04. Models

Music Store

Scaffolding (Grundgerüst…)

Model Binding

Models

- Wir sprechen über Models als
  - Objekte um Informationen an die DB zu senden.
  - Geschäftsprozesse berechnen und ausführen.
  - Daten in der View rendern.
- Diese Objekte repräsentieren die Domäne.
- ASP.NET MVC Tools
  - Focus auf eigentliche Problemdomäne.
  - Scaffolding: Tools, welche Controller- und View-Grundgerüste für die Standardszenarien Index, Create, Edit und Delete für jedes Model erstellen.

Music Store

```
//Album.cs
public class Album
{
    public virtual int AlbumId { get; set; }
    public virtual int GenreId { get; set; }
    public virtual int ArtistId { get; set; }
    public virtual string Title { get; set; }
    public virtual decimal Price { get; set; }
    public virtual string AlbumArtUrl { get; set; }
    public virtual Genre Genre { get; set; }
    public virtual Artist Artist { get; set; }
}
```

```
//Artist.cs
public class Artist
{
    public virtual int ArtistId { get; set; }
    public virtual string Name { get; set; }
}
```

Music Store

```
//Genre.cs
public class Genre
{
   public virtual int GenreId { get; set; }
   public virtual string Name { get; set; }
   public virtual string Description { get; set; }
   public virtual List<Album> Albums { get; set; }
}
```

- `virtual` properties?
  - Nicht notwendig, bietet aber Entity Framework Vorteile.
  - Ermöglicht Lazy Loading, effizientere Änderungsverfolgung (change tracking)

- Album hat **zwei** Properties für die Artist-Assoziation: `Artist` und `ArtistId`.
  - Das Foreign Key Property wird im Modelobjekt nicht benötigt.
  - Bietet aber viele Vorteile bei Verwendung der Tools.

- TODO: UML-Klassendiagramm erstellen.

Store Manager – Controller

- Ein Controller, welcher für ein Album die grundlegenden Funktionen/Operationen erlaubt: Lesen, Erstellen, Ändern und Löschen.
  - CRUD (Create, Read, Update, Delete)
- Pro Aufgabe/Operation braucht es mindestens eine Action-Methode im Controller und mindestens eine entsprechende View pro Action-Methode.
  - Kann schnell eine langweilige Fleissarbeit werden.
- Code-Grundgerüst automatisch erstellen aufgrund des Models.
  - Scaffolding (Code-Generierung)
  - Erstellt und benennt Controller mit allen Action-Methoden.
  - Erstellt und benennt zugehörige Views.
  - Data Acess.

Vorlagen (Scaffolding)

- MVC 5 Controller – leer
  - Leere `Index`-Action.
- MVC 5 Controller mit leeren Lese-/Schreib-Actions
  - Erstellt (leere) `Index`-, `Details`-, `Create`-, `Edit`-, und `Delete`-Actions.
- MVC 5 Controller mit Views unter Verwendung von Entity Framework
  - Erstellt `Index`-, `Details`-, `Create`-, `Edit`-, und `Delete`-Actions.
  - Erstellt benötigte Views.
  - Erstellt Code für Datenbankzugriffe.
- Web API 2 API Controller Grundgerüste
  - Dazu später mehr…

Scaffolding und Entity Framework

- Neues ASP.NET MVC 5 Projekt fügt automatisch eine Referenz zu EF hinzu.
- MVC-Scaffolders benutzen Code-First.
- Database First oder Model First ebenso möglich.
- Freie Wahl der Persistenz.
  - Datenbanken, simple Text-Files oder auch Web Services.

Code First Konventionen

- Zum Beispiel, um ein Objekt vom Typ `Album` in der DB zu speichern:
  - EF nimmt an, die Tabelle heisst `Albums`.
  - Falls eine Property **`ID`** existiert, nimmt EF an, diese sei der Primärschlüssel und setzt ein Autoinkrement auf dem Server…
  - …
  - Ab EF6 können alle Annahmen, bzw. Konventionen und Default-Werte definiert werden.
- *http://msdn.microsoft.com/en-us/data/jj819164*

DbContext

- Zugang zur Datenbank bietet eine von der DbContext abgeleitete Klasse.

```
//MusicStoreDB.cs
public class MusicStoreDB : DbContext
{
   public DbSet<Album> Albums { get; set; }
   public DbSet<Artist> Artists { get; set; }
   public DbSet<Genre> Genres { get; set; }
}
```

- Bietet äusserst praktische und kurze Funktionen, zum Beispiel alle Alben sortiert lesen:
  - ```
    var db = new MusicStoreDB();
    var albums = db.Albums.OrderBy(x => x.Title).ToList();
    ```
    ```

Grundgerüst erstellen

- TODO: Beispiel erstellen…

- Scaffolding fügt eine DataContext Datei, zum Beispiel `MusicStoreDB.cs`, zum `Models` Verzeichnis hinzu.
  - DbContext-Generierung ist eine **einmalige Sache**. -> kein überschreiben!
  - Data Migrations.
  - *https://msdn.microsoft.com/en-us/data/jj591621.aspx*
  - *https://msdn.microsoft.com/en-us/data/jj554735.aspx*

eager loading – lazy loading

```
//StoreManagerController.cs
public class StoreManagerController : Controller
{
  private MusicStoreDB db = new MusicStoreDB();

  // GET: /StoreManager/
  public ActionResult Index()
  {
    var albums =db.Albums.Include(a=>a.Artist).Include(a=>a.Genre);
    return View(albums.ToList());
  }
  //  ...
}
```

- Include-Methode teilt EF mit, die **eager loading** Strategie zu benutzen, um die zugehörigen Artist und Genre zu laden.
  - Eager loading versucht, sämtliche Daten (Album, Artist und Genre) mit einer einzigen Query zu laden.
- **Lazy loading** (Standard): Artist und Genre werden erst bei Bedarf geladen.

Index.cshtml

Database Initialisierung

```
//MusicStoreDbInitializer.cs
public class MusicStoreDbInitializer
        : System.Data.Entity.DropCreateDatabaseAlways<MusicStoreDB>
{
   protected override void Seed(MusicStoreDB context)
   {
      context.Artists.Add(new Artist {Name = "Al Di Meola"});
      context.Genres.Add(new Genre { Name = "Jazz" });
      context.Albums.Add(new Album
                         {
                            Artist = new Artist { Name="Rush" },
                            Genre = new Genre { Name="Rock" },
                            Price = 9.99m,
                            Title = "Caravan“
                         });
      base.Seed(context);
   }
}
```

Database Initialisierung

```
//global.asax.cs
//...
protected void Application_Start() {
   Database.SetInitializer(new MusicStoreDbInitializer());

   AreaRegistration.RegisterAllAreas();
   FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
   RouteConfig.RegisterRoutes(RouteTable.Routes);
   BundleConfig.RegisterBundles(BundleTable.Bundles);
}
//...
```

Edit Album

```
//StoreManagerController.cs
//...
// GET: /StoreManager/Edit/5
public ActionResult Edit(int? id)
{
  if (id == null)
    return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

  Album album = db.Albums.Find(id);
  if (album == null)
    return HttpNotFound();

  ViewBag.ArtistId =
    new SelectList(db.Artists, "ArtistId", "Name", album.ArtistId);
  ViewBag.GenreId =
    new SelectList(db.Genres, "GenreId", "Name", album.GenreId);
  return View(album);
}
```

Edit Album

```
//Edit.cshtml
//...DropDown-Liste für Genres
<div class="col-md-10">
   @Html.DropDownList("GenreId", String.Empty)
   @Html.ValidationMessageFor(model => model.GenreId)
</div>
//...
```

SelectList

```
ViewBag.ArtistId =
    new SelectList(db.Artists, "ArtistId", "Name", album.ArtistId);

ViewBag.GenreId =
    new SelectList(db.Genres, "GenreId", "Name", album.GenreId);
```

- Erster Parameter: Values/Items
- Zweiter Parameter: Key
- Dritter Parameter: Bezeichnung/Anzeigename
- Vierter Parameter: Ausgewähltes Item

Model Binding

```
//EinController.cs
public ActionResult EineAction()
{
   System.Collections.Specialized.NameValueCollection coll = null;

   if(Request.HttpMethod.Equals("GET"))
      coll = Request.QueryString;
   else if(Request.HttpMethod.Equals("POST"))
      coll = Request.Form;
   else
      return new HttpStatusCodeResult(
                              System.Net.HttpStatusCode.BadRequest);

   string name = coll["familienname"];
   int alter = Int32.Parse(coll["alter"]); //Request.Form["alter"]
   string sonstiges = coll["etwas"]; //Request.QueryString["alter"]

   return ...
}
```

Model Binding

```
//Kunde.cs
public class Kunde
{
   public string familienname { get; set; }
   public int alter {get; set;}
   public string etwas {get; set;}
}
```

```
//EinController.cs
public ActionResult EineAction(Kunde kunde)
{
   //...
}
```

Explizites Model Binding

```
//EinController.cs
public ActionResult EineAction()
{
    Kunde kunde = new Kunde();

    try {
        UpdateModel(kunde);
    }
    catch {
        //...
    }
}
```

```
//...
    if(TryUpdateModel(kunde)) {
        //...
    }
}
```