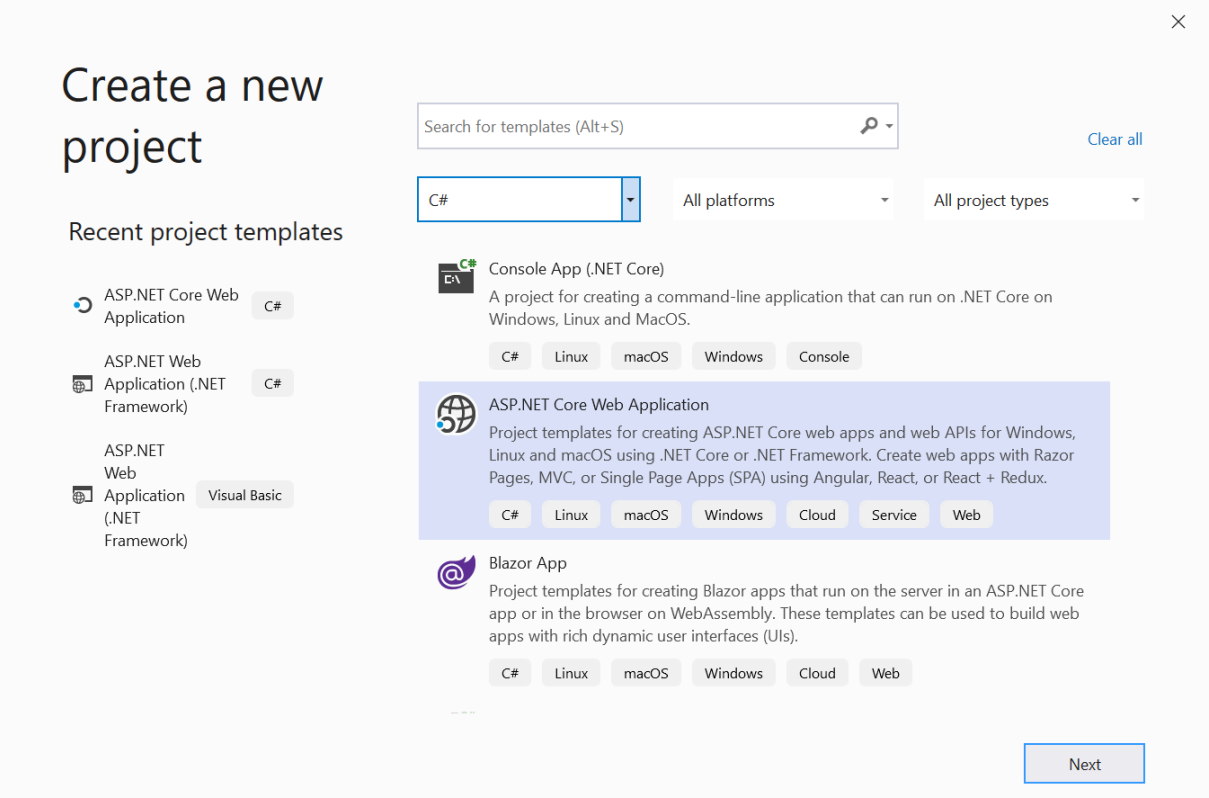
Microsoft .NET: ASP.NET MVC + Entity Framework (Code First)

Do realizacji projektu potrzebne jest zintegrowane środowisko programistyczne Microsoft Visual Studio 2019.

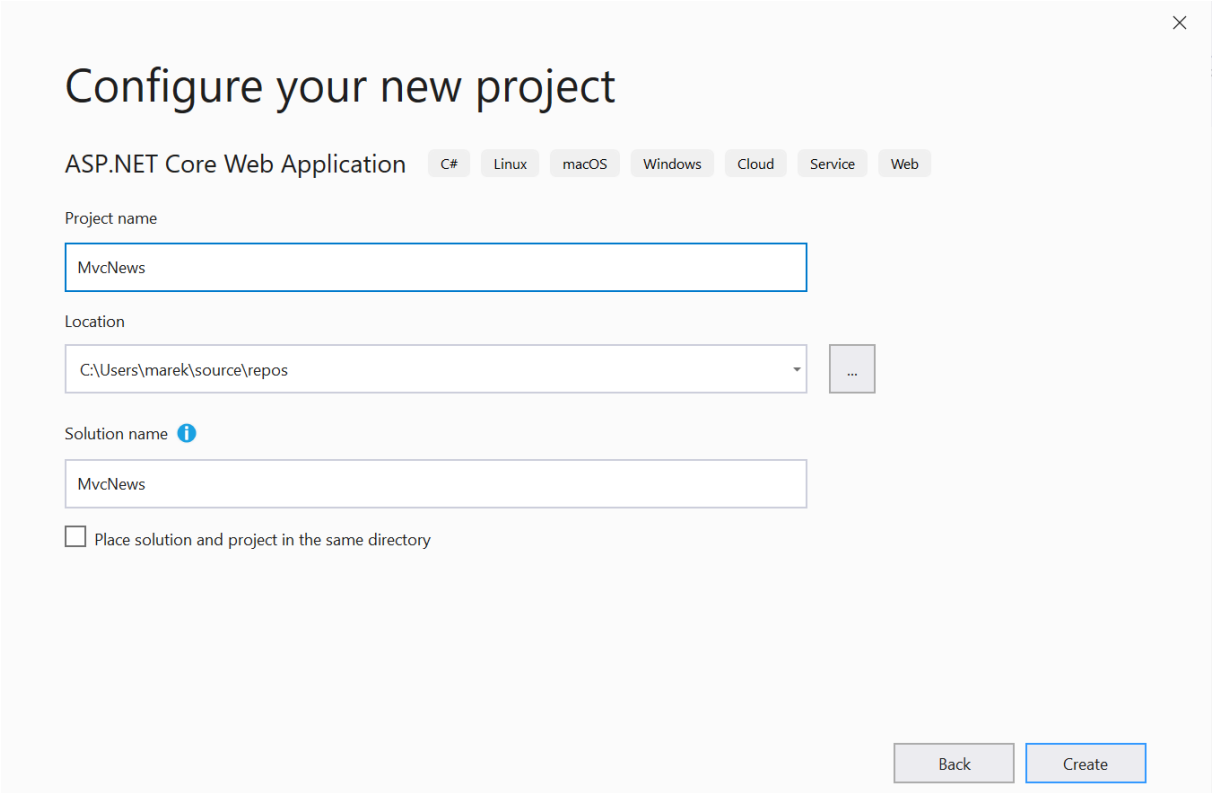
W ramach projektu budowana jest prosta aplikacja wykorzystująca framework MVC, bazująca na modelu bazy danych utworzonym zgodnie ze strategią Code First. Aplikacja zostanie wygenerowana kreatorem MVC scaffolding.

Ćwiczenie 1 (Podstawowa aplikacja CRUD)

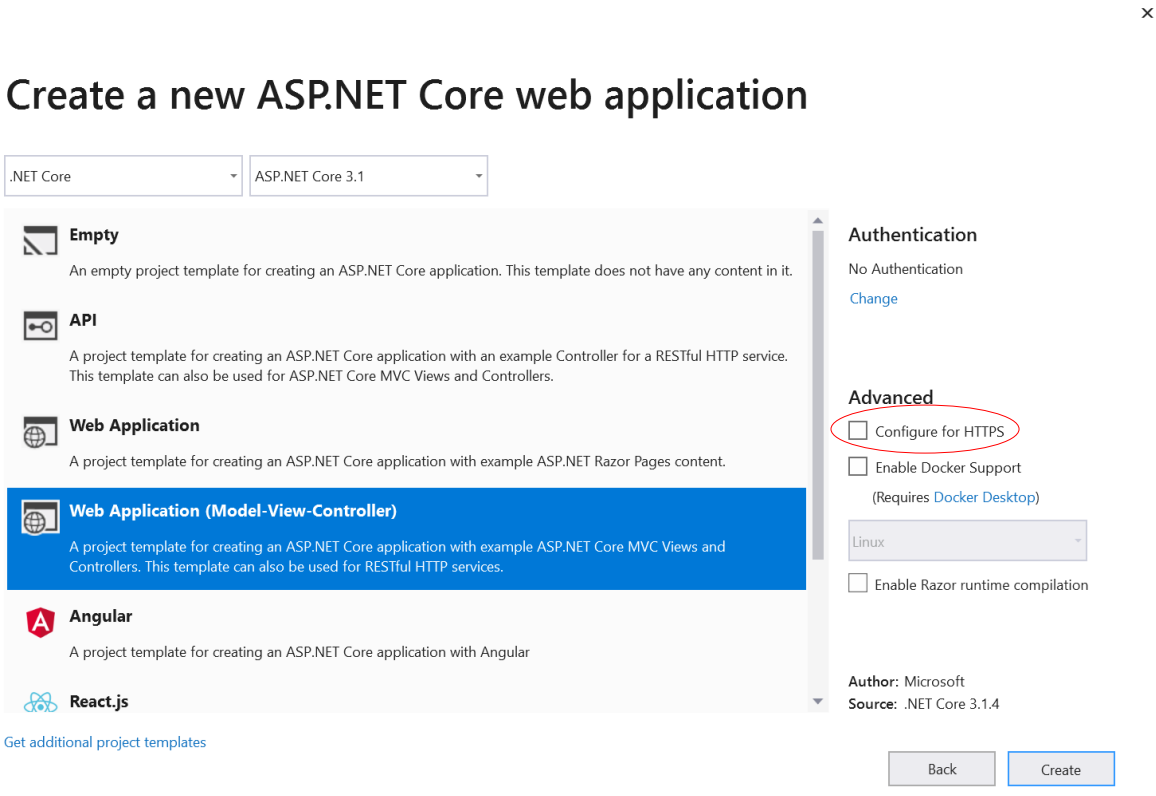
1. Utworzenie nowego projektu.
   a) Uruchom narzędzie Microsoft Visual Studio.
   b) Z menu głównego wybierz File→New Project. Wybierz szablon ASP.NET Core Web Application dla języka C# (szczególną uwagę zwróć na wybór właściwego języka). Kliknij przycisk Next.



   c) W drugim kroku kreatora jako nazwę projektu podaj „MvcNews”. Pole Place solution and project in the same directory pozostaw niezaznaczone. Pozostałe opcje pozostaw domyślne i kliknij przycisk Create.



d) W oknie wyboru frameworków wybierz Web Application (Model-View-Controller). Odznacz pole wyboru Configure for HTTPS. Pozostałe opcje pozostaw domyślne i kliknij przycisk Create.



e) Obejrzyj strukturę projektu w panelu Solution Explorer zwracając uwagę na utworzone przez kreator foldery.

f) Obejrzyj kod startowy aplikacji w pliku `Program.cs`. Zawiera on konfigurację domyślnego serwera do uruchomienia aplikacji.

g) Obejrzyj kod odpowiedzialny za inicjalizację aplikacji w pliku `Startup.cs`. Odszukaj metodę zawierającą konfigurację usług wymaganych przez aplikację. Odszukaj metodę odpowiedzialną za konfigurację obsługi żądań HTTP, a w niej definicję reguły routingu z charakterystycznym dla ASP.NET MVC schematem adresów URL.

h) Uruchom projekt kombinacją klawiszy Ctrl+F5 (Start Without Debugging).

i) Przetestuj nawigację po stronach aplikacji zwracając uwagę na zawartość paska adresu w przeglądarce.

2. Modyfikacja obsługi strony z polityką prywatności wygenerowanej przez kreator, w celu zapoznania się ze współpracą kontrolera z widokiem.

a) Otwórz do edycji klasę kontrolera HomeController i wstaw poniższą linię kodu na początku ciała metody Privacy():

```
ViewData["Policy"] = "Please do not enter any sensitive information.";
```

b) Otwórz do edycji stronę widoku wyświetlającą politykę prywatności i zastąp w niej statyczną zawartość elementu <p> poniższym kodem:

```
@ViewData["Policy"]
```

c) Odśwież stronę w przeglądarce i sprawdź czy tekst polityki został poprawnie przekazany do widoku i wyświetlony.

d) Zmień kod zarówno kontrolera jak i widoku tak, aby przekazanie tekstu polityki było realizowane poprzez obiekt ViewBag. Odśwież stronę w przeglądarce by upewnić się, że aplikacja działa tak samo jak wcześniej.

Komentarz: ViewBag stanowi wygodną abstrakcję nad ViewData, ale trzeba pamiętać, że nie jest dostępny w modelu programistycznym Razor Pages.

3. Utworzenie obiektowego modelu danych w projekcie zgodnie ze strategią Code First.

a) W panelu Solution Explorer wywołaj prawym klawiszem myszy menu kontekstowe dla folderu `Models` projektu i wybierz opcję Add -> Class. Jako nazwę pliku podaj `NewsItem.cs`.

b) W utworzonej klasie zdefiniuj jako publiczne poniższe właściwości (właściwości w sensie składni C#, a nie pola!):

typ	nazwa
int	Id
DateTime	TimeStamp
string	Text

4. Dodanie do projektu pakietów Entity Framework i dostawcy usług dla SQL Server.

   a) Uruchom konsolę mendżera pakietów opcją Tools -> NuGet Package Manager -> Package Manager Console z głównego menu.

   b) Z poziomu konsoli uruchom następującą komendę:

      ```
      Install-Package Microsoft.EntityFrameworkCore.SqlServer
      ```

      Komentarz: Powyższa komenda instaluje pakiet dostawcy EF Core dla serwera SQL Server, ale spowoduje to zainstalowanie również pakietu EF Core jako jego zależności.

5. Utworzenie klasy kontekstu bazy danych.

   a) Utwórz w strukturze projektu folder `Data` opcją Add -> New Folder z poziomu węzła projektu w Solution Explorer.

   b) W utworzonym folderze utwórz klasę o nazwie NewsDbContext, a następnie podmień jej definicję na poniższą (z zachowaniem utworzonej przez kreator przestrzeni nazw).

```
public class NewsDbContext : DbContext
{
  public NewsDbContext(DbContextOptions< NewsDbContext > options) :
                                                          base(options)
  {}

  public DbSet<NewsItem> News { get; set; }
}
```

      Zaimportuj niezbędne przestrzenie nazw.

      Komentarz: Konstruktor z parametrem DbContextOptions (zamiast domyślnego bezargumentowego) umożliwi ustawianie opcji konfiguracyjnych kontekstu podczas jego rejestracji w aplikacji.

   c) Przejdź do edycji pliku konfiguracyjnego aplikacji `appsettings.json`. Dodaj na końcu (przed ostatnim nawiasem klamrowym, oddzielając przecinkiem od poprzedniej opcji) poniższą sekcję z łańcuchem połączenia z bazą danych:

```
"ConnectionStrings": {
  "NewsDbContext":
  "Server=(localdb)\\mssqllocaldb;Database=News;Trusted_Connection=True"
}
```

      Komentarz: Podany łańcuch połączeniowy zakłada wykorzystanie instancji serwera SQL Server w wersji LocalDb, odpowiedniej dla zastosowań deweloperskich, instalowanej wraz z Visual Studio. Należy zauważyć, że wskazana w parametrach połączenia baza danych jeszcze nie istnieje. Zostanie ona utworzona później zgodnie ze strategią Code First.

      Konieczna jest jeszcze rejestracja kontekstu bazy danych jako jednej z usług wykorzystywanych w aplikacji. W tym celu w klasie Startup na końcu metody ConfigureServices dodaj poniższy kod:

```
services.AddDbContext<NewsDbContext>(options =>
    options.UseSqlServer(Configuration
                        .GetConnectionString("NewsDbContext")));
```

d) Po dodaniu powyższej instrukcji, do listy instrukcji using dodaj instrukcję importującą przestrzeń nazw klasy kontekstu bazy danych oraz poniższą instrukcję importującą przestrzeń nazw EF Core:

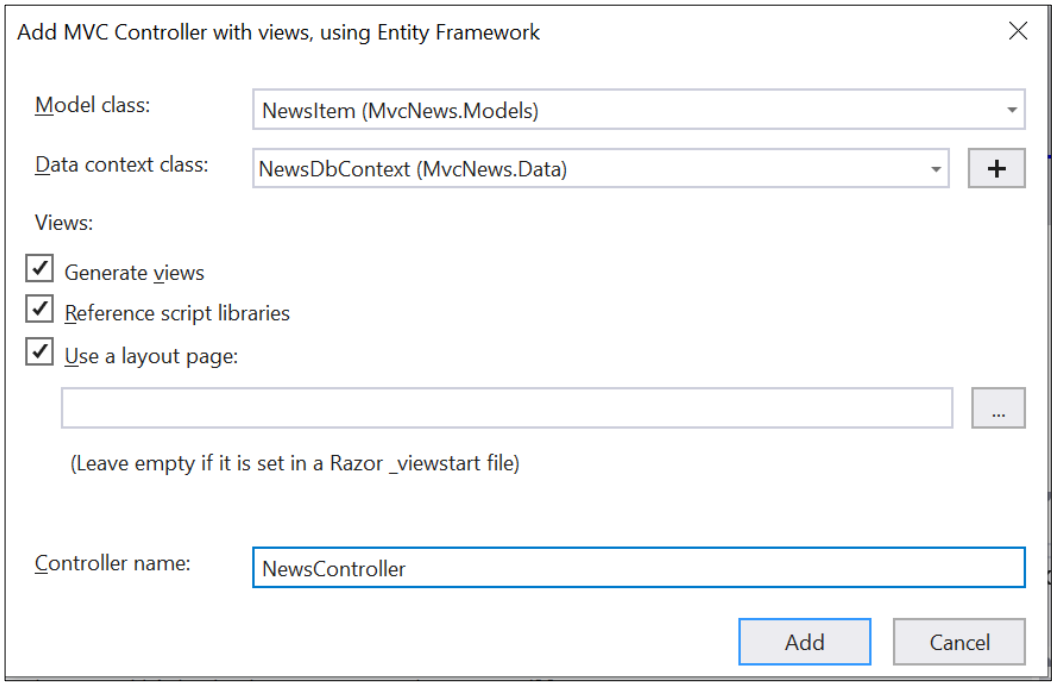
```
using Microsoft.EntityFrameworkCore;
```

e) Zapisz wszystkie zmiany.

f) Przebuduj projekt, aby upewnić się, że nie zawiera błędów (Build).

6. Wygenerowanie kontrolerów i widoków do obsługi modelu.

a) Z menu kontekstowego węzła Controllers w panelu Solution Explorer wybierz opcję Add -> Controller (ewentualnie poprzez opcję Add New Scaffolded Item). W kreatorze kontrolera wybierz szablon scaffoldingu MVC controller with views, using Entity Framework. Kliknij przycisk Add i w kolejnym kroku kreatora wybierz utworzoną wcześniej klasę modelu i kontekstu bazy danych oraz popraw nazwę klasy kontrolera na NewsController. Pozostałe opcje pozostaw domyślne i kliknij przycisk Add.



b) Obejrzyj wygenerowaną klasę kontrolera i związane z nim widoki. W klasie kontrolera zwróć uwagę na to, że metody akcji, które realizują operacje na bazie danych są asynchroniczne (i wykorzystują asynchroniczne metody Entity Framework). Zwiększa to skalowalność aplikacji, gdyż wątki usługowe serwera nie będą blokowane oczekiwaniem na odpowiedź bazy danych, a będą mogły być wykorzystane do obsługi innych żądań.

c) Odszukaj w strukturze projektu stronę wzorcową `_Layout.cshtml` i dodaj w menu, które wyświetla się na każdej stronie u góry, link wywołujący akcję Index kontrolera News.

d) Przebuduj projekt (Rebuild).

7. Utworzenie bazy danych przy użyciu migracji.

a) Przejdź do konsoli menedżera pakietów (lub otwórz ją ponownie, jeśli została zamknięta: Tools->NuGet Package Manager->Package Manager Console).

b) Wykonaj w oknie Package Manager Console kolejno poniższe komendy.

```
Add-Migration InitialCreate

Update-Database
```

c) Obejrzyj treść metod Up i Down w wygenerowanej klasie migracji, służących w tym wypadku odpowiednio do utworzenia i usunięcia tabeli w bazie danych.

8. Uruchomienie aplikacji i przetestowanie jej działania.

a) Uruchom aplikację (Ctrl+F5)

b) Przetestuj działanie aplikacji dodając, edytując, przeglądając i usuwając dane.

9. Ustawienie domyślnej daty dodawanych newsów na bieżącą.

a) W metodzie Create() kontrolera NewsController (wywoływanej metodą GET HTTP) utwórz nowy obiekt NewsItem. Ustaw w nim datę na bieżącą (System.DateTime.Now), a następnie przekaż go do widoku.

b) Uruchom aplikację i przetestuj dodawanie nowych newsów.

10. Dodanie obowiązkowości i weryfikacji długości tekstu newsa.

a) Dodaj w klasie encji NewsItem następujące adnotacje: (1) włączającą obowiązkowość tekstu newsa ([Required]), (2) weryfikującą, że jego długość wynosi od 5 do 140 znaków ([StringLength]), (3) wskazującą że etykieta czasowa newsa jest typu daty ([DataType]), aby nie wyświetlały się składniki czasu. Nie zapomnij o zaimportowaniu przestrzeni nazw adnotacji.

b) Uruchom aplikację aby przetestować działanie dodanych adnotacji.

Ćwiczenie 2 (Zarządzanie współbieżnością)

1. Wejdź na stronę aplikacji w dwóch przeglądarkach i sprawdź zachowanie aplikacji w przypadku współbieżnej edycji treści tego samego newsa.

Komentarz: Aplikacja w obecnej postaci nie zawiera mechanizmów zarządzania współbieżnością. Oznacza to, że w sytuacji współbieżnej modyfikacji tych samych danych ostatni zapis „wygrywa". Co więcej, użytkownik zapisujący zmiany jako drugi nie ma świadomości, że nadpisuje dane, które zostały zmodyfikowane po momencie ich odczytu przez niego (anomalia utraconej modyfikacji, ang. lost update). Jeśli takie zachowanie aplikacji jest nieakceptowalne, należy zaimplementować w niej zarządzanie współbieżnością. Strategia pesymistyczna (z blokowaniem danych w bazie danych na czas między ich odczytem i modyfikacją) jest nieodpowiednia dla aplikacji internetowych, w których odczyt i późniejsza modyfikacja są realizowane w odrębnych transakcjach bazodanowych. Zresztą, Entity Framework obecnie nie wspiera strategii pesymistycznej. Zostaje więc strategia optymistyczna, opierająca się na wykrywaniu potencjalnych zmian w bazie danych w momencie zapisu.

2. Włącz mechanizm optymistycznego zarządzania współbieżnością dla encji NewsItem z wykorzystaniem dedykowanej do tego celu właściwości (RowVersion):

- dodaj w klasie encji NewsItem publiczną właściwość RowVersion typu byte[] i oznacz ją adnotacją [Timestamp]

- dodaj migrację AddRowVersionMig i uaktualnij bazę danych

3. Uwzględnij właściwość wersji w procesie edycji danych w aplikacji:

- zmień adnotację Bind metody kontrolera do edycji newsa wołanej przez POST, tak aby uwzględniała dodaną do encji właściwość wersji

- spraw aby właściwość wersji była uwzględniona na stronie do edycji newsa jako ukryte pole formularza, analogicznie do identyfikatora encji (bez tego oryginalna wartość właściwości wersji byłaby tracona w procesie edycji danych w aplikacji ASP.NET MVC!)

- uruchom aplikację w dwóch przeglądarkach i ponownie przetestuj współbieżną edycję danych

Komentarz: Oczekiwany rezultat to zgłoszony wyjątek podczas zapisu, który doprowadziłby do anomalii utraconej modyfikacji.

4. Dodaj obsługę wyjątku optymistycznego zarządzania współbieżnością:

Uwaga: Kreator kontrolera wygenerował instrukcję try/catch łapiącą stosowny wyjątek, ale w przypadku gdy konfliktową operacją była również modyfikacja, a nie usunięcie, wyjątek jest rzucany ponownie.

- odszukaj instrukcję throw rzucającą ponownie wyjątek po jego złapaniu

- zamiast ponownego rzucenia wyjątku, za pomocą metody ModelState.AddModelError, dodaj komunikat informujący o tym że edytowane dane zostały zmodyfikowane lub usunięte przez innego użytkownika. Uwaga: Jako klucz (key) błędu podaj pusty łańcuch znaków, gdyż błąd nie jest związany z żadnym konkretnym atrybutem modelu

- pamiętaj, aby po przekazaniu komunikatu o błędzie za pomocą odpowiedniej instrukcji return sprawić aby nastąpiło ponowne wyświetlenie strony do edycji danych, na której ma być wyświetlony ten komunikat wraz z prezentacją edytowanych danych

- ponownie uruchom aplikację w dwóch przeglądarkach i ponownie przetestuj współbieżną edycję danych zwracając uwagę na własny komunikat o błędzie

- na stronie wyświetlającej komunikat o błędzie ponów próbę zapisu danych

Komentarz: Kolejne próby zapisu (bez wcześniejszego odczytu aktualnego wiersza z bazy danych) są skazane na niepowodzenie ze względu na niezgodność wersji edytowanego wiersza w aplikacji i w bazie danych. Poza tym, umożliwienie użytkownikowi zapisu danych bez sprawdzenia nadpisywanych w bazie danych informacji trudno uznać za pełne rozwiązanie problemu utraconej modyfikacji… Oba problemy rozwiążemy w następnym kroku ćwiczenia.

5. W kodzie metody kontrolera do edycji newsa wołanej przez POST, między dodaniem komunikatu o błędzie współbieżnej modyfikacji a zwróceniem widoku, wstaw kolejno następujące fragmenty kodu:

- odczyt aktualnych danych z bazy (e reprezentuje złapany wyjątek) :

```
var entry = e.Entries.Single();
var databaseEntry = entry.GetDatabaseValues();
var databaseEntity = (NewsItem)databaseEntry.ToObject();
```

- podmiana wartości właściwości wersji w edytowanej instancji encji na aktualną z bazy danych oraz usunięcie atrybutu wersji z ModelState (wartości z ModelState mają „wyższą rangę" niż właściwości obiektu modelu przekazanego do widoku i podczas zapisu do bazy danych je nadpisują gdy są ustawione):

```
newsItem.RowVersion = (byte[]) databaseEntity.RowVersion;
ModelState.Remove("RowVersion");
```

- przekazanie do widoku informacji o aktualnych wartościach edytowanych atrybutów, wykorzystując do tego celu komunikaty o błędach powiązane z atrybutami modelu:

```
ModelState.AddModelError("TimeStamp", "Current value: " +
  (DateTime) databaseEntity.TimeStamp);
ModelState.AddModelError("Text", "Current value: " +
  (string) databaseEntity.Text);
```

Komentarz: Jest to jeden z możliwych sposobów poinformowania użytkownika o aktualnych wartościach edytowanych danych. Alternatywnie, można byłoby podmienić właściwości modelu przekazywanego do widoku na odczytane z bazy danych (tak jak zrobiliśmy z właściwością wersji). W takim wypadku użytkownik traciłby swoje modyfikacje.

6. Ponownie uruchom aplikację w dwóch przeglądarkach i ponownie przetestuj współbieżną edycję danych z ponowieniem zapisu zmienionych danych po wystąpieniu błędu współbieżnej modyfikacji.

7. Sprawdź jak zachowa się aplikacja gdy podczas próby usunięcia newsa (przed potwierdzeniem tej operacji) dany news zostanie skutecznie zmodyfikowany przez współbieżną sesję.

Komentarz: Sytuacja taka nie powoduje wyjątku, gdyż w aplikacji z kreatora do metody kontrolera obsługującej potwierdzenie usunięcia przekazywany jest tylko identyfikator newsa, a nie cały obiekt.

8. W sygnaturze metody do usuwania danych wołanej metodą POST dodaj jako drugi parametr newsItem typu NewsItem.

Uwaga: Nie jest konieczne oznaczenie dodanego parametru atrybutem [Bind]. Bez niego wszystkie właściwości będą podlegały wiązaniu. Przy obsłudze edycji modyfikowaliśmy atrybut [Bind] gdyż dodał go kreator.

9. Usuń pierwszy wiersz kodu metody do usuwania danych wołanej metodą POST, odpowiadający za odczytanie z bazy danych instancji encji do usunięcia na podstawie identyfikatora. Instancję tę będziemy mieli przekazaną przez framework poprzez dodany parametr.

10. Aby wszystkie właściwości usuwanej instancji encji zostały przekazane poprzez wiązanie danych, dodaj na stronie do usuwania danych ukryte pola formularza dla wszystkich atrybutów newsa (po istniejącym już polu z identyfikatorem):

```
        <input type="hidden" asp-for="TimeStamp" />
        <input type="hidden" asp-for="Text" />
        <input type="hidden" asp-for="RowVersion" />
```

11. Sprawdź jak teraz zachowa się aplikacja gdy podczas próby usunięcia newsa (przed potwierdzeniem tej operacji) dany news zostanie skutecznie zmodyfikowany przez współbieżną sesję.

Komentarz: Powinien pojawić się ten sam wyjątek, co wcześniej w przypadku edycji.

12. Otocz linie kodu realizujące usunięcie instancji encji blokiem try i dodaj sekcję catch obsługującą wyjątek współbieżnej modyfikacji):

```
try
{
    _context.News.Remove(newsItem);
    await _context.SaveChangesAsync();
}
catch (DbUpdateConcurrencyException e)
{
    if (!NewsItemExists(newsItem.Id))
    {
        return NotFound();
    }
    else
    {
        ModelState.AddModelError("", "Unable to save changes. The record
was modified by another user after you got the original value");

        var entry = e.Entries.Single();
        var databaseEntry = entry.GetDatabaseValues();
        var databaseEntity = (NewsItem)databaseEntry.ToObject();

        ModelState.Remove("RowVersion");

        return View(databaseEntity);
    }
}
```

Komentarz: Powyższy kod, w sytuacji gdy miała miejsce współbieżna modyfikacja usuwanego newsa, odczyta jego aktualną postać z bazy danych i przekaże do widoku, aby użytkownik mógł podjąć decyzję odnośnie usunięcia danych po zapoznaniu się z ich aktualną zawartością.

13. Odszukaj w kodzie strony do edycji element wyświetlający globalne komunikaty o błędach walidacji, skopiuj go, a następnie wstaw do kodu strony do usuwania danych przed pytaniem o chęć usunięcia.

14. Ponownie sprawdź jak zachowa się aplikacja gdy podczas próby usunięcia newsa (przed potwierdzeniem tej operacji) dany news zostanie skutecznie zmodyfikowany przez współbieżną sesję. Zwróć uwagę na komunikat o błędzie i to czy przy ponownym wyświetleniu strony z prośbą o potwierdzenie usunięcia newsa wyświetlona będzie aktualna postać danych. Upewnij się, że po zapoznaniu się z komunikatem o współbieżnej modyfikacji, faktycznie można usunąć newsa.

15. Na zakończenie sprawdź jak zachowuje się aplikacja gdy w trakcie edycji lub usuwania newsa zostanie on usunięty we współbieżnej sesji.

Komentarz: Zostawiliśmy rozwiązanie dla takich scenariuszy, wygenerowane przez kreator kontrolera dla operacji edycji i skopiowaliśmy je do kodu obsługującego współbieżność dla operacji usuwania. Polega ono na zwróceniu kodu statusu 404 HTTP. W aplikacji produkcyjnej, po pierwsze należałoby rozważyć dodanie przyjaznej strony z komunikatem o błędzie 404, a po drugie dodanie dedykowanego widoku do obsługi sytuacji gdy nieznalezienie danych do edycji lub usunięcia jest wynikiem usunięcia ich we współbieżnej sesji.