити існуючий. Якщо ідентифікатор має нульове значення, Entity Framework Core додає новий об’єкт до контексту, інакше — оновлює вже наявний. При цьому приховані поля форми відіграють ключову роль, оскільки вони забезпечують передачу ідентифікаторів пов’язаних об’єктів, що дозволяє EF Core коректно пройти за навігаційними властивостями та зберегти зміни у базі даних.

2.3. Зміна відношення «один до одного»

При зміні відношень між об'єктами повинна дотримуватися обережність, особливо у разі обов'язкових відношень. Інфраструктура Entity Framework Core не нав'язує обмеження перед відправкою оновлень БД, тому легко виконати оновлення, яке порушить посилальну цілісність БД, що призводить до помилки. У наступних розділах буде показано, як оновлювати обов'язкові та необов'язкові відношення, починаючи з існуючого обов'язкового відношення, яке вже налаштовано в БД для прикладу застосування.

Зміна обов’язкового відношення «один до одного»

У обов’язковому відношенні залежна сутність завжди має бути пов’язана з головною. Зміна зв’язку ускладнена, бо головна сутність може бути вільною або вже асоційованою з іншою залежною. Якщо головна сутність вільна, оновлюють зовнішній ключ залежної сутності. Якщо зайнята – створюють тимчасовий об’єкт, щоб уникнути одночасного зв’язку двох залежних сутностей із однією головною. Це гарантує цілісність даних і дотримання обмежень БД.

Зміна необов’язкового відношення «один до одного»

У необов’язковому відношенні залежна сутність може існувати без зв’язку. Зовнішній ключ роблять nullable, що дозволяє просте редагування: старий зв’язок автоматично розривається, а новий встановлюється без використання тимчасових об’єктів. Це спрощує підтримку і зміну відношень у БД.

3. Визначення відношень «багато до багатьох»

В Entity Framework Core можна визначати та керувати відношеннями «багато до багатьох», коли об’єкти одного типу можуть бути пов’язані з множиною об’єктів іншого типу і навпаки.

Такі відношення дозволяють моделювати складні взаємозв’язки між сутностями, які не обмежуються «один до одного» або «один до багатьох». Для реалізації таких зв’язків EF Core створює проміжну таблицю, яка зберігає асоціації між об’єктами обох типів, забезпечуючи можливість ефективного запитування та підтримки цілісності даних.

3.1. Створення з’єднуючого класу

У Entity Framework Core відношення «багато до багатьох» реалізується через проміжний (з’єднуючий) клас. Такий клас містить властивості зовнішніх ключів і навігаційні властивості для обох сутностей, що беруть участь у відношенні. По суті, він створює два відношення «один до багатьох», об’єднаних у єдину проміжну сутність, яка зберігає асоціації між об’єктами обох типів і забезпечує підтримку цілісності даних.

3.2. Укомплектування відношення «багато до багатьох»

У відношенні «багато до багатьох» класи-учасники отримують навігаційні властивості до з’єднуючого класу. Це дозволяє комбінувати дві асоціації «один до багатьох» у єдину проміжну сутність, яка зберігає зв’язки між об’єктами. Через такі навігаційні властивості можна опосередковано переходити від одного об’єкта до множини пов’язаних об’єктів іншого класу. У базі даних проміжна сутність створює таблицю з зовнішніми ключами для обох класів, забезпечуючи відображення відношення «багато до багатьох» і підтримку цілісності даних.

3.3. Підготовка проекту

Для підготовки проекту створюють початкові дані, які додаються до бази даних через метод Seed(). Нові об’єкти Shipment додаються окремо та не мають відношень до інших сутностей, що дозволяє спершу наповнити БД ізольованими даними. Метод ClearData() забезпечує видалення цих об’єктів за потреби. Початкові дані застосовуються тільки для порожньої бази, тому після внесення змін у модель або міграції рекомендується видаляти та відтворювати БД, щоб забезпечити відповідність структури та даних усім поточним міграціям.

3.4. Запитування зв'язаних даних у відношенні «багато до багатьох»

У відношеннях «багато до багатьох» прямого зв’язку між сутностями немає, тому запити проходять через з’єднуючий клас. Для отримання пов’язаних даних використовуються навігаційні властивості та Include()/ThenInclude(). Представлення через LINQ, наприклад Select() і string.Join(), формує списки пов’язаних елементів, забезпечуючи коректне відображення зв’язків.

3.5. Управління відношеннями «багато до багатьох»

Управління відношеннями «багато до багатьох» здійснюється через з’єднуючий клас із зовнішніми ключами обох сутностей. Контролер завантажує головну сутність разом із пов’язаними елементами через Include(). Представлення відображає всі об’єкти та позначає пов’язані. Після вибору користувачем колекція з’єднуючого класу замінюється новою, і SaveChanges() автоматично додає нові зв’язки та видаляє непотрібні, підтримуючи актуальні відношення в БД.

Лекція 11. Формування шаблонів для існуючих баз даних

1. Підготовка проекту
   1.1. Існуюча база даних

Приклади в попередніх розділах починалися з визначення класів C#, які описували модель і застосовувалися для створення БД, що відомо як розробка в стилі «спочатку код». У проектах, де необхідно використовувати існуючу БД, потрібен інший підхід, який називається розробкою у стилі «спочатку БД». У цьому розділі буде показано, як використовувати засіб формування шаблонів (scaffolding) інфраструктури Entity Framework Core, який перевіряє БД і автоматично генерує модель даних. Згаданий засіб найкраще підходить для простих БД, тоді як складніші проекти ефективніше обслуговуються ручним моделюванням даних, яке розглядається в наступному розділі.

1. Підготовка проекту

Матеріал розділу спирається на БД, яка створюється без застосування Entity Framework Core з метою імітації існуючої БД. Для створення та заповнення БД будуть використовуватись засоби виконання SQL-запитів середовища Visual Studio, як пояснюється в наступних розділах.

1.1. Існуюча база даних

База даних містить центральну таблицю з основними даними, яка пов’язана з іншими таблицями різними типами відношень. Частина відношень реалізована як «багато до багатьох» через сполучну таблицю, частина – як «один до багатьох», а деякі – як «один до одного». Така структура демонструє основні принципи моделювання даних і взаємодії між таблицями в Entity Framework Core.

1.2. Підключення до сервера баз даних

Необхідно запустити Visual Studio, не відкриваючи та не створюючи проект. Вибрати пункт меню Tools-SQL Server-New Query та ввести (localdb)\MSSQLLocalDB у полі Server Name.

Необхідно переконатися, що в полі Authentication вибрано варіант Windows Authentication, а в полі Database Name – варіант <default>, і натиснути на кнопці Connect. Середовище Visual Studio відкриє вікно редактора запитів, у якому можна вводити та виконувати SQL-оператори.

1.3. Створення бази даних

Базу даних створюють або очищують стару версію, потім таблиці створюють у правильному порядку: спочатку незалежні, потім з відношенням «один до багатьох» і «один до одного», а «багато до багатьох» через сполучну таблицю. Після цього таблиці заповнюють даними і

перевіряють запитами SELECT, щоб гарантувати правильну структуру та зв’язки для роботи з EF Core.

1.4. Створення проекту ASP.NET Core MVC

Для роботи з існуючою БД у ASP.NET Core MVC створюють порожній проект, підключають пакети EF Core (ядро, SQL Server, Design, Tools), налаштовують у Program.cs служби MVC, маршрутизацію та middleware для статичних файлів. Додають простий контролер і представлення для тесту, а для стилізації HTML підключають Bootstrap через wwwroot. Це створює базову інфраструктуру для подальшої роботи з даними через EF Core.

1.5. Тестування прикладу застосування

Щоб встановити фіксований порт для запуску проекту, у файлі Properties/launchSettings.json редагують властивість applicationUrl, вказавши потрібний порт (наприклад, 5000). Це дозволяє завжди відкривати застосунок за одним і тим же URL.

Після зміни налаштувань запуск через dotnet run відкриває застосунок у браузері за зазначеним портом, що спрощує тестування та демонстрацію функціональності.

2. Формування шаблонів для існуючої бази даних

Найпростіший спосіб роботи з існуючою БД – використовувати засіб формування шаблонів EF Core, який аналізує базу, створює класи контексту та моделі для роботи з даними. Навіть при ручному створенні моделі цей інструмент корисний для перевірки коректності структури БД та подальшого використання у ASP.NET Core MVC.

2.1. Виконання процесу формування шаблонів

Для роботи з існуючою базою даних у ASP.NET Core MVC використовується scaffolding через команду dotnet ef dbcontext scaffold. Вона створює класи моделі та клас контексту на основі структури БД.

Класи моделі відповідають таблицям у БД, містять властивості для стовпців та навігаційні властивості для відносин між таблицями.

Клас контексту містить DbSet<T> для кожної таблиці, перевизначає методи OnConfiguring() (для підключення до БД) та OnModelCreating() (для налаштування відносин і ключів).

Scaffolding дозволяє швидко створити C#-модель для існуючої БД, зберігаючи відносини та структуру таблиць, щоб одразу працювати з даними через Entity Framework Core без ручного написання класів.

2.2. Використання моделі даних, згенерованої процесом формування шаблонів, в ASP.NET Core MVC

Після створення модель даних можна застосовувати в ASP.NET Core MVC, після виконання лише кількох змін, описаних у наступних розділах.

Для роботи ASP.NET Core MVC з існуючою БД рядок підключення переноситься в appsettings.json, а метод OnConfiguring() у класі контексту видаляється. Контекст реєструється у DI-контейнері через AddDbContext, що дозволяє впроваджувати його в контролери. Дані отримуються через DbSet і методи Include()/ThenInclude(), а у представленнях відображаються за допомогою Razor та LINQ. Це забезпечує доступ до зв’язаних таблиць без зміни структури БД і повну інтеграцію з MVC.

3. Реагування на зміни у базі даних

Якщо виконується використання існуючої БД не одноосібно, тоді може виникнути необхідність у реагуванні на зміни, внесені в інтересах інших застосувань. У наступних розділах буде з імітовано зміну в БД і показано, як оновити модель даних, згенеровану процесом формування шаблонів, щоб врахувати зміну в застосуванні.

3.1. Модифікація бази даних

Для емуляції змін у БД додається нова таблиця та стовпець зовнішнього ключа в існуючу таблицю. SQL-код створює таблицю, заповнює її даними, додає стовпець у таблицю товарів і встановлює зв'язок через зовнішній ключ. Після оновлення таблиця відображає значення нового стовпця. Однак така зміна призводить до проблем у ASP.NET Core MVC, оскільки модель даних ще не містить нового стовпця, і запити чи додавання нових об'єктів стають некоректними.

3.2. Оновлення моделі даних

Оновлення моделі даних виконується повторним формуванням шаблонів із додаванням аргументів --force для заміни існуючих класів і --no-build для пропуску збірки проекту. Це дозволяє синхронізувати модель з новими змінами в БД, не блокуючи процес через помилки існуючої кодифікації.

3.3. Оновлення класу контексту

Після повторного формування шаблонів клас контексту перезаписується, тому для роботи з ASP.NET Core MVC потрібно знову закоментувати OnConfiguring(), залишивши DbSet і OnModelCreating() для відображення таблиць та їхніх зв'язків.

3.4. Оновлення контролерів та представлень

Після внесення змін у БД потрібно оновити контролер Home, щоб включити нову навігаційну властивість Fitting, та додати відповідну колонку у представлення Index.cshtml. Контролер використовує Include(s => s.Fitting) для підвантаження зв'язаних даних, а представлення відображає значення @s.Fitting.Name. Після запуску застосування (dotnet run) таблиця з оновленими даними буде доступна за http://localhost:5000.

4. Додавання можливостей сталості моделі даних

Щоб зберегти додаткову логіку в моделі даних після оновлень БД, використовують часткові класи (partial). Наприклад, у папці Models/Logic створюють частковий клас Shoe з властивістю PriceIncTax, яка обчислює ціну з податком. Представлення Index.cshtml використовує цю властивість замість Price. Після запуску застосування (dotnet run) на http://localhost:5000 відображаються ціни з урахуванням податку.

Лекція 12. Ручне моделювання баз даних

1. Створення ручної моделі даних
   - 1.1. Створення класу контексту та сутнісних класів
   - 1.2. Створення контролера та представлення
   - 1.3. Основні угоди для моделі даних
   - 1.4. Перевизначення угод для моделі даних
   - 1.5. Моделювання відношень
   - 1.6. Завершення моделі даних
2. Використання моделі даних, створеної вручну

2.1. Запитування даних у моделі даних, створеної вручну
2.2. Оновлення даних у моделі даних, створеної вручну

У попередньому розділі використовувався процес формування шаблонів для автоматичного створення моделі даних на основі існуючої БД. Процес формування шаблонів зручний, але він не пропонує особливо багато можливостей щодо точного контролю, а результат може виявитися незручним у застосуванні. У поданому розділі буде показано, як моделювати БД вручну. Такий процес вимагає більшого обсягу зусиль, але виробляє модель даних, з якою природніше працювати і якою легше управляти, коли у БД вносяться зміни.

1. Створення ручної моделі даних

Перш ніж приступати до моделювання існуючої БД, необхідно зрозуміти її схему і з'ясувати, які частини БД потрібні для застосування ASP.NET Core MVC. У відсутності докладного опису БД і її структури хорошою відправною точкою може бути виконання процесу формування шаблонів, розглянутого в попередньому розділі, навіть якщо створена в результаті нього модель даних буде застосовуватися лише у довідкових цілях при побудові ручної моделі.

1.1. Створення класу контексту та сутнісних класів

Для роботи з Entity Framework створюють клас контексту, який наслідує DbContext, містить конструктор із параметром налаштувань підключення та властивості для доступу до колекцій сутностей.

Кожна сутність визначається класом із властивостями, які відображають стовпці таблиці в базі даних.

Щоб контекст можна було використовувати в додатку, його реєструють у службах залежностей із вказанням рядка підключення до бази даних.

1.2. Створення контролера та представлення

Щоб протестувати модель даних, створюють контролер, який отримує контекст через конструктор і передає колекції сутностей до методу дії. Метод дії повертає ці дані стандартному представленню для відображення.

Представлення формує таблицю або інший інтерфейс, де відображаються властивості сутностей. Після цього застосунок запускають, і дані можна переглянути через веб-браузер за відповідною URL-адресою.

1.3. Основні угоди для моделі даних

Коли модель даних створюється вручну, Entity Framework Core використовує певні угоди, які спрощують роботу з базою даних. Властивості класу контексту зазвичай відповідають таблицям у базі даних, а властивості сутнісних класів – стовпцям цих таблиць. Первинний ключ автоматично визначається за властивістю з іменем Id або <НазваКласу>Id. Завдяки цим угодам відображення між об’єктами та базою даних може працювати без додаткової конфігурації, навіть для вже існуючої бази.

1.4. Перевизначення угод для моделі даних

У випадках, коли структура бази даних не збігається з тим, що очікує застосунок, стандартні угоди Entity Framework Core можна перевизначити двома способами: через атрибути або Fluent API. Атрибути застосовуються безпосередньо до сутнісних класів і властивостей для вказівки таблиці, стовпців або первинного ключа, що робить їх зручними для простих змін. Fluent API

надає більший контроль, дозволяючи програмно описувати відображення класів на таблиці та властивостей на стовпці, навіть якщо імена класів чи властивостей не відповідають структурі бази.

Після налаштування моделі даних через атрибути або Fluent API контекст та сутнісні класи можна використовувати звичайним чином у контролерах і представленнях. Дані передаються до представлень через моделі або допоміжні об'єкти, як-от ViewBag, і відображаються у вигляді таблиць чи інших інтерфейсних елементів. Це дозволяє інтегрувати існуючу базу даних у застосунок без необхідності змінювати структуру бази.

1.5. Моделювання відношень

Щоб моделювати відношення між таблицями в Entity Framework Core, у сутнісних класах додають навігаційні властивості та зовнішні ключі, що описують зв'язки між об'єктами, наприклад «один до багатьох». Для явного вказання ключів і зворотної сторони відношення використовують атрибути, а для більш тонкого контролю структури — Fluent API в методі OnModelCreating(). Це дозволяє повністю описати відношення без зміни класів і забезпечує правильне відображення на базу даних та коректну роботу запитів.

1.6. Завершення моделі даних

Для завершення моделі даних додають сутнісні класи для нових таблиць та описують відношення між ними. Клас для кампаній продажів має навігаційну властивість до взуття у відношенні «один до одного», а категорії пов'язані зі взуттям через з'єднувальний клас у відношенні «багато до багатьох». У класі контексту створюють відповідні властивості DbSet для нових сутностей, що дозволяє отримувати доступ до даних і підтримувати зв'язки між таблицями без додаткової конфігурації, якщо угоди Entity Framework Core дотримані.

2. Використання моделі даних, створеної вручну

Модель даних завершена, і дані можна використовувати в частині ASP.NET Core MVC застосування, дозволивши інфраструктурі Entity Framework Core обробляти їх відображення на БД, коли угоди не дотримувалися. У наступних розділах буде демонструватися процес запитування та оновлення даних.

2.1. Запитування даних у моделі даних, створеної вручну

Щоб відобразити всі дані моделі, запит у контролері розширюють із використанням Include та ThenInclude для навігаційних властивостей, а додаткові сутності передають у представлення через ViewBag. У представленні створюють таблиці для кожного типу даних та відображають пов'язані об'єкти, включаючи відношення «один до одного» і «багато до багатьох». Результат демонструє, що ручна модель даних працює так само, як згенерована або створена для міграцій, а Entity Framework Core автоматично забезпечує правильне відображення класів і властивостей на таблиці та стовпці бази даних.

2.2. Оновлення даних у моделі даних, створеної вручну

Дія редагування реалізується так само, як і для моделей, створених автоматично або через міграції: спочатку отримується об'єкт з усіма пов'язаними даними для відображення в формі, а потім у методі оновлення приймається об'єкт, створений зв'язувачем моделей MVC, разом із даними про вибрані та попередні зв'язки. Старі непотрібні зв'язки видаляються, нові створюються, а існуючі залишаються, після чого оновлюється основний об'єкт і зберігаються

зміни в базі. Це дозволяє одночасно редагувати об’єкт і його відношення «багато до багатьох» без конфліктів відстеження змін.

Створення часткових представлень

Для редагування об’єктів створюються часткові представлення, кожне з яких відповідає за один аспект даних. Одне часткове представлення містить поля для редагування властивостей основного об’єкта, інші відображають елементи вибору пов’язаних об’єктів у відношеннях «один до одного» або «багато до багатьох». Для зв’язків «багато до багатьох» використовуються приховані поля та прапорці, які дозволяють зв’язувачу моделей правильно створювати або видаляти з’єднуючі записи без додаткових запитів до бази. Такий підхід забезпечує гнучке редагування об’єкта разом із усіма його відношеннями через окремі, повторно використовувані часткові представлення.

Створення представлення редактора

Для створення повноцінного редактора об’єкта формується основне представлення, яке об’єднує всі часткові представлення. Кожне часткове відповідає за окрему групу даних: властивості основного об’єкта, вибір пов’язаних об’єктів у відношеннях «один до одного» або «багато до багатьох». Основне представлення містить форму з кнопками збереження та скасування, а часткові вставки дозволяють відображати й редагувати всі зв’язки одночасно. Такий підхід забезпечує користувачу єдиний інтерфейс для редагування об’єкта разом із його пов’язаними даними та гарантує коректне оновлення в базі даних після відправлення форми.

Лекція 13. Введення до системи ідентифікації

1. Налаштування ASP.NET Core Identity
    1.1. Створення класу користувача
    1.2. Створення класу контексту бази даних
    1.3. Конфігурація налаштування рядка підключення до бази даних
    1.4. Створення бази даних Identity
2. Використання ASP.NET Core Identity
    2.1. Перелік облікових записів користувача
    2.2. Створення користувачів
    2.3. Перевірка паролів
    2.4. Перевірка деталей, пов'язаних із користувачем
3. Завершення побудови засобів адміністрування
    3.1. Реалізація можливості видалення
    3.2. Реалізація можливості редагування

Сучасні вебзастосунки потребують надійних і гнучких механізмів керування користувачами, які забезпечують автентифікацію, авторизацію та безпечне зберігання облікових даних. Платформа ASP.NET Core надає вбудовану систему Identity, призначену для централізованого управління користувацькими обліковими записами.

ASP.NET Core Identity реалізує набір сервісів і компонентів проміжного програмного забезпечення, які дозволяють працювати з користувачами через єдиний програмний інтерфейс. Система тісно інтегрується з Entity Framework Core, що забезпечує збереження та обробку даних у реляційних базах даних.

У цьому розділі розглядаються основні принципи налаштування та використання ASP.NET Core Identity, зокрема створення класу користувача, конфігурація контексту бази даних, підключення служб Identity до застосунку та початкове налаштування політик перевірки облікових даних.

1. Налаштування ASP.NET Core Identity

Налаштування ASP.NET Core Identity охоплює підготовку інфраструктури застосунку до використання механізмів керування обліковими записами та збереженням пов’язаних даних. Процес починається з визначення конфігурації доступу до бази даних, яка завантажується під час запуску застосунку та використовується для ініціалізації контексту даних. На основі цього контексту створюються структури сховища, необхідні для функціонування системи автентифікації та авторизації.

Подальший етап полягає у підключенні служб Identity, які забезпечують обробку даних користувачів, управління ролями та виконання операцій, пов’язаних із маркерами безпеки. Система інтегрується у конвеєр обробки запитів, що дозволяє пов’язувати облікові дані з HTTP-запитами. Завершальним кроком є підготовка бази даних через механізм міграцій, що формує початкову схему та забезпечує можливість подальшого використання компонентів Identity у застосунку.

1.1. Створення класу користувача

Клас користувача призначений для подання облікового запису користувача в застосунку та забезпечує взаємодію з системою ідентифікації. Він містить базові дані, необхідні для автентифікації та керування користувачами.

Клас користувача створюється на основі стандартної реалізації та може бути розширений додатковими властивостями відповідно до вимог застосунку. Це дозволяє зберігати як обов’язкову інформацію, так і специфічні дані користувача.

1.2. Створення класу контексту бази даних

Клас контексту бази даних відповідає за взаємодію застосунку з базою даних та керування доступом до даних користувачів. Він визначає зв’язок між моделями застосунку та фізичним сховищем даних.

Такий клас створюється на основі стандартного контексту та налаштовується для роботи з системою ідентифікації. Контекст забезпечує збереження, отримання та оновлення інформації, а також є основою для формування структури бази даних.

1.3. Конфігурація налаштування рядка підключення до бази даних

Рядок підключення визначає параметри доступу застосунку до бази даних і є необхідним для встановлення з’єднання з системою зберігання даних. Він містить інформацію про сервер, назву бази даних та спосіб автентифікації.

Налаштування рядка підключення зазвичай виконується у файлі конфігурації застосунку та завантажується під час запуску. Це забезпечує централізоване керування параметрами підключення та спрощує зміну конфігурації без модифікації програмного коду.

1.4. Створення бази даних Identity

База даних Identity призначена для зберігання інформації про користувачів, їхні облікові записи та пов’язані з ними дані безпеки. Вона формується на основі визначених моделей та налаштованого контексту бази даних.

Створення бази даних виконується за допомогою механізму міграцій, який автоматично генерує структуру таблиць і забезпечує узгодженість схеми даних із поточним станом застосунку. Це дозволяє керувати змінами структури бази даних у процесі розвитку системи.

2. Використання ASP.NET Core Identity

Використання ASP.NET Core Identity забезпечує інтеграцію механізмів керування користувачами в структуру веб-застосунку. Система надає інструменти для створення, перегляду та модифікації облікових записів, зберігаючи їх у базі даних через узгоджену інфраструктуру доступу до даних. Робота з обліковими записами здійснюється через спеціалізовані служби, що відповідають за виконання типових операцій, включно з перевіркою достовірності введених даних, генерацією хешованих паролів та застосуванням політик безпеки.

Система також підтримує централізоване адміністрування, яке дозволяє виконувати перегляд і коригування інформації про користувачів. Використання стандартного API спрощує розробку інструментів керування, забезпечуючи узгодженість виконання операцій та покладаючись на перевірені механізми автентифікації. Таким чином, Identity є основою для побудови функцій, пов’язаних з обробкою облікових даних, та інтегрується в загальну логіку управління користувачами в застосунку.

2.1. Перелік облікових записів користувача

Перелік облікових записів користувача забезпечує отримання та відображення інформації про всіх зареєстрованих користувачів у системі. Для цього використовується доступ до сховища даних Identity через відповідні сервіси керування користувачами.

Формування переліку дозволяє централізовано переглядати основні дані облікових записів і є основою для подальших операцій адміністрування, таких як створення, редагування або видалення користувачів.

2.2. Створення користувачів

Створення користувачів у системі автентифікації передбачає використання спеціалізованого механізму, що забезпечує формування та збереження облікових записів у відповідному сховищі даних. Процес ґрунтується на застосуванні сервісів, які відповідають за створення, перевірку та обробку інформації про користувачів, включно з їхніми основними атрибутами та параметрами безпеки.

Під час формування нового облікового запису система виконує перевірку коректності введених даних, дотримання політик безпеки та цілісності облікової інформації. Після успішної валідації створена сутність зберігається у базі даних через відповідний контекст доступу до даних. Механізм передбачає гнучке налаштування вимог до облікового запису та можливість розширення структури даних користувача відповідно до потреб застосунку.

2.3. Перевірка паролів

Перевірка паролів є складовою політики безпеки системи автентифікації та забезпечує контроль відповідності введеного пароля встановленим вимогам. Під час створення або зміни облікового запису пароль проходить автоматизовану валідацію, яка включає оцінювання його довжини, складності та наявності певних символів відповідно до визначених параметрів.

Система допускає гнучке конфігурування правил перевірки, що дає змогу враховувати організаційні чи галузеві стандарти. Додатково може бути реалізована спеціалізована логіка перевірки, яка розширює базові механізми або замінює їх для задоволення унікальних вимог. Результатом процесу є визначення успішності перевірки або формування повідомлень про невідповідність політиці безпеки.

2.4. Перевірка деталей, пов'язаних із користувачем

Перевірка деталей, пов’язаних із користувачем, спрямована на забезпечення коректності та допустимості облікових даних, що зберігаються для конкретного облікового запису. До таких

даних належать ідентифікатори, контактна інформація та інші атрибути, визначені політикою системи. Механізми валідації контролюють унікальність та формат цих значень, а також їх відповідність встановленим обмеженням.

Система дозволяє конфігурувати критерії перевірки, зокрема щодо дозволених символів, вимог до унікальності або доменних обмежень. За потреби можуть застосовуватися додаткові, спеціалізовані правила, що розширюють базову логіку. Результати валідації визначають можливість створення, оновлення або відхилення облікових даних користувача.

3. Завершення побудови засобів адміністрування

Завершення побудови засобів адміністрування включає впорядкування всіх елементів, необхідних для повноцінного керування обліковими записами користувачів у межах застосунку. На цьому етапі узгоджуються механізми отримання, редагування та видалення даних, які інтегруються через відповідні служби системи Identity. Контролер адміністратора поєднує ці функції, забезпечуючи централізований доступ до операцій керування обліковими записами.

У процесі завершення реалізації конфігурація перевірки користувацьких даних і паролів узгоджується зі встановленими політиками, що гарантує коректну обробку введених значень. Інтерфейс редагування та перегляду облікових записів доповнюється обробкою можливих помилок і перевірок, забезпечуючи узгодженість з логікою збереження даних. У результаті формується закінчений інструментарій адміністрування, який використовує можливості Identity для надійного та структурованого управління користувачами.

3.1. Реалізація можливості видалення

Реалізація можливості видалення передбачає забезпечення механізму, який дозволяє вилучати облікові записи користувачів із системи автентифікації. Цей процес виконується за допомогою спеціалізованого інтерфейсу керування, що отримує ідентифікатор користувача, здійснює пошук відповідного запису та ініціює операцію видалення.

Під час виконання операції система перевіряє наявність облікового запису та обробляє результат, формуючи відповідне повідомлення про успішність або невдачу дії. Механізм видалення інтегрується з інфраструктурою доступу до даних та використовує узгоджені засоби обробки помилок, що забезпечує узгодженість і цілісність даних у сховищі.

3.2. Реалізація можливості редагування

Реалізація можливості редагування передбачає запровадження механізму, що забезпечує зміну даних облікового запису користувача в межах системи автентифікації. Такий механізм отримує поточні значення атрибутів користувача, дозволяє оновлювати їх відповідно до встановлених правил і здійснює перевірку коректності змінених даних.

У процесі редагування система виконує валідацію оновлених параметрів, що включає перевірку відповідності політикам безпеки та вимогам до облікових записів. Після успішної перевірки зміни зберігаються у сховищі даних через узгоджений інтерфейс доступу. У разі виявлення невідповідностей система формує повідомлення про помилки, забезпечуючи контроль цілісності та достовірності облікової інформації.

Лекція 14. Застосування системи ідентифікації

1. Аутентифікація користувачів
   1.1. Підготовка до реалізації аутентифікації
   1.2. Додавання аутентифікації користувачів
   1.3. Тестування аутентифікації
2. Авторизація користувачів за допомогою ролей

2.1. Створення та видалення ролей
2.2. Управління членством у ролях
2.3. Використання ролей для авторизації
3. Приміщення до бази даних початкової інформації

Даний розділ присвячений детальному розгляду основних підходів до забезпечення безпеки вебзастосунку з використанням механізмів аутентифікації та авторизації. У ньому висвітлюються принципи підтвердження особи користувача, організації процесу входу в систему та формування захищеного контексту, який використовується під час подальшої взаємодії користувача із застосунком.

Значна увага приділяється ролям як ключовому інструменту авторизації, що дозволяє реалізувати гнучке розмежування прав доступу до ресурсів і функціональних можливостей системи. Розглядаються підходи до створення, керування та використання ролей, а також принципи контролю доступу на їх основі. Окремо описуються підготовчі етапи, пов'язані з налаштуванням інфраструктури безпеки, ініціалізацією початкових даних у базі даних та перевіркою коректності роботи механізмів аутентифікації й авторизації.

Розглянуті в розділі теоретичні положення відповідають сучасним підходам до побудови систем безпеки вебзастосунків і формують цілісне уявлення про організацію надійної, масштабованої та керованої моделі доступу в застосунках на платформі ASP.NET Core.

1. Аутентифікація користувачів

Аутентифікація користувачів є одним із ключових елементів системи безпеки вебзастосунку та відіграє визначальну роль у процесі захисту інформаційних ресурсів. Її основним призначенням є підтвердження особи користувача під час звернення до системи шляхом перевірки наданих облікових даних і встановлення факту, що запит дійсно надходить від зареєстрованого та відомого системі користувача.

У ході аутентифікації відбувається створення захищеного контексту користувача, який асоціюється з поточним сеансом роботи та використовується під час обробки наступних запитів. Збереження стану автентифікації між окремими запитами дозволяє системі ідентифікувати користувача без необхідності повторного введення облікових даних, що підвищує зручність використання застосунку та забезпечує безперервність роботи.

Реалізація механізму аутентифікації є фундаментом для подальшого контролю доступу до функціональних можливостей вебзастосунку. Вона сприяє підвищенню рівня інформаційної безпеки, запобігає несанкціонованому доступу до ресурсів системи та створює необхідні умови для ефективної авторизації користувачів відповідно до їхніх прав і ролей.

1.1. Підготовка до реалізації аутентифікації

Підготовка до реалізації аутентифікації передбачає налаштування інфраструктури вебзастосунку для перевірки та підтвердження особи користувача під час обробки HTTP-запитів. Основою цього процесу є інтеграція механізмів керування ідентичністю, які забезпечують зберігання облікових даних, перевірку автентичності та формування контексту безпеки для кожного запиту.

На етапі підготовки визначається спосіб обмеження доступу до ресурсів застосунку, зокрема через використання декларативних механізмів контролю доступу, що дозволяють розрізняти автентифіковані та неавтентифіковані запити. Також налаштовується взаємодія між проміжним програмним забезпеченням та компонентами прикладного рівня, відповідальними за обробку результатів аутентифікації.

Важливим аспектом є конфігурація маршруту входу, на який перенаправляються користувачі у разі відсутності підтвердженої автентифікації, а також підготовка моделей даних для отримання та валідації облікових даних. У межах підготовчого етапу формується структура

контролерів і подань, необхідних для збору інформації користувача та подальшої обробки результатів входу.

Загалом підготовка до реалізації аутентифікації створює базу для безпечної і керованої ідентифікації користувачів, забезпечуючи коректну інтеграцію механізмів безпеки в життєвий цикл запитів вебзастосунку.

1.2. Додавання аутентифікації користувачів

Додавання аутентифікації користувачів передбачає впровадження комплексу механізмів, спрямованих на перевірку облікових даних та підтвердження факту успішного входу користувача в систему. На цьому етапі організовується взаємодія між даними, введеними користувачем, і системою керування ідентичністю, яка виконує їх зіставлення з інформацією, збереженою у базі даних, та приймає рішення щодо можливості доступу.

Процес аутентифікації охоплює отримання облікових даних, їх перевірку на коректність і повноту, а також аналіз результатів валідації. У разі успішного проходження перевірки формується захищений контекст користувача, що асоціюється з поточним сеансом роботи. Цей контекст зберігається між окремими запитами та використовується для ідентифікації користувача під час подальшої взаємодії із застосунком.

Важливою складовою додавання аутентифікації є автоматичне керування станом автентифікації, яке реалізується за допомогою механізмів сеансів або спеціальних маркерів безпеки. Такий підхід дозволяє системі коректно розпізнавати автентифіковані запити без необхідності повторного введення облікових даних, забезпечуючи зручність використання та безперервність роботи.

Отже, реалізація аутентифікації користувачів забезпечує ефективний контроль доступу до функціональності застосунку, підвищує загальний рівень інформаційної безпеки та створює надійну основу для подальшої авторизації і розмежування прав доступу між різними категоріями користувачів.

1.3. Тестування аутентифікації

Тестування аутентифікації спрямоване на перевірку коректності роботи механізмів входу користувачів у систему та обробки результатів автентифікації. На цьому етапі оцінюється, чи правильно система реагує на запити автентифікованих і неавтентифікованих користувачів, а також чи забезпечується обмеження доступу до захищених ресурсів.

У процесі тестування перевіряється правильність перенаправлення користувачів до сторінки входу у разі відсутності підтвердженої автентифікації та коректне встановлення стану автентифікованого користувача після успішного входу. Окрему увагу приділяють обробці помилкових облікових даних і відображенню відповідних повідомлень.

Також тестування дозволяє переконатися, що стан автентифікації зберігається між запитами та коректно завершується під час виходу користувача із системи. Загалом цей етап підтверджує надійність і стабільність реалізованого механізму аутентифікації перед його подальшим використанням.

2. Авторизація користувачів за допомогою ролей

Авторизація користувачів за допомогою ролей є механізмом контролю доступу, що ґрунтується на визначенні наборів дозволів для окремих груп користувачів. Роль виступає логічним посередником між користувачем і доступними йому ресурсами, дозволяючи абстрагувати правила доступу від конкретних облікових записів.

У процесі авторизації система аналізує ролі, закріплені за автентифікованим користувачем, та порівнює їх із вимогами доступу до певних функцій або розділів застосунку. Це забезпечує

гнучке й масштабоване керування правами, оскільки зміна доступу виконується через зміну ролей, а не програмного коду.

Використання ролей для авторизації підвищує безпеку та спрощує адміністрування, забезпечуючи чітке розмежування повноважень і централізований контроль доступу в межах системи.

2.1. Створення та видалення ролей

Створення та видалення ролей є складовою системи керування доступом і забезпечує логічне групування користувачів відповідно до їхніх повноважень. Роль виступає абстрактним рівнем доступу, який використовується для подальшого обмеження або надання прав на виконання окремих дій у застосунку.

Під час створення ролей система зберігає їх у відповідному сховищі ідентичностей, забезпечуючи унікальність та коректність ідентифікаторів. Видалення ролей передбачає їх вилучення з системи разом із припиненням асоціацій із користувачами, що гарантує актуальність і цілісність моделі безпеки.

Реалізація операцій створення та видалення ролей дозволяє гнучко адаптувати структуру доступу до змін вимог застосунку та є основою для подальшого керування авторизацією.

2.2. Управління членством у ролях

Управління членством у ролях є важливою складовою системи авторизації та полягає у призначенні користувачів до відповідних ролей або їх вилученні з цих ролей з метою ефективного контролю прав доступу. Такий підхід дозволяє гнучко керувати дозволами користувачів без необхідності змінювати внутрішню структуру або бізнес-логіку вебзастосунку.

Під час керування членством система здійснює аналіз поточного складу ролей і надає можливість змінювати його відповідно до визначених правил безпеки та організаційних вимог. Усі зміни членства безпосередньо впливають на доступ користувачів до функціональних можливостей застосунку та реалізуються в межах наявних механізмів ідентифікації й авторизації, що забезпечує їхню узгодженість і коректність.

Отже, управління членством у ролях забезпечує централізований і керований контроль прав доступу, сприяє підтримці актуальної моделі розмежування повноважень і підвищує загальний рівень безпеки вебзастосунку.

2.3. Використання ролей для авторизації

Використання ролей для авторизації передбачає обмеження доступу до ресурсів і функціональності застосунку на основі належності користувача до певної ролі. Такий підхід дозволяє відокремити логіку перевірки прав доступу від бізнес-логіки та забезпечити централізоване керування дозволами.

Під час авторизації система аналізує ролі, пов’язані з автентифікованим користувачем, і приймає рішення щодо надання або заборони доступу до конкретних дій чи розділів застосунку. Ролі виступають узагальненими рівнями доступу, що спрощує масштабування та супровід системи безпеки.

Застосування ролей для авторизації підвищує керованість і надійність контролю доступу, дозволяючи легко змінювати права користувачів шляхом оновлення їх членства у ролях без внесення змін до програмного коду.

3. Приміщення до бази даних початкової інформації

Приміщення до бази даних початкової інформації є важливим етапом підготовки системи безпеки вебзастосунку до повноцінної експлуатації. Воно полягає в автоматичному створенні

базового набору даних, необхідних для коректної роботи механізмів аутентифікації та авторизації одразу після розгортання застосунку. Такий підхід дозволяє забезпечити наявність ключових облікових записів і ролей ще до початку взаємодії кінцевих користувачів із системою.

До складу початкових даних, як правило, входять стандартні ролі та облікові записи з підвищеними правами доступу, які використовуються для виконання адміністративних операцій і подальшого налаштування системи безпеки. Процес ініціалізації здійснюється під час запуску застосунку автоматично та не потребує ручного втручання адміністратора, що зменшує ймовірність помилок і спрощує первинне налаштування.

Загалом приміщення початкової інформації до бази даних забезпечує безперервність роботи застосунку, спрощує процес його розгортання та гарантує готовність системи аутентифікації й авторизації до використання з першого запуску. Це створює надійну основу для подальшого адміністрування та безпечної експлуатації вебзастосунку.

Лекція 15. Розширені можливості системи ідентифікації

1. Додавання спеціальних властивостей до класу користувача
   1.1. Підготовка міграції бази даних
   1.2. Тестування спеціальних властивостей
2. Робота із заявками та політиками
   2.1. Поняття заявок
   2.2. Створення заявок
   2.3. Використання політик
   2.4. Використання політик для авторизації доступу до ресурсів
3. Використання сторонньої автентифікації
   3.1. Реєстрація програми в Google
   3.2. Увімкнення аутентифікації Google

У сучасних вебзастосунках система керування користувачами відіграє ключову роль у забезпеченні автентифікації, авторизації та збереження персоналізованих даних. Базові механізми ідентифікації надають стандартний набір властивостей користувача, однак на практиці цього часто недостатньо для повноцінної роботи прикладної системи.

Розширені засоби ідентифікації дозволяють адаптувати модель користувача до потреб конкретного застосунку шляхом зберігання додаткової інформації та інтеграції її в існуючу інфраструктуру безпеки. Це забезпечує гнучкість, масштабованість і можливість подальшого розвитку системи, не порушуючи цілісності даних і механізмів доступу.

У цьому розділі розглядаються підходи до розширення моделі користувача, що дозволяють зберігати спеціальні властивості та використовувати їх у процесі роботи застосунку.

1. Додавання спеціальних властивостей до класу користувача

У більшості прикладних систем виникає потреба зберігати додаткову інформацію про користувачів, яка виходить за межі стандартних облікових даних. Для цього механізм керування користувачами дозволяє розширювати базову модель користувача шляхом додавання спеціальних властивостей. Такі властивості можуть описувати індивідуальні характеристики, налаштування або інші дані, що мають зберігатися між сеансами роботи.

Оскільки система ідентифікації використовує об’єктно-реляційне відображення для роботи з базою даних, додавання нових властивостей до моделі користувача потребує відповідного оновлення схеми бази даних. Це забезпечує коректне збереження та відновлення розширених даних без порушення вже існуючої інформації про користувачів.

Після розширення моделі користувача оновлення значень спеціальних властивостей виконується через стандартні механізми керування обліковими записами. При цьому відповідальність за фіксацію змін у сховищі даних покладається на компоненти керування

користувачами. Такий підхід дозволяє інтегрувати додаткові дані в єдину систему автентифікації та авторизації, зберігаючи цілісність і розширюваність моделі користувача.

1.1. Підготовка міграції бази даних

Підготовка міграції бази даних полягає у приведенні структури сховища даних у відповідність до змін у моделі предметної області. На цьому етапі аналізуються нові або модифіковані атрибути сутностей, що потребують збереження, та визначається необхідність оновлення схеми бази даних без втрати наявної інформації.

Процес підготовки передбачає тимчасове вимкнення механізмів автоматичної ініціалізації або початкового заповнення даних, щоб уникнути конфліктів зі зміненою структурою. Далі формується опис змін схеми у вигляді міграції, яка містить узгоджений набір операцій для додавання, зміни або видалення елементів структури бази даних.

Коректно підготовлена міграція забезпечує контрольоване та відтворюване оновлення схеми, підтримує цілісність даних і створює основу для подальшого розвитку системи. Особливу увагу приділяють перевірці безпечності змін, особливо у випадку роботи з даними користувачів, щоб мінімізувати ризики втрати або пошкодження інформації.

1.2. Тестування спеціальних властивостей

Тестування спеціальних властивостей спрямоване на перевірку коректності їх збереження, обробки та відображення після внесення змін до моделі даних і структури бази даних. На цьому етапі оцінюється, чи правильно нові властивості інтегровані в логіку застосунку та чи узгоджуються вони з механізмами аутентифікації й авторизації.

Процес тестування передбачає перевірку доступності спеціальних властивостей для читання та оновлення, а також їхньої стійкості під час повторних сеансів роботи. Окрема увага приділяється правильному оновленню даних у базі та відсутності побічних ефектів для вже наявних записів.

Результатом тестування має бути підтвердження того, що спеціальні властивості працюють стабільно, не порушують цілісність системи та можуть безпечно використовуватися в подальшій логіці застосунку.

2. Робота із заявками та політиками

Робота із заявками та політиками спрямована на реалізацію гнучкої та розширюваної моделі авторизації користувачів. Заявки використовуються для представлення різноманітних характеристик користувача у вигляді структурованих тверджень, тоді як політики визначають правила доступу на основі цих тверджень.

Поєднання заявок і політик дозволяє відокремити дані про користувача від логіки контролю доступу. Політики аналізують наявні заявки та приймають рішення щодо надання або обмеження доступу до функціональних можливостей чи ресурсів системи.

Такий підхід підвищує адаптивність і безпеку застосунку, спрощує супровід авторизаційної логіки та створює основу для реалізації складних сценаріїв керування доступом.

2.1. Поняття заявок

Заявки є універсальним механізмом представлення інформації про користувача у вигляді окремих тверджень, що описують його ідентичність, властивості або повноваження. Кожна заявка містить значення певного типу даних та відомості про джерело їх походження, що дозволяє оцінювати надійність і контекст отриманої інформації.

Використання заявок забезпечує гнучкий підхід до аутентифікації та авторизації, оскільки вони можуть формуватися як усередині застосунку, так і надходити із зовнішніх систем. На

відміну від жорстко фіксованих ролей, заявки дозволяють більш детально описувати характеристики користувача та застосовувати їх у процесі контролю доступу.

Таким чином, заявки слугують основою для побудови розширених механізмів безпеки, поєднуючи дані з різних джерел і забезпечуючи адаптивну модель керування доступом.

2.2. Створення заявок

Створення заявок полягає у формуванні та додаванні до контексту користувача додаткових тверджень, які описують його характеристики, права або атрибути. Заявки можуть створюватися під час аутентифікації, у процесі обробки запитів або шляхом отримання даних із зовнішніх джерел.

Механізм створення заявок дозволяє динамічно розширювати інформацію про користувача без зміни базової моделі даних. При цьому кожна заявка містить не лише значення, а й відомості про її тип та джерело, що забезпечує контроль достовірності інформації.

Завдяки створенню заявок система отримує можливість гнучко керувати доступом і поведінкою користувачів, використовуючи актуальні та контекстні дані в межах процесів авторизації.

2.3. Використання політик

Політики є механізмом централізованого керування авторизацією, який визначає правила доступу користувачів до ресурсів на основі заданих вимог. Такі вимоги можуть включати наявність певних заявок, ролей або інших умов, що мають бути виконані для отримання дозволу на доступ.

Використання політик дозволяє відокремити логіку авторизації від прикладного коду, підвищуючи його зрозумілість і підтримуваність. Політики застосовуються на рівні контролерів, дій або окремих ресурсів і перевіряються автоматично під час обробки запитів.

Завдяки політикам система авторизації стає більш гнучкою та масштабованою, оскільки правила доступу можна змінювати або розширювати без суттєвого впливу на загальну структуру застосунку.

2.4. Використання політик для авторизації доступу до ресурсів

Авторизація доступу до ресурсів за допомогою політик передбачає оцінювання прав користувача з урахуванням конкретного об’єкта, до якого запитується доступ. На відміну від загальної авторизації на рівні дій або контролерів, такий підхід дозволяє враховувати властивості самого ресурсу та контекст взаємодії з ним.

Політики авторизації ресурсів використовують спеціальні вимоги та обробники, які аналізують як дані користувача, так і характеристики ресурсу. Це забезпечує більш точний контроль доступу, коли рішення залежить від взаємозв’язку між користувачем і конкретним елементом даних.

Застосування цього підходу підвищує рівень безпеки системи та дозволяє реалізовувати складні сценарії доступу, зберігаючи при цьому структурованість і розширюваність механізмів авторизації.

3. Використання сторонньої автентифікації

Стороння автентифікація передбачає підтвердження особи користувача за допомогою зовнішніх сервісів, що виступають довіреними постачальниками ідентифікації. Такий підхід дозволяє делегувати процес автентифікації спеціалізованим платформам і зменшити відповідальність застосунку за зберігання та захист облікових даних.

Використання сторонньої автентифікації забезпечує інтеграцію з поширеними обліковими записами користувачів і спрощує доступ до системи. Дані, отримані від зовнішнього постачальника, можуть використовуватися для створення або пов’язування внутрішніх облікових записів і подальшої авторизації.

Застосування цього механізму підвищує безпеку, зручність та масштабованість системи, а також сприяє уніфікації процесів входу користувачів у різних застосунках.

3.1. Реєстрація програми в Google

Реєстрація програми в Google є необхідним етапом для використання зовнішньої аутентифікації користувачів. Вона передбачає офіційне оголошення застосунку як довіреного клієнта, який має право взаємодіяти з сервісами автентифікації Google.

У процесі реєстрації визначаються основні параметри безпеки, зокрема ідентифікаційні дані програми та допустимі адреси зворотного виклику. Це дозволяє сервісу перевіряти справжність запитів на аутентифікацію та запобігати несанкціонованому доступу.

Результатом реєстрації є створення облікових даних, які використовуються застосунком для безпечної інтеграції з механізмами сторонньої аутентифікації та забезпечення довіри між системами.

3.2. Увімкнення аутентифікації Google

Увімкнення аутентифікації Google полягає в налаштуванні застосунку для використання зовнішнього постачальника ідентифікації користувачів. На цьому етапі застосунок конфігурується таким чином, щоб приймати та обробляти дані, отримані від сервісів Google під час процесу входу.

Інтеграція аутентифікації Google забезпечує можливість підтвердження особи користувача без необхідності зберігати його облікові дані всередині системи. Отримана від стороннього постачальника інформація перетворюється у внутрішній формат і використовується в механізмах авторизації.

Завдяки увімкненню аутентифікації Google підвищується зручність користування та рівень безпеки застосунку, а також спрощується процес керування користувачами.

Література

Базова

1. Freeman A. Pro Entity Framework Core 2 for ASP.NET Core MVC. Apress, 2018. 652 p.

2. Naylor L. ASP.NET MVC with Entity Framework and CSS. Apress, 2016. 628 p.
3. Gorman B. Practical Entity Framework Core 6: Database Access for Enterprise Applications. Apress, 2022. 812 p.
4. Smith J. Entity Framework Core in Action. Manning, 2021. 626 p.
5. Freeman A. Pro ASP.NET Core Identity: Under the Hood with Authentication and Authorization in ASP.NET Core 5 and 6 Applications. Apress, 2021. 746 p.

Допоміжна

6. Jürgen W. HTML. The Comprehensive Guide. Rheinwerk Computing, 2023. 817 p.
7. Meyer E. CSS. The Definitive Guide. O'Reilly, 2023. 1129 p.
8. Flanagan D. JavaScript. The Definitive Guide. O'Reilly, 2020. 707 p.
9. Schildt H. C# 4.0 The Complete Reference. McGraw-Hill, 2010. 976 p.
10. Freeman A. Pro ASP.NET Core 6 Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. Apress, 2022. 1267 p.

Інтернет ресурси

11. freeCodeCamp.org. ASP.NET Core MVC Course for Beginners (.NET 9). URL: https://www.youtube.com/watch?v=RWXKysImabs
12. Frank Liu. Full Course – Learn ASP.NET Core MVC in .NET 8 | CRUD Operations | EntityFramework | MVC Tutorial. URL: https://www.youtube.com/watch?v=BzlPrVB_DwA
13. Frank Liu. ASP.NET CORE Authentication & Authorization Overview | Security & Identity Series. URL: https://www.youtube.com/watch?v=p-00eBUpGa4&list=PLgRlicSxjeMOxypAEL2XqIc2m_gPmoVN-
14. Teddy Smith. ASP.NET Core MVC 2022. 13. Identity Framework. URL: https://www.youtube.com/watch?v=THKTQKrg4a4
15. Milan Jovanović. Authentication made easy with ASP.NET Core Identity in .NET 8. URL: https://www.youtube.com/watch?v=S0RSsHKiD6Y

Навчальне видання

Конспект лекцій з дисципліни «Сучасні мови програмування» для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 124 Системний аналіз (частина 2) / Укл.: К. О. Трифонова. − Одеса, 2025. − 29 с.

Укладач: Трифонова К. О., ст. викл.

Підписано до друку _____. Формат 60х84/16. Папір газетний. Друк офсетний. 0,87 ум. друк. арк. 0,94 обл. - вид. арк.
Тираж 100 пр. Зам. №

Національний університет «Одеська політехніка»
65044, Одеса, пр. Шевченка, 1