# Programowanie zaawansowane

11

## Getting Started with Entity Framework 6 Code First using MVC 5 (część 3/3)

Ćwiczenia będę realizowane w oparciu o tutorial na stronie:
https://www.asp.net/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application

**Wskazówka 1**. Autoryzacja

Aby dostęp do danej akcji miała tylko zalogowana osoba wystarczy poprzedzić ją atrybutem *Authorize* (atrybuty podajemy przed akcją – metodą w kontrolerze – w nawiasach kwadratowych). Jeżeli chcemy zabezpieczyć dostęp do wszystkich akcji kontrolera wystarczy podać ten atrybut przed kontrolerem.

Jeżeli chcemy dokładniej zdefiniować warunki dostęp, w okrągłych nawiasach po nazwie atrybutu możemy wymienić (poprzez nazwane parametry) nazwy wybranych użytkowników (parametr *Users*) lub role (parametr *Roles*). Jeżeli chcemy podać więcej niż jednego użytkownika lub jedną rolę podajemy je wszystkie w jednym stringu oddzielone przecinkami. Poniżej widzimy przykład ograniczenia dostępu do akcji do zalogowanych użytkowników z rolą *Admin* lub *Moderator*.

```
[Authorize(Roles = "Admin, Moderator")]
public ActionResult Create()
{
```

Po kliknięciu na link do akcji, do której nie mamy uprawnień, zostaniemy przeniesieni do formularza logowania.

W niektórych przypadkach lepszym rozwiązaniem może być po prostu niewyświetlanie linków do akcji, do których użytkownik nie ma praw. Aby to zrealizować wystarczy w widoku dodać sprawdzanie odpowiednich warunków. Poniższe przykłady pokazują sposób na sprawdzenie, czy aktualny użytkownik jest zalogowany oraz czy aktualny użytkownik ma przypisaną rolę *Admin*.

```
21  @if (Request.IsAuthenticated)
22  {
23      <li> @Html.ActionLink("Books", "Index", "Book") </li>
24  }
```



wbartyna@gmail.com

```
@if (Context.User.IsInRole("Admin"))
{
    <li> @Html.ActionLink("Books", "Index", "Book") </li>
}
```

Aby w akcji uzyskać dane o aktualnie zalogowanym użytkowniku wystarczy skorzystać z właściwości *User*. Jego nazwa jest dostępna jako właściwość *Name* obiektu zwracanego przez właściwość *Identity* klasy *User* (czyli *User.Identity.Name*).

**Wskazówka 2.** Przesyłanie zdjęć (plików) na serwer

Aby dodać wrzucanie zdjęć na serwer (np. zdjęć okładek książek), należy zrealizować następujące kroki (przedstawiam tu jedno z możliwych rozwiązań, można to robić na różne sposoby, korzystając z różnych klas, itp.):

1. W dokumencie XML będziemy przechowywać nazwę pliku, a nie cały plik. Dlatego do odpowiedniej klasie należy dodać właściwość reprezentującą nawę zdjęcia (niech ta właściwość w naszym przykładzie nazywa się *Obrazek* i będzie zdefiniowana w klasie reprezentującej książkę.

2. Do projektu dodajemy folder (menu kontekstowe projektu w *Solution Explorer*; *Add -> New Folder*), w którym będą przechowywane wrzucone zdjęcia. W naszym przykładzie ten folder nazywać się będzie *Obrazki*.

3. W widoku dodawania nowej książki

    a. dodajemy odpowiednie parametry w metodzie *BeginForm()*, tak aby możliwe było przesyłanie plików:

```
@using (Html.BeginForm("Create", "Ksiazki", FormMethod.Post,
                       new { enctype = "multipart/form-data" }))
```

    b. dodajemy kontrolkę z wyborem pliku i odpowiednio ją nazywamy (w przykładzie: *plikZObrazkiem*):

```
<div class="form-group">
    @Html.LabelFor(model => model.Obrazek, htmlAttributes:
                                    new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        <input type="file" name="plikZObrazkiem" />
        @Html.ValidationMessageFor(model => model.Obrazek, "",
                                    new { @class = "text-danger" })
    </div>
</div>
```

4. Po stronie kontrolera, akcję obsługującą dodawanie nowej książki możemy zaimplementować następującą (zmiany w stosunku do tego, co już znamy obejmują tylko pobranie pliku z żądania, wstawienie odpowiedniej nazwy obrazka do właściwości *Obrazek* i zapisanie pliku w utworzonym przez nas folderze (zakomentowana instrukcja pokazuje, w jaki sposób uwzględniać możliwość powtarzaniem się nazw obrazków i ich nadpisywaniem poprzez wykorzystanie unikalnego identyfikatora).



wbartyna@gmail.com

```
42      Ksiazka ksiazka = new Ksiazka();
43      UpdateModel(ksiazka);
44
45      HttpPostedFileBase file = Request.Files["plikZObrazkiem"];
46      if (file != null && file.ContentLength > 0)
47      {
48          //ksiazka.Obrazek = System.Guid.NewGuid().ToString();
49          ksiazka.Obrazek = file.FileName;
50          file.SaveAs(HttpContext.Server.MapPath("~/Obrazki/") + ksiazka.Obrazek);
51      }
52
53      biblio.DodajKsiazke(ksiazka);
54      biblio.Zapisz();
```

5. Aby wyświetlić zdjęcie w odpowiednim widoku dodajemy tag *img* z odpowiednimi parametrami (podajemy ścieżkę do pliku i jego nazwę z modelu).

```
42      <img src="~/Obrazki/@item.Obrazek" alt="" height="50px" width="50px" />
```

Obsługując edycję postępujemy w ten sam sposób.

**Wskazówka 3.** Wysyłanie wiadomości pocztą elektroniczną

Aby wysłać wiadomość należy stworzyć obiekt klasy *Message* (przestrzeń nazw *System.Net.Mail* oraz stworzyć i skonfigurować obiekt klasy *SmtpClient*.

```
public static void SendMail(string to, string subject, string body)
{
    var message = new System.Net.Mail.MailMessage(ConfigurationManager.AppSettings["sender"], to)
    {
        Subject = subject,
        Body = body
    };
    var smtpClient = new System.Net.Mail.SmtpClient
    {
        Host = ConfigurationManager.AppSettings["smtpHost"],
        Credentials = new System.Net.NetworkCredential(
            ConfigurationManager.AppSettings["sender"],
            ConfigurationManager.AppSettings["passwd"]),
        EnableSsl = true
    };
    smtpClient.Send(message);
}
```

Dane dotyczące adresu klienta poczty (*Host*) oraz uwierzytelniania (*Credentials*), czyli adres i hasło, można przechowywać w pliku konfiguracyjnym aplikacji. Do danych tych odwołujemy się za pomocą indeksatora kolekcji *ConfigurationManager.AppSettings*. Plik konfiguracyjny aplikacji sieciowej o nazwie *Web.config* znajduje się na dole drzewa katalogów projektu.

W pliku konfiguracyjnym widzimy konfigurację połączenia z bazą danych (*connectionString*) dla domyślnej bazy z ustawieniami aplikacji (w tym konta zarejestrowanych użytkowników) oraz dla naszej bazy z przedmiotami. W kolejnej grupie (*appSeettings*) możemy definiować własne parametry. W poniższym przykładnie dotyczą one poczty elektronicznej.



wbartyna@gmail.com

```
<configuration>
  <connectionStrings>
    <add name="ApplicationServices" connectionString="data source=.\S(
      providerName="System.Data.SqlClient" />
    <add name="InventoryConnectionString" connectionString="Data Sour(
      providerName="System.Data.SqlClient" />
  </connectionStrings>

  <appSettings>
    <add key="sender" value="jacekKosa@gmail.com"/>
    <add key="passwd" value="szarik"/>
    <add key="smtpHost" value="smtp.gmail.com"/>
  </appSettings>
```



wbartyna@gmail.com