Notities Web apps III

Unit testing

Exception checken

```
[Fact]
public void Draw_EmptyDeck_ThrowsException() {
    for (int i = 1; i <= 52; i++)
        _deck.Draw();
    Assert.Throws<InvalidOperationException>(() => _deck.Draw());
}
```

Geparameteriseerd testen

```
[Theory]
[MemberData(nameof(Data))]
public void CanAddTheoryMemberDataProperty(int value1, int value2, int expected) {
    var calculator = new Calculator();
    var result = calculator.Add(value1, value2);
    Assert.Equal(expected, result);
}

public static IEnumerable<object[]> Data =>
    new List<object[]> {
        new object[] { 1, 2, 3 },
        new object[] { -4, -6, -10 },
        new object[] { -2, 2, 0 },
        new object[] { int.MinValue, -1, int.MaxValue }};
```

Type parameters voor mock training

- Om het even welke soort Product:

```
_mockProductRepository.Setup(m => m.Add(It.IsAny<Product>()));
```

- Een product die niet null mag zijn:

```
_mockProductRepository.Setup(p => p.Add(It.IsNotNull<Product>()));
```

Extension methods

Nieuwe map Extensions met bestand Extension.cs:

```
namespace Extensions {
    public static class Extension {
        public static IEnumerable<T> Shuffle<T>(this IEnumerable<T> collection) {
            int sizeOfCollection = collection.Count();
            IList<T> result = new List<T>(new T[sizeOfCollection]);
            ISet<int> positions = new HashSet<int>();
            Random random = new Random();
            foreach (T element in collection) {
                int randomPosition = random.Next(0, sizeOfCollection);
                while (positions.Contains(randomPosition)) {
                    randomPosition = random.Next(0, sizeOfCollection);
                }
                positions.Add(randomPosition);
                result.Insert(randomPosition, element);
            }
            return result.Where(element => element != null).AsEnumerable();
        }
    }
}
```

Entity

De te installeren package voor SQL Server:

```
Microsoft.EntityFrameworkCore.SqlServer
```

Entity - Linq

Overerving:

```
var courses = _brewer.Courses.OfType<OnlineCourse>().ToList();
```

Include multiple levels:

```
var category = context.Categories.Include(c => c.CategoryBrewers)
                                 .ThenInclude(b => b.Brewer).FirstOrDefault();
_brewers = category.Brewers;
```

N:M relatie creëren

N:M relatie tussen Product en Category:

1) Creëer een nieuwe modelklasse met daarin de properties: Category, Product, CategoryId, ProductId
2) Creëer Entityconfiguration klasse:

```
builder.ToTable("CategoryProduct");
builder.HasKey(b => new {b.CategoryId, b.ProductId});
builder.HasOne(b => b.Category).WithMany().
        HasForeignKey(c => c.CategoryId).OnDelete(DeleteBehavior.Cascade);
builder.HasOne(b => b.Product).WithMany().
        HasForeignKey(p => p.ProductId).OnDelete(DeleteBehavior.Cascade);
```

3) Pas configuration toe door een nieuwe instantie van CategoryProductConfiguration aan te maken in OnModelCreating van ApplicationDbContext

Opmerkingen

- Niet vergeten voor elke klasse die gepersisteerd moet worden, een protected lege constructor te creëren
- Indien men zaken wil ophalen drm van entity-linq: checken als het om objectreferenties gaat –> `include/theninclude` gebruiken

MVC

Repository pattern

Implementatie repositoryklasse:

- bij elke methode: `Include()` methoden niet vergeten voor referentie-attributen
- bij `GetAll()`: `AsNoTracking()` toevoegen (om performanter te maken)

Bv.: CategoryRepository:

```
public IEnumerable<Category> GetAll() { return _categories.AsNoTracking().ToList();}
```

Locale gegevens opslaan

1) In startUp klasse:
    1) In methode ConfigureServices: `services.AddSession();`
    2) In methode Configure: `app.useSession();` (voor `app.useMVC()`)
2) Klassen en properties taggen:
    1) Boven naam klassen die gepersisteerd moeten worden: `[JsonObject(MemberSerialization.OptIn)]`
    2) Boven properties die gepersisteerd moeten worden: `[JsonProperty]`
    3) Als klasse anders geïnitialiseerd moet worden wanneer het van lokale opslag afgelezen wordt (dan wanneer het expliciet aangemaakt wordt met de keyword `new`): Een extra Json constructor toevoegen:

```
[JsonConstructor]
private Product(int productId) {
 ProductId = productId;
}
```

3. Schrijven en lezen:
    1) Schrijven:

```
Cart c = JsonConvert.DeserializeObject<Cart>(HttpContext.Session.GetString("cart"));
```

2) Lezen:

```
HttpContext.Session.SetString("cart", JsonConvert.SerializeObject(_cart));
```

Opmerkingen

- Indien men parameter-id wil gebruiken in een controller-actionmethode, dan MOET de parameternaam `int id` zijn, het mag NIET `public IActionResult Edit(int brewerId) {...}` zijn
- Unit testing van controllers: Get en Post methoden APART testen

View snippets

Selection lists

1) In de controller de lijst als een SelectList-object doorgeven aan ViewData, bv. `ViewData["Categories"]`
2) HTML:

```
<form>
    <div class="form-inline">
        <div class="form-group">
            <label for="categoryId"></label>
            <select id="categoryId" name="categoryId"
                    asp-items="@(ViewData["Categories"] as SelectList)"
                    class="form-control">
                <option value="">-- all categories --</option>
            </select>
        </div>
        <button type="submit" class="btn btn-default">Submit</button>
    </div>
</form>
```

Form actions

```
<form>
    <button formaction="/Cart/Plus/@cartLine.Product.ProductId" type="submit">Plus</button>
    <button formaction="/Cart/Min/@cartLine.Product.ProductId" type="submit">Min</button>
    <button formaction="/Cart/Delete/@cartLine.Product.ProductId" type="submit">Delete</button>
</form>
```

Validatie

DisplayName

Gebruik van DiplayName voor enums:

1) Voeg `[Display(Name = ...)]` toe voor elke enum-element in de enumklasse
2) Creëer een extensionklasse EnumHelpers met een extension-hulp-methode `GetDisplayName(TEnum)`:

```
public static string GetDisplayName<TEnum>(this TEnum enumValue) {
    return typeof(TEnum).GetMember(enumValue.ToString())[0]
                        .GetCustomAttribute<DisplayAttribute>()?
                        .Name ?? enumValue.ToString();
}
```

3) Voeg in Views/_ViewImports.cshtml: `@using .../EnumHelpers.cs`
4) Gebruik de methode in een view:

```
    <td>
        @EnumHelpers.GetDisplayName(item.Availability)
    </td>
```

Scripts aan einde pagina toevoegen

In de view waarin validatie uitgevoerd wordt (bv. Edit.cshtml), aan het einde, de volgende toevoegen:

```
@section scripts {
    <script src="~/lib/jquery-validation/dist/jquery.validate.js"></script>
    <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js">
    </script>
}
```

DeleteConfirmed

In de DeleteConfirmed moet alles binnen een try-catch-block komen:

```
[HttpPost, ActionName("Delete")]
public IActionResult DeleteConfirmed(int id) {
    try {
        Product product = _productRepository.GetById(id);
        if (product == null)
            return NotFound();
        _productRepository.Delete(product);
        _productRepository.SaveChanges();
        TempData["message"] = $"You successfully deleted product {product.Name}.";
    }
    catch {
        TempData["error"] = "Sorry, something went wrong, the product was not deleted...";
    }
    return RedirectToAction(nameof(Index));
```

Authenticatie

1) In Startup klasse:
   A) In ConfigureServices de volgende toevoegen:

```
AddDefaultIdentity<IdentityUser>().AddEntityFrameworkStores<ApplicationDbContext>();
services.AddAuthorization(options => {
 options.AddPolicy("AdminOnly", policy => policy.RequireClaim(ClaimTypes.Role, "admin"));
 options.AddPolicy("Customer", policy => policy.RequireClaim(ClaimTypes.Role, "customer"));
});
```

B) In Configure de volgende toevoegen (voor `app.useMvc()`):

`app.UseAuthentication();` En de InitializeData()-oproep aanpassen:

`sportsStoreDataInitializer.InitializeData().Wait();`

C) In ApplicationDbContext klasse: klasse laten extenden van IdentityDbContext ipv DbContext

D) In DataInitializer klasse:

D1) DI:

```
private readonly ApplicationDbContext _dbContext;
private readonly UserManager<IdentityUser> _userManager;

public SportsStoreDataInitializer(ApplicationDbContext dbContext,
                                  UserManager<IdentityUser> userManager) {
    _dbContext = dbContext;
    _userManager = userManager;
}
```

D2) Aanmaken van nieuwe users:

```
public async Task InitializeData() {
    ...
    var userName = klant.CustomerName + "@hogent.be";
    await CreateUser(userName, userName, "P@ssword1", "Customer");
    ...
    await CreateUser("admin@sportsstore.be", "admin@sportsstore.be",
                     "P@ssword1", "Admin");
}
private async Task CreateUser(string userName, string email, string password, string role) {
    var user = new IdentityUser { UserName = userName, Email = email };
    await _userManager.CreateAsync(user, password);
    await _userManager.AddClaimAsync(user, new Claim(ClaimTypes.Role, role));
}
```

Layout aanpassen naargelang welke user aangemeld is

in _Layout.cshtml:

```
@using Microsoft.AspNetCore.Authorization
@inject IAuthorizationService AuthorizationService
...
<ul class="nav navbar-nav">
    <li><a asp-area="" asp-controller="Home" asp-action="Index">Home</a></li>
    @if ((await AuthorizationService.AuthorizeAsync(User, "Admin")).Succeeded) {
        <li><a asp-area="" asp-controller="Product" asp-action="Index">Products</a></li>
    }
    ...
</ul>
}
```