Projektuppgift

Datateknik GR(B), ASP.NET med C#

Projektarbete
Lagersystem

Cristina Löfqvist

Mittuniversitetet
MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

MITTUNIVERSITETET
Avdelningen för informationssystem och -teknologi

Författare: Cristina Löfqvist, crlo1900@student.miun.se
Utbildningsprogram: Webbutveckling, 120 hp
Huvudområde: Datateknik
Termin, år: VT, 2021

Sammanfattning

Detta projektet har gått ut på att utveckla en webbaserad och databasdriven applikation i form av ett lagersystem med ASP.NET Core och Core MVC. Resultatet av projektet har blivit en webb-applikation med följande funktionalitet:

- Lagersystem med CRUD funktionalitet för varor, användare och kategorier
- Login funktionalitet
- Varor som finns i lager visas då man ej är inloggad.
- Inloggad användare har full CRUD tillgång för varor, användare och kategorier

Projektet har inneburit att kunskaper kring ASP.NET Core och Core MVC och C# har inhämtats.

Slutprodukten blev en applikation som fungerar som ett lagersystem.

Innehållsförteckning

Förkortningar

EF Core	Entity Frameworc Core
O/RM	Object-relational mapper
MVC	Model View Controller
CRUD	Create Read Update Delete

1 Introduktion

1.1 Bakgrund och problemmotivering

En webbapplikation ska skapas i form av ett lagersysyem med hjälp av ASP.NET.

Applikationen skall vara databasdriven och uppfylla följande krav:

1. Ett webbgränssnitt skall finnas med CRUD funktionalitet för att ändra läsa och skapa data i en underliggande databas

2. Routing inställd så att den startar där den ska när man inte specificerar controller/action i URL

3. Inloggningsfunktionalitet

4. Presentation av lagerförda varor utan att vara en inloggad användare

5. HTML och CSS validerar

6. Applikationen skall följa generella riktlinjer vad det gäller tillgänglighet och användbarhet.

2 Teori

2.1 Wave

Wave står för Web Accessibillity Evaluation Tool och erbjuder ett set med verktyg som hjälper utvecklare att tillgänglighetanpassa innehållet av webapplikationer. Wawe identifierar fel uitfrån Web Content Accesibility Guidelines (WCAG) och presenterar dessa i webbläsarfönstret som ett extra lager utanpå applikationen där brister appliceras som varningsrutor eller moduler, tydligt synliga vid respektive funnet fel [5].

2.2 Entity Framework Core

Entity Framework Core (EF Core) rekommenderas att användas för ASP.NET Core applikationer som hanterar relationsdata [1]. EF Core är en så kallad Object relational Mapper (O/RM) som gör det möjligt att upprätthålla objekt till och från en datakälla och hantera applikationer med avsevärt mindre kod. Det som gör detta möjligt är för att EF Core tillhandahåller funktionalitet för att mappa modeller (klasser) till en databas samt att upprätthålla en session (koppling) mot en databas [1] [2].

2.3 NuGet

NuGet är en pakethanterare för .NET. NuGet möjliggör för produktion och konsumtion av olika paket [6]. För att kunna använda entity framework krävs att man installera en uppsättning olika paket. Dels är det ett paket för server. I detta projektet är det SQLite för mindre applikationer och även SQL server om man behöver bygga ut applikationen till större. Även paketet Microsoft.VisualStudio.Web.CodeGeneration installerades. Det möjliggör att man kan skapa kontroller och vyer automatiskt för modellerna i applikationen. Denna behövs för att kunna skapa ett gränssnitt till databasen [6].

2.4 Model first

Modell first utveckling innebär att man skapar modeller (klasser) och sedan genererar databas-schema från de skapade modellerna [3]. Genom att skapa en klass som utökar DbContext kan man i den klassen definiera DbSet<modell> av den valda modellen man vill generera en databastabell för, relationerna mellan modellerna och sedan de skapade databastabellerna beskrivs i själva modellen i sig. En instans av klassen DbContext representerar en session(koppling) med/ mot databasen [3].

2.5 Schaffolding

Schaffolding kan användas för att skapa MVC design för de olika modellerna som finns implementerade. Schaffolding är ett tillvägagångssätt som man kan

använda sig utav i Visual studio, som kan generera kod för full CRUD funktionalitet för en vald Model mot ett valt DbContext [9].

2.6 MVC

MVC innebär att det finns en Model (M) en eller flera Views (V) och en Controller ©, alltså man kan säga att MVC är en Design struktur. Enligt stycket ovan under rubriken Schaffolding genererars kod för full CRUD funktionalitet för en vald Modell fås flera Views:

1. Create (Create)
2. Delete (Delete)
3. Details (Read)
4. Edit (Update)
5. Index (Read)

Controllern har flera uppgifter:

1. Bestämma vilken View som skall visas (Routing)
    1. Routing till en annan Controller.
2. Hantera anrop till databasen:
    1. Create
    2. Delete
    3. Update
    4. Read

    [10].

2.7 Routing

När man skapar en ny ASP.NET MVC applikation så är routingen redan konfigurerad i form av ASP.NET Routing [8]. ASP.NET Routing modulen mappar inkommande förfrågningar från webbläsaren till specifika MVC controllers åtgärder [8].

2.8 Migration

Så kallad Migration i EF Core är enkelt uttryckt verktyg som kan användas för att skapa och underhålla databaser. Med hjälp av Migrations verktyget i EF Core så kan man uppdatera databas schema så att de matchar applikationens datamodeller samtidigt som existerande data i databasen bevaras. För att använda Migration måste man först installera det via CLI. Detta görs genom kommandot: `dotnet tool install –global dotnet-ef` [4]. Genom att använda argumentet `dotnet ef migration add namnpåMigration` så skapar EF Core en katalog vid namn Migrations i projektet innehållande filer med skript för att generera databas och tabellerna i databasen. Därefter kan man genom EF skapa och uppdatera databasen genom kommandot: `dotnet ef database update` [4].

3 Metod

3.1 IDE

Visual Studio har använts, eftersom Visual Studio tillhanda håller en full utvecklingsmiljö för ASP.NET.

3.2 Entity Framework Core

Entity Framework core används för att generera databas samt för att upprätthålla en session med den genererade databasen, och på så sätt utföra databas operationer [2].

3.2.1 Model first

Model first är en designprincip som innebär att modeller skapats rent programmatiskt för de olika databas entiteterna som identifierades som nödvändiga för att implementera applikationen. Sedan via följande kommandon skapas en databas och modellerna mappas till databas tabeller:

- dotnet ef migrations add InitialCreate
- dotnet ef database update

[3]

3.2.2 Schaffolding

Schaffolding using Entity Framework Core är ett verktyg i IDE visual studio som tillhandahåller generation av en full MVC struktur med CRUD funktionalitet för valda modeller och för valt DbContext. Detta bildar då ett gränssnitt för att kunna skapa, läsa ut, redigera och radera data [3].

3.2.3 Wave

Wave har använts som ett verktyg. Detta kördes i webbläsaren och gav ett mått på hur väl tillgänglighetsanpassad webbsidorna i applikationen var [5].

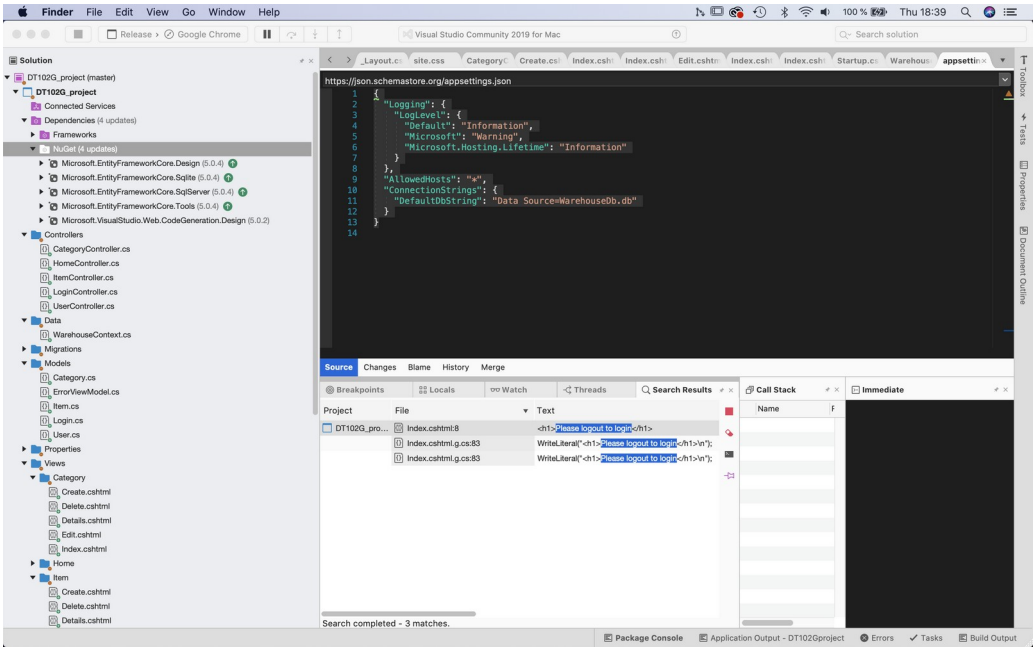
4 Konstruktion

4.1 Entity framework core

Genom användandet av entity framework core har en databas genererats, samt att man använt sig av entity framework för att upprätthålla en session mot den genererade databasen.

4.1.1 Uppsättning av entity framework

För att kunna använda entity framework har en uppsättning med olika paket från pakethanteraren NuGet installerats [6].



Figur 1 pintScreen av NuGet packages.

Figur 1 ovan visar printscreen över vilka paket som finns för applikationen.

4.2 MVC, Model first

Applikationen är byggd enligt designprinceperna MVC och model first.

4.2.1 Model first

De modeller som skapades för applikationen är följande:

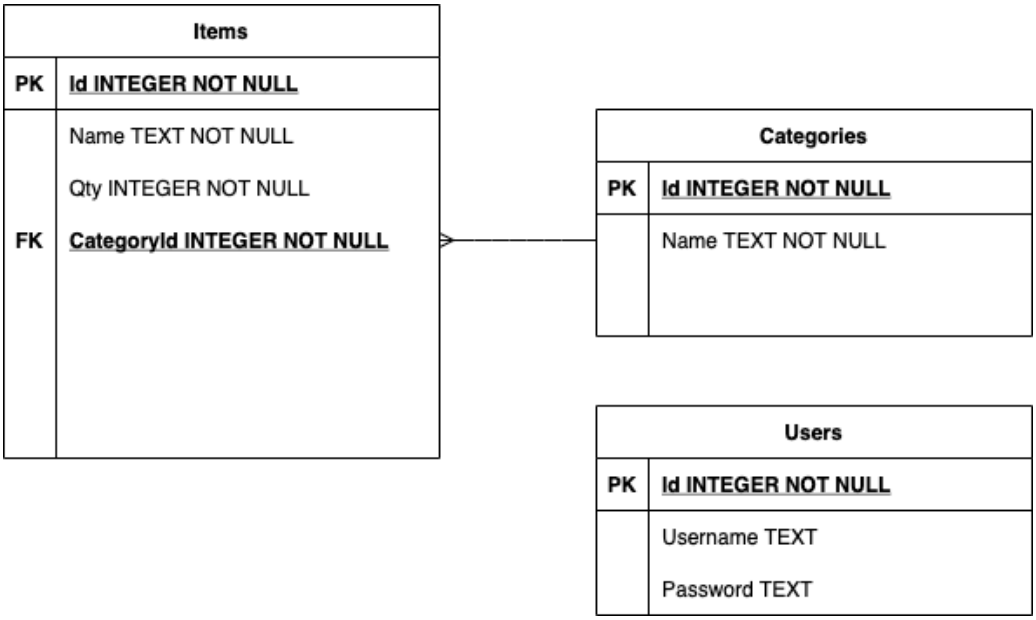
- User

- Item
- Category

Där varje modell är en klass. Här nedan visas modellen för Category

```
using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;



namespace DT102G_project.Models

{

    public class Category

    {

        public int Id { get; set; }

        [Required] public string Name { get; set; }

        public virtual ICollection<Item> Items { get; set; }

        public Category()

        {

        }

    }

}
```

När man senare migrerade modellerna till databasen blev varje modell en egen tabell och varje fält (field) en egen kolumn för den tabellen. Man kunde med hjälp av olika tekniker skapa relationer, unika nycklar, primära nycklar osv för en modell. I ovan exempel kan man se `public virtual ICollection<Item> Items { get; set; }` och i modellen för Item så återfinns `public virtual Category Category { get; set; }`vad denna strukturen gav var ett 1 till många relation från Category till Item, alltså en Category kunde tillhöra många Item men varje Item kunde bara ha 1 category, alltså varje Item måste ha en category och CategoryId blev en foreing key för Item. Se ER-diagrammet här nedan för alla Tabeller i databasen och deras kolumner som skapats via migration av Modellerna (klasserna ovan) och deras relation till varandra.



Databaskoppling, Databas kontext

Genom att skapa en klass som extendar klassen DbContext som innehåller funktionalitet för att upprätta en session med en databas samt att beskriva de tabeller (data modeller) som finns, kunde man beskriva programmatiskt vilka av de skapade modellerna man senare ville migrera till sql databasen samtidigt som man för systemet beskrev vilka modeller skulle som ingå i just denna databassessionen. Detta gjordes på följande vis:

En klass WareHouseContext som extendar klassen DbContext skapades enligt nedan exempel:

```
namespace DT102G_Project.Data

{

    public class WarehouseContext : DbContext

    {

        public WarehouseContext(DbContextOptions<WarehouseContext>
             options) : base(options)

        {

        }

        public DbSet<User> Users { get; set; }

        public DbSet<Item> Items { get; set; }

        public DbSet<Category> Categories { get; set; }
```

```
    }

}
```

Som kan ses här ovan är fieldsen för denna klass 3 stycken med typen DbSet av typen för var och en av de modellerna man önskar ha med i databasen (User, Category och Item), namnet på dessa 3 fields kommer bli namnen på de tabeller som kommer att skapas i sql databasen.[7].

Database context klassen registrerades sedan i startup.cs filen genom att ange vilken databaskontext som skulle användas och även vilken databasserver som skulle användas. Som exemplet nedan visar så registrerades WarehouseContext och databasservern Sqlite. Namnet på anslutningssträngen skickades även med som inställning; `"DefaultDbString"` [7].

```
public void ConfigureServices(IServiceCollection services) {
       services.AddControllersWithViews();

       _ = services.AddDbContext<WarehouseContext>(options =>
options.UseSqlite(Configuration.GetConnectionString("DefaultDbString"))
);

}
```

Anslutningssträngen registrerades sedan i filen appsettings.json för att utgöra ett landmärke till databasuppkopplingen. Den innehöll filen `"Data Source=WarehouseDb.db"` till Sqlite servern.

```
  "ConnectionStrings": {

    "DefaultDbString": "Data Source=WarehouseDb.db"

  }

}
```

[7].

Som beskrivet ovan har allt arbete först gjorts programmatiskt, man har beskrivit de olika tabellerna programmatiskt genom att skapa klasser (modeller) först, alltså designprincipen model first.

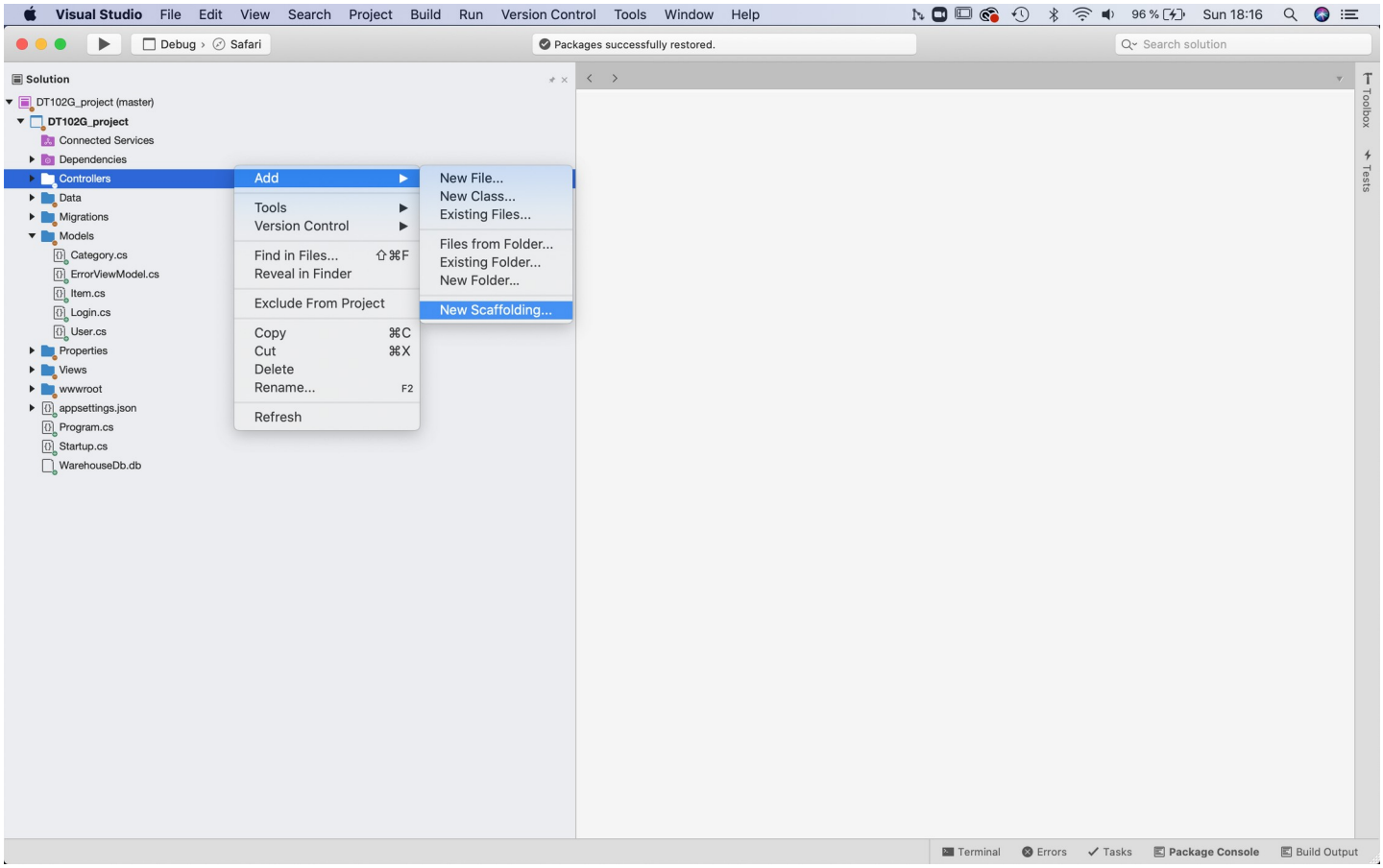
4.2.2 MVC

MVC står för model view control och denna designprincipen finns beskriven i kapitel 2.6. För att skapa en MVC struktur för applikationen gjordes följande:

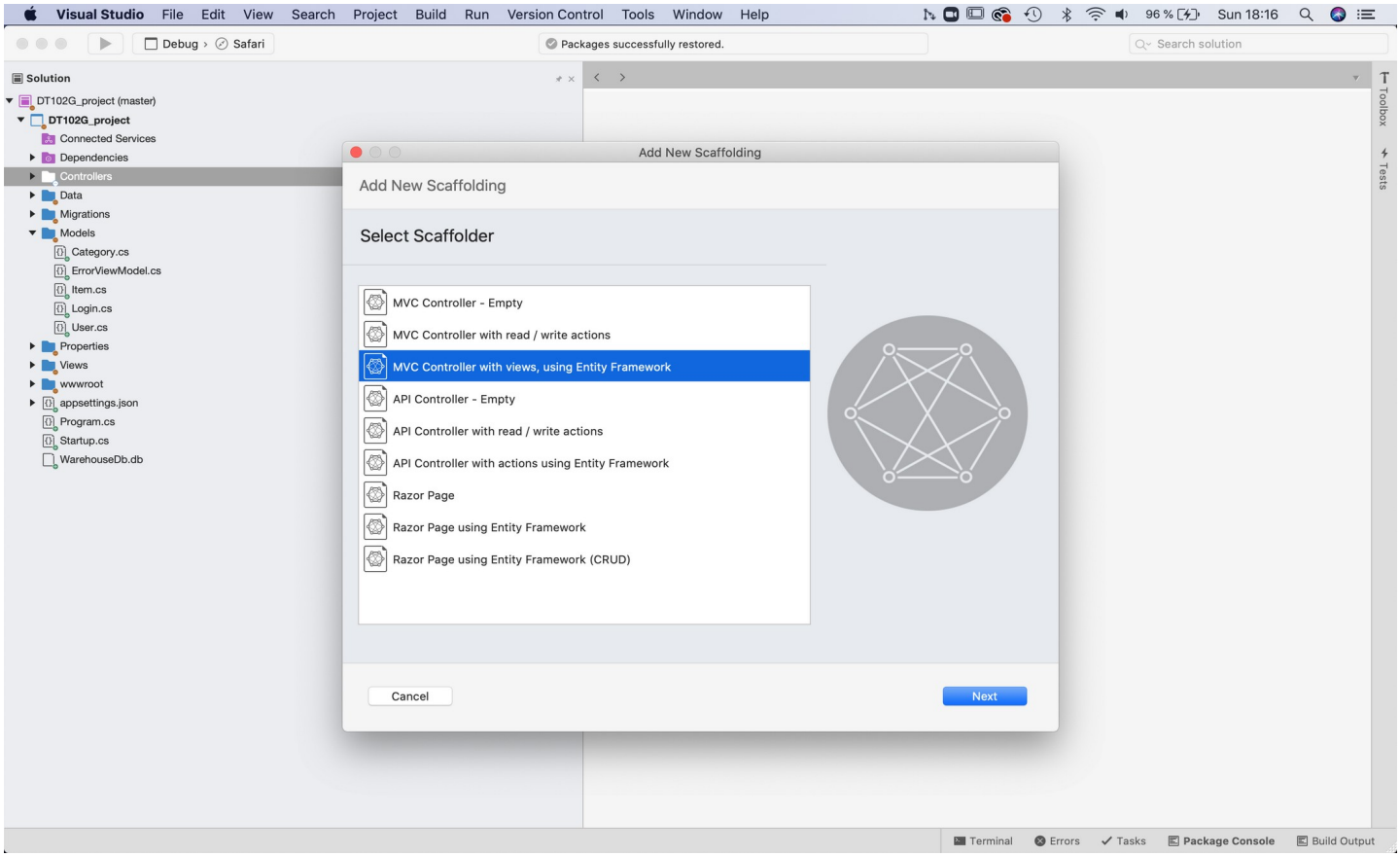
Först skapades modellerna (beskrivet i kapitel 4.2.1)

Sedan genom scaffolding (se kapitel 3.2.2) skapades controller och vyer för CRUD mot modellerna [9].

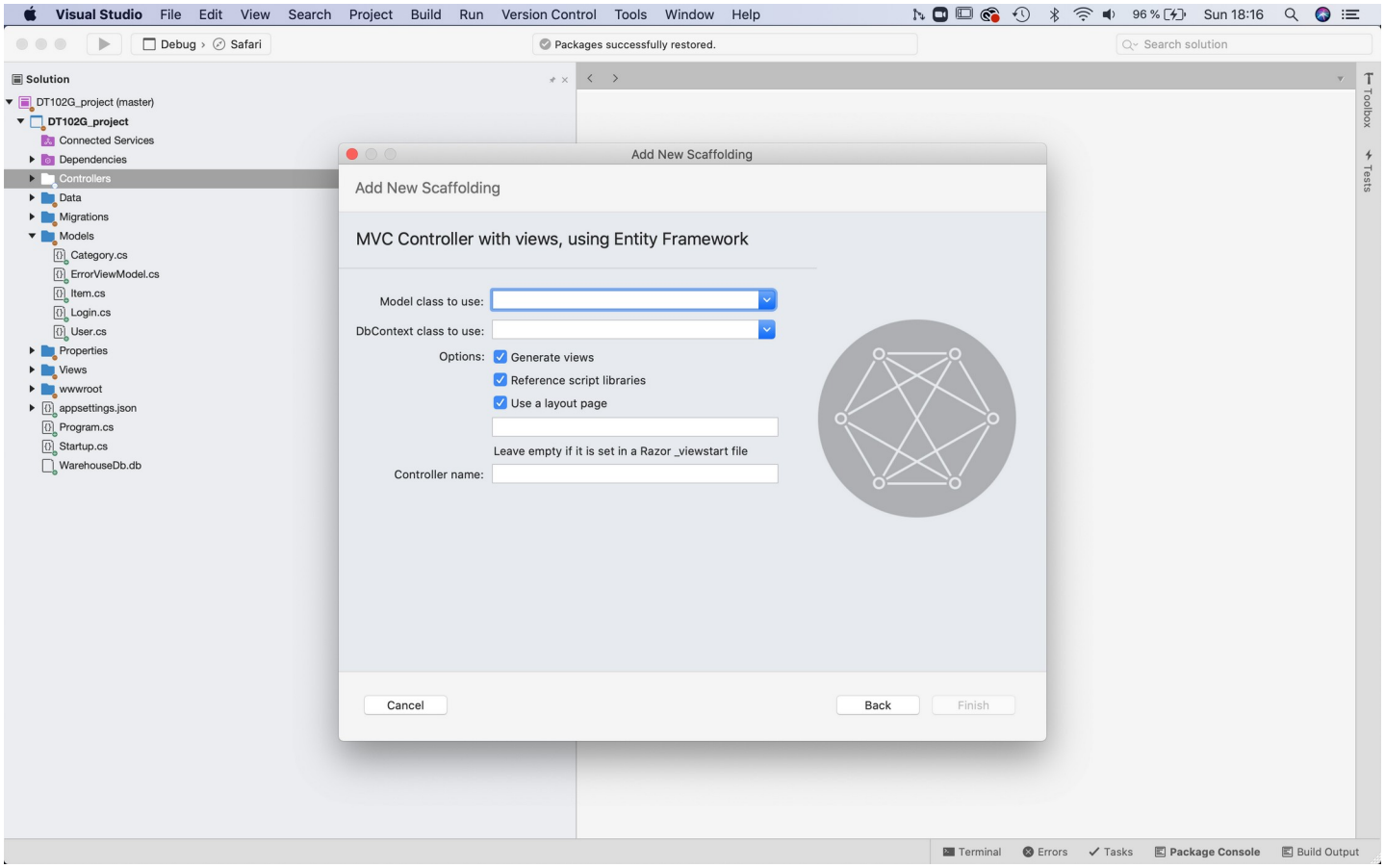
Nedan ses en bildserie (figur 2-4) över hur detta genomfördes.



Figur 2 PrintScreen över tillägg av ny Scaffolding.



Figur 3 Printscreen över MVC val Entity Framework.



Figur 4. Printscreen över val av vilken modell klass man vill generera MVC struktur för.

I sista steget i figur 4 valdes vilken modell klass man ville generera MVC struktur för, samt vilken DbContext klass som skulle användas och även namnet på Kontroll klassen [9].

4.3 Migration

Databasen skapades sedan genom så kallad Migration med hjälp av EF Migration verktyget [4]. En katalog för migrations skapades genom att ange i terminalen: `dotnet ef migrations add InitialCreate`. Databasen skapades sedan genom kommandot: `dotnet ef database update` [4].

4.4 Validering och tillgänglighet

4.4.1 Validering

HTML koden som genereras från applikationen samt CSS som appliceras på den genererade html koden validerades genom https://validator.w3.org/#validate_by_input samt via https://jigsaw.w3.org/css-validator/.

4.4.2 Tillgänglighet

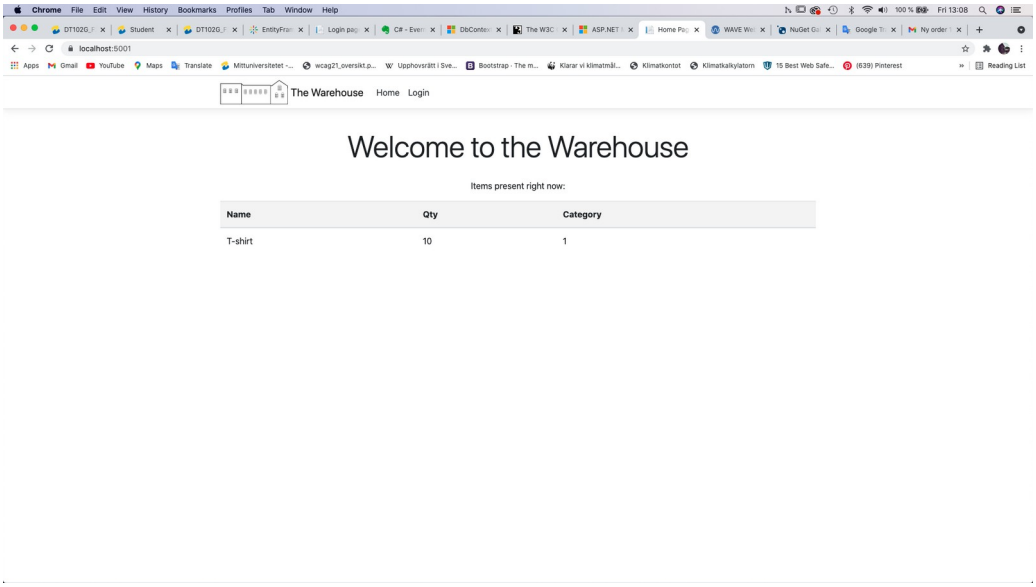
Alla undersidor testades med Wave för att se till att Wave inte rapporterade några fel.

5 Resultat

Resultatet blev ett fullt fungerande lagersystem som är en databasdriven applikation. Den resulterande applikationen uppfyller de krav som finns beskrivna i kapitel 1.1.

5.1 Databasdriven, CRUD, Inloggningsfunktionalitet

Applikationen som utvecklats har en startsida där produkter som finns i lager visas i form av en lista vilket uppfyller krav 4 specificerat i kapitel 1.1.



Figur 5 printscreen över startsida.

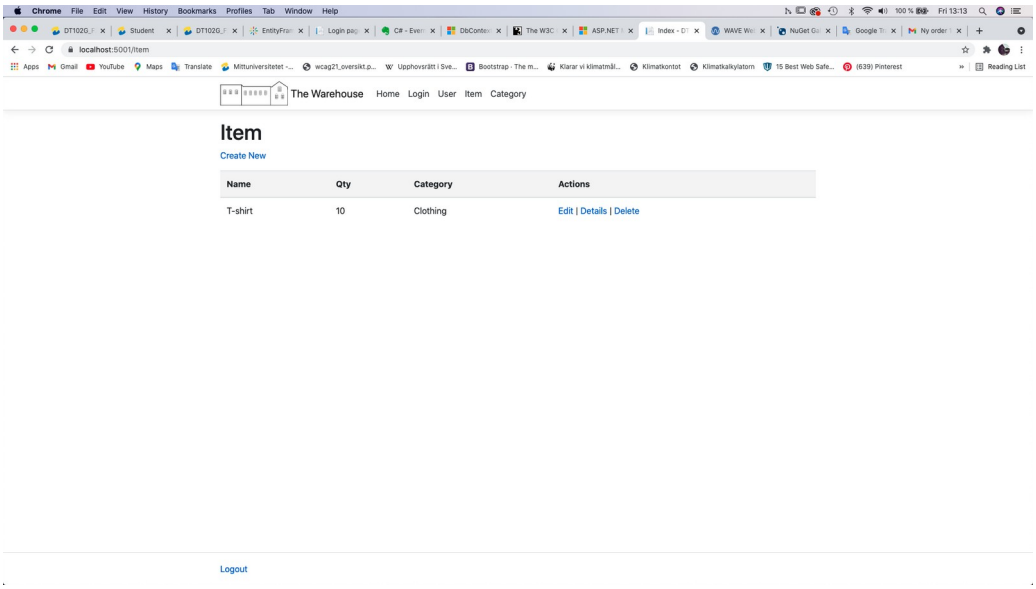
En inloggningssida har utvecklats och efter att en användare har loggat in via denna sidan, har man möjlighet att ändra, läsa och skapa ny data för alla databasentiteter som finns i applikationen vilket uppfyller kraven 1 och 3 specificerat i kapitel 1.1. De entiteter som finns i applikationen är följande:

- User
- Item
- Category

Mellan Entiteten category och item finns en relation som är följande:

Ett item kan ha en category och en category kan tillhöra flera olika items, alltså en one to many relation för category to item.

Den färdiga applikationen har en undersida för respektive databasentitet. Dessa undersidor går att nå via navigationen som är placerad i headern av applikationen se figur 6 nedan:

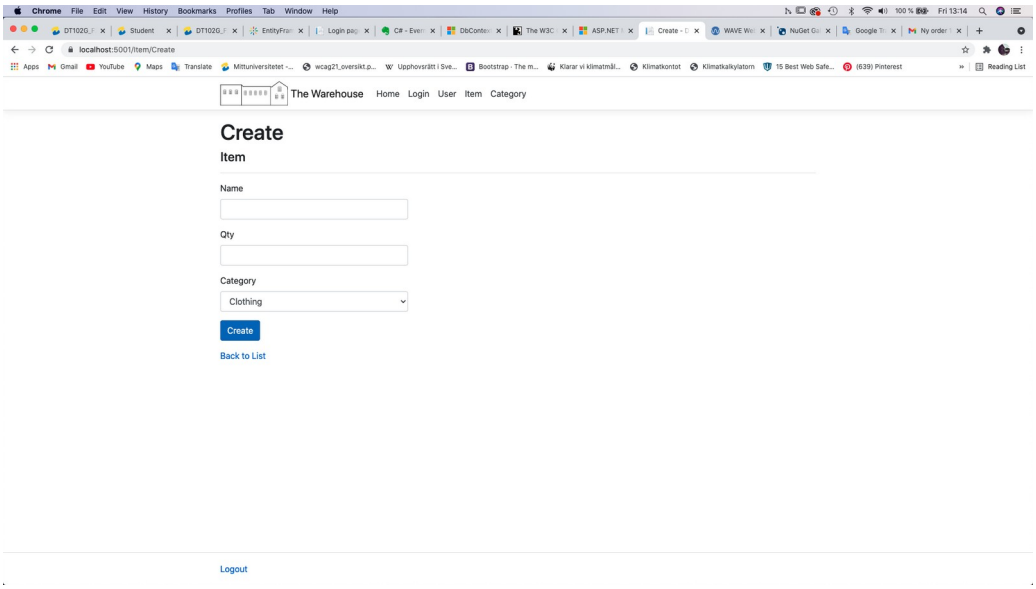


Figur 6. printscreen över undersida för Item.

I bilden ovan ser man headern med navigationen samt undersidan för Item.

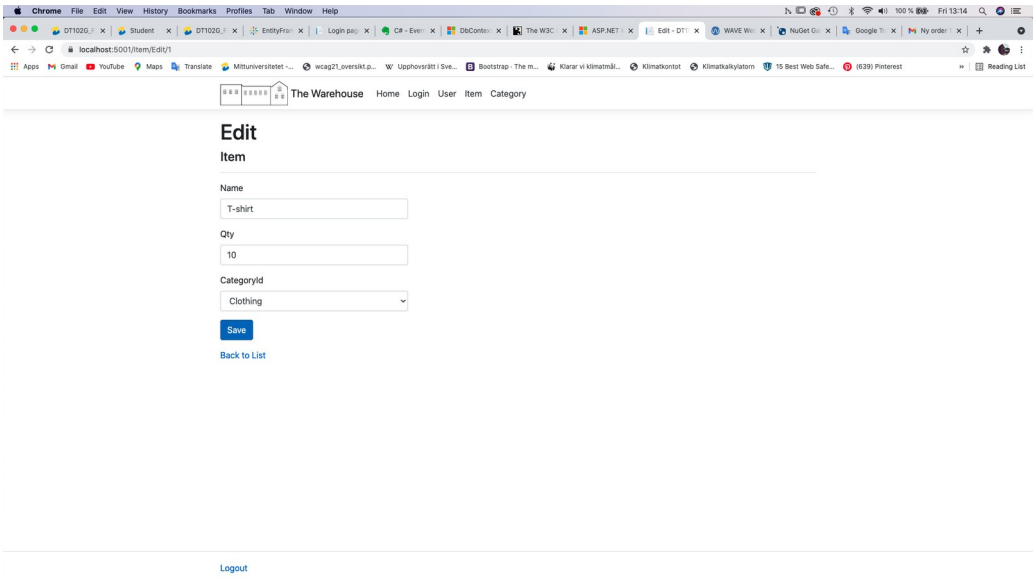
Varje undersida likt den som visas för databasentiteten Item har funktionalitet för att Skapa ett nytt item, editera ett redan närvarande item, läsa detaljer om ett redan närvarande item samt att ta bort ett item i listan.

För skapa ett nytt item eller någon av de andra databasentieterna som finns i applikationen finns det en undersida med följande utseende:



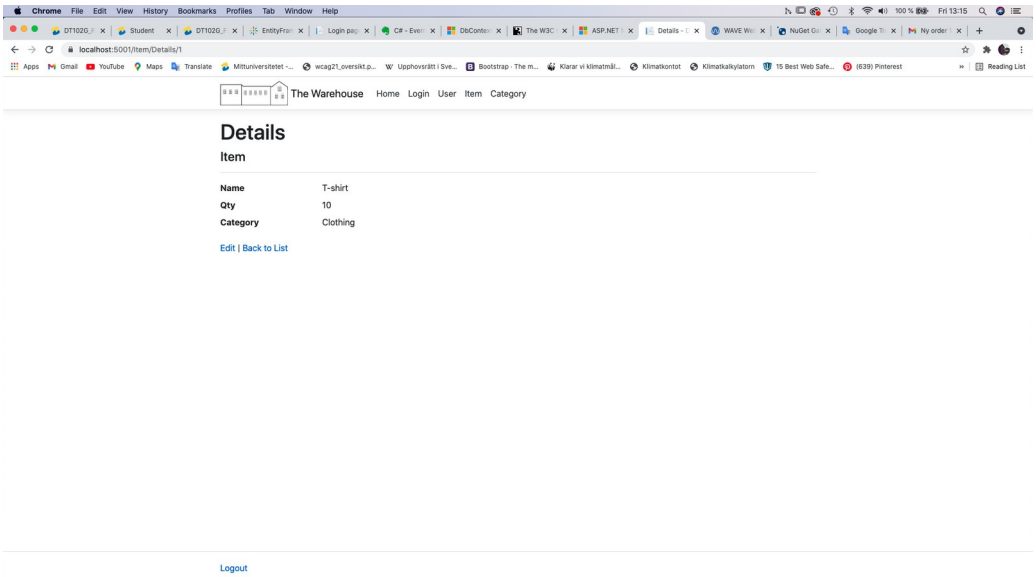
Figur 7 printscreen över undersida för att skapa nya item.

För att editera ett Item,User eller Category finns en undersida med följande utseende:



Figur 8 print screen över undersida för att redigera item.

För att läsa detaljer för ett Item,User eller Category finns en undersida med följande utseende:



Figur 9. Printscreen över undersida för detaljer.

5.2 Routing

Routingen fungerar på så vis att den startar där den ska när man inte specificerar controller/action i URL vilket uppfyller krav 2 specificerat i kapitel 1.1 [8].

5.3 Validering, Tillgänglighet

5.3.1 Validering

All html som genereras av applikationen samt den css som appliceras på den genererade html koden validerar enligt https://validator.w3.org/#validate_by_input samt via https://jigsaw.w3.org/css-validator/ utan errors vilket uppfyller krav 5 specificerat i kapitel 1.1.

5.3.2 Tillgänglighet

Genom användandet av verktyget Wave säkerställdes det att alla undersidor för applikationen passerade Waves tester utan några varningar eller fel på så sätt säkerställdes det att applikationen följer generella krav på tillgänglighet vilket uppfyller krav 6 specificerat i kapitel 1.1 [5].

6 Slutsatser

Genom att använda de valda metoderna och verktygen kan det konstateras att det på ett snabbt och smidigt sätt, kan skapas en webbapplikation med både bakomliggande databas och logik, tillsammans med en front-end som använder sig av de bakomliggande funktionaliteterna och databas.

De problem som uppkom under projektets gång var följande:

- Ett kopplat till hur relationer skapas mellan de olika modellerna. Det var tänkt att ett Item skulle kunna ha flera kategorier men detta uppnåddes aldrig, det slutade med en one to many relation med en obligatorisk foriegn key i Item mot Category alltså varje item kan endast ha en kategori medans en kategori kan tillhöra många olika items.

- Det uppstod även ett problem när modellerna skulle migreras till sql databasen, att relationen beskriven i modellerna Item och category skapades aldrig, det var oklart varför detta hände, för då databasen togs bort och migrationen gjordes om så fick man väntat resultat.

Något som skulle kunna vara ett ämne för vidareutveckling är att man tillåter ett Item ha flera olika kategorier.

Källförteckning

[1] Microsoft ”Working with Data in ASP.NET Core Apps” https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/work-with-data-in-asp-net-core-apps Publicerad: 2020-01-12. Hämtad: 2021-03-21.

[2] Entety Framework Tutorial ”What is Entety Framework?” https://www.entityframeworktutorial.net/what-is-entityframework.aspx Publicerad: 2020. Hämtad: 2021-03-21.

[3] Microsoft ”Model First” https://docs.microsoft.com/en-us/ef/ef6/modeling/designer/workflows/model-first publicerad: 2016-10-23. Hämtad: 2021-03-21.

[4] Microsoft ”Migrations Overview” https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli publicerad: 2020-02-28. Hämtad: 2021-06-03.

[5] Wave ”Wave Web Accessibillity Evaluation Tool” https://wave.webaim.org/ Hämtad: 2021-06-04.

[6] nuget ”what is NuGet?” https://www.nuget.org/ Hämtad: 2021-06-04.

[7] Microsoft ”DbContext Lifetime, Configuration, and Initialization” https://docs.microsoft.com/en-us/ef/core/dbcontext-configuration/ Publicerad: 2020-07-11. Hämtad: 2021-06-04.

[8] Microsoft ”ASP.NET MVC Routing Overview (C#)” https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/controllers-and-routing/asp-net-mvc-routing-overview-cs Publicerad: 2008-08-19. Hämtad: 2021-06-04.

[9] Microsoft ”ASP.NET Scaffolding in Visual Studio 2013” https://docs.microsoft.com/en-us/aspnet/visual-studio/overview/2013/aspnet-scaffolding-overview publicerad: 2014-09-04. Hämtad: 2021-06-04.

[10] Microsoft ”ASP.NET MVC Pattern” https://dotnet.microsoft.com/apps/aspnet/mvc Hämtad: 2021-06-04.