

# SayHi: En webbshop

Ett projekt skapat med ASP.NET Core MVC, JavaScript och SASS

Charlotte Ranemo

**MITTUNIVERSITETET**

**Institutionen för Informations-system och teknologi**

**Examinator:** Mikael Hasselmalm, [Mikael.Hasselmalm@miun.se](mailto:Mikael.Hasselmalm@miun.se)

**Handledare:** Mattias Dahlgren, [Mattias.Dahlgren@miun.se](mailto:Mattias.Dahlgren@miun.se)

**Författare:** Charlotte Ranemo, [chra1901@student.miun.se](mailto:chra1901@student.miun.se)

**Utbildningsprogram:** TWEUG, Webbutveckling, 120 hp

**Huvudområde:** Datateknik

**Termin, år:** VT, 2021

## Sammanfattning

Syftet med detta arbete var att ta fram en webbshop som möjliggjorde för mig som ägare av SayHi att sälja egengjorda gratulationskort. Genom att skapa en administratörsdel skulle det gå för registrerade administratörer att lägga till, redigera och ta bort gratulationskort som är till salu, samt att göra detsamma med de kategorier av kort som ligger uppe på siden. Själva webbshoppen skulle ha ett fungerande flöde, från försäljning och visning av kort till genomförd beställning och e-postbekräftelse till den e-post som kunden angivit. Denna webbplats och alla dess funktioner skapas med ASP.NET Core MVC som backend, samt SASS, HTML5 och JavaScript som frontend. SayHi är namnet som applikationen tilldelades, och ska ligga tillgänglig på en domän med samma namn. Målet med applikationen är att skapa en webbshop som fungerar fullt ut, men med faktura som enda tillgängliga betalmedel. Webbplatsen ska se attraktiv ut för besökare, och den ska validera och genomgå användartester innan den anses vara färdig. Resultatet av detta arbete är en fungerande webbshop som möjliggör försäljningen av gratulationskort på ett användarvänligt sätt.

**Nyckelord:** Webbdesign, ASP.NET, JavaScript, E-handel

## Abstract

The purpose of this website was to create a platform for the company SayHi to sell their selfmade greeting cards. By creating an administrator part of the website it will be enabled for administrators to log in and add new cards for sale, as well as categories and new users. The webshop will be fully functioning and have invoice as the option for payment, and the customer will receive an order confirmation by email after a purchase has been done, and these purchases will be kept on record and will be displayable for the administrators. The website will be made using ASP.NET Core MVC, SASS, HTML5 and JavaScript and will be coded in Visual Studio 2019. By the end of this project, we will have a functioning webshop with a user-tested design that speaks to the customers and enables them to buy greeting cards online. The result of this project is a functioning e-commerce store that enables the selling of self made greeting cards in a user friendly manner.

**Keywords:** Web design, ASP.NET, JavaScript, E-commerce

# Innehållsförteckning

<b>Sammanfattning.....</b>	<b>2</b>
<b>Abstract.....</b>	<b>3</b>
<b>Terminologi.....</b>	<b>6</b>
<b>1 Introduktion.....</b>	<b>1</b>
1.1 Bakgrund och problemmotivering.....	1
1.2 Övergripande syfte.....	2
1.3 Avgränsningar.....	2
1.4 Konkreta och verifierbara mål.....	2
1.5 Översikt.....	4
<b>2 Teori.....</b>	<b>5</b>
2.1 ASP.NET Core MVC.....	5
2.2 Synthetically Awesome Style Sheets.....	6
2.3 Entity Framework.....	6
2.4 Scaffolding.....	6
<b>3 Metod.....</b>	<b>7</b>
3.1 Förarbete.....	7
3.2 Utveckling.....	7
3.3 Design.....	8
3.4 Användartester.....	8
3.5 Nuget-paket och utvecklingsmiljö.....	9
<b>4 Konstruktion.....</b>	<b>10</b>
4.1 Sitemap.....	10
4.2 Wireframes.....	10
4.3 Moodboard.....	11
4.4 ER-diagram.....	11
4.5 Skapandet av startsidan.....	12
4.6 Administratörsgränssnittet.....	13
4.7 Inloggningsfunktionalitet.....	15
4.8 Design av administratörsgränssnittet.....	16
4.9 Gränssnitt för att visa ordrar.....	16
4.10 Webbshoppens uppbyggnad.....	17
4.11 Kundvagnen.....	18
4.12 Checkout.....	20
4.13 Email-bekräftelse.....	22
4.14 Den responsiva designen.....	22
4.15 Funktionalitetstestning och användartestning.....	23
<b>5 Resultat.....</b>	<b>25</b>
<b>6 Slutsatser.....</b>	<b>26</b>
6.1 Analys och resultatdiskussion.....	26

6.2	Projektmetod diskussion.....	26
6.3	Etisk och social diskussion.....	26
	<b>Källförteckning.....</b>	<b>28</b>
	<b>Bilaga A: Wireframes.....</b>	<b>30</b>
	Mobil-startsida.....	30
	Desktop-startsida.....	31
	Desktop-alla kort.....	32
	Desktop-enskilt kort.....	32
	Mobil-enskilt kort.....	33
	Mobil-alla kort.....	34

# Terminologi

## Förkortningar

MVC	Model-View-Controller. Ett designmönster för utveckling av .NET applikationer.
SASS	Syntactically Awesome Style Sheets. En CSS preprocessor.
EF	Entity Framework. Ett .NET ramverk för att skapa databaser.
SQL	Structured Query Language. Ett språk som kan göra frågor (så kallade "queries") till en databas.
GDPR	General Data Protection Regulation. En förordning som bestämmer hur man ska behandla personuppgifter inom EU.
.NET	Ett C# ramverk.
ER-diagram	Entity relationship diagram. Ett schema för att visa på hur en databas ska byggas upp.

# 1 Introduktion

## 1.1 Bakgrund och problemmotivering

I tider som dessa där social distansiering är så pass viktigt som det är, så välkomnar vi gärna nya sätt att underhålla våra relationer till andra. Trots svåra tider, så kommer det fortfarande att finnas tillfällen som vi vill fira, och det är där SayHi kommer in på marknaden. SayHi är ett start-up företag som har som mål att sälja egengjorda gratulationskort, både fysiska och digitala, till privatpersoner i Sverige. Gratulationskortet ska presenteras på den webbplats som tillhör SayHi, och det är den webbplatsen som denna rapport ska behandla.

Företaget SayHi har i nuläget ingen webbplats, utan förlitar sig helt på utvecklingen som kommer att dokumenteras i denna rapport. SayHi vill ha en plattform för e-handel som tilltalar så många som möjligt, och som via ett lättanvänt användargränssnitt för administratörer tillåter uppladdning av nya objekt för försäljning som genast ska synas i själva webbshoppen.

Det är administratörerna själva som ska skapa alla de kort som ska säljas på SayHi, och detta kommer att göras allt eftersom att webbplatsen växer. Genom att ta mig an uppdraget att skapa SayHis webbplats, så läggs det på mig att utveckla en applikation som fungerar både nu och i framtiden om webbplatsen växer.

Då det i dessa tider är extra viktigt att komma på sätt att umgås på distans, så kommer SayHi in på marknaden som ett företag som vill göra just det – underlätta för människor att gratulera varandra på bemarkelsedagar och på så sätt upprätthålla vänskaper som tagit stryk av social distansiering. Genom att skapa denna webbplats hoppas jag att fler människor når ut till sina nära och kära då det via SayHis webbplats ska gå enkelt och snabbt att göra detta.

SayHi ska därmed vara en lättanvänd e-handelsplattform som är behaglig att besöka, både för administratörer och för kunder. Det ska gå att lägga beställningar för kunder och att underhålla webbplatsen för administratörer, och detta ska göras på ett så användarvänligt sätt som möjligt – något som ska säkerställas med utförliga användartester och funktionstester.

I slutet av detta projekt ska SayHi stå redo att ta sig an den kundbas som söker efter gratulationskort, genom en webbplats som utvecklas av mig med användarvänlighet i fokus.

I dagar som dessa när mycket sker online, inte minst shopping, så kommer start-up företaget SayHi in på marknaden som en e-handelsplattform som säljer gratulationskort, både fysiska och digitala, som enkelt kan skickas till nära och

kära. SayHi behöver för detta ändamål en ny webbplats, och det är där denna rapport kommer in i bilden. Här kommer den dokumentation att skrivas som leder fram till en ny, välfungerande webbplats som tillhör SayHi.

## 1.2 Övergripande syfte

Detta projekt syftar till att ta fram en välfungerande e-handelsplattform åt företaget SayHi som möjliggör försäljningen av gratulationskort för alla tillfällen.

## 1.3 Avgränsningar

Detta projekt fokuserar på att få fram en välfungerande webbplats som fungerar väl både för kund och för administratör. Inom den angivna tidsramen finns inte tid att utveckla ett fungerande betalsystem, så vid slutet av denna tidsperiod kommer endast betalningsalternativet ”faktura” att finnas, och inget faktiskt sätt att betala med kort, Paypal eller övrigt. Vid vidareutveckling av applikationen efter det att de mest nödvändiga funktionerna har utvecklats, så kommer SayHi själva att utveckla applikationen så att övriga betalsätt går att genomföra.

Fakturorna och varorna kommer inte heller att skickas ut, då SayHi främst fokuserar på att få en fungerande webbplats. Logistiken kommer de att lösa utanför tidsramarna för detta projekt.

Ingen GDPR-popup kommer att skapas, trots att cookies sparas (såsom användarens e-post). Detta kommer att läggas till vid vidareutveckling av projektet.

## 1.4 Konkreta och verifierbara mål

Huvudmålet, och även det övergripande målet med detta projekt är att skapa en webbplats som är väl fungerande och som möjliggör försäljning av gratulationskort. För att nå fram till detta mål på ett så bra sätt som möjligt följer nedan de mål som ska uppfyllas vid slutet av denna projektperiod:

Förarbete:

- Sitemap som beskriver webbplatsens sidor.
- Wireframes som visar huvudsidorna på både desktop och mobil.
- Flödesscheman som beskriver flödet i applikationen.
- ER-diagram som beskriver databasens utformning.

Utveckling:



- Tilltalande startsida som är landningssidan för webbshoppen.
- Undersidor som möjliggör visning av upplagda kort, både på kategori-nivå och på enskild nivå, vilket ska innehålla en lägg till i kundvagn-knapp.
- En kundvagn som ska visa de objekt som lagts till i kundvagnen av kund.
- Statiska sidor som beskriver kontaktmöjligheter med företaget samt köpvillkor.
- Administratörssidor för att underhålla administratörsregistret: skapa, ta bort och visa alla.
- Administratörssidor för att underhålla kortregistret samt kategoriregistret: skapa, ta bort, redigera, visa detaljer och visa alla. Det ska gå att sätta pris på alla kort, samt sätta valfria kategorier på rea.
- Två layout-sidor, en för administratörer och en för kund-sidorna, som sedan ska tilldelas rätt sidor.
- Checkout-sida som möjliggör för kund att genomföra en beställning.
- E-post bekräftelse ska sedan skickas till kund.

#### Design:

- Designen ska skapas med SASS-kod och övrig frontend-funktionalitet ska skapas med JavaScript i en separat fil.
- Webbplatsen ska valideras, samt ska genomgå noga testning genom användartester över Zoom samt funktionalitetstestning av mig själv. Detta för att säkerställa att designen på sidan är tilltalande och användarvänlig, samt för att se så att alla funktioner fungerar som det är tänkt.

Applikationen kommer att byggas med ASP.NET Core MVC som backend, och allt som sker på backend kommer därmed att ha sin grund i språket C#. För frontend kommer som sagt SASS, HTML5 och JavaScript att användas, vilket ger stor potential för en ypperlig design som även är tänkt att vara enkel att underhålla. Funktionerna som skapas i JavaScript ska döpas till beskrivande namn och SASS ska användas och kompileras till CSS-kod för att kunna visas i webbläsaren.

## 1.5 Översikt

Kapitel 2 beskriver teorin som är bra att ha koll på vid fortsatt läsning av denna rapport. Kapitel 3 kommer att beskriva metoderna som används för att nå fram till alla de mål som beskrivits i detta kapitel. Kapitel 4 beskriver hur konstruktionen av webbplatsen gick till och i kapitel 5 beskrivs resultatet av detta projekt. Mina egna slutsatser och diskussion presenteras sedan i kapitel 6, slutsater.

## 2 Teori

### 2.1 ASP.NET Core MVC

Det främsta ramverket som användes för att bygga detta projekt var ASP.NET Core MVC, vilket är ett ramverk som använder model-view-controller (MVC) designmönstret. Det betyder att applikationen delas upp i delar, nämligen så kallade models, views och controllers, som alla spelar en roll i slutresultatet [2]. Se figur 1 för en översikt över hur de olika delarna relaterar till varandra.

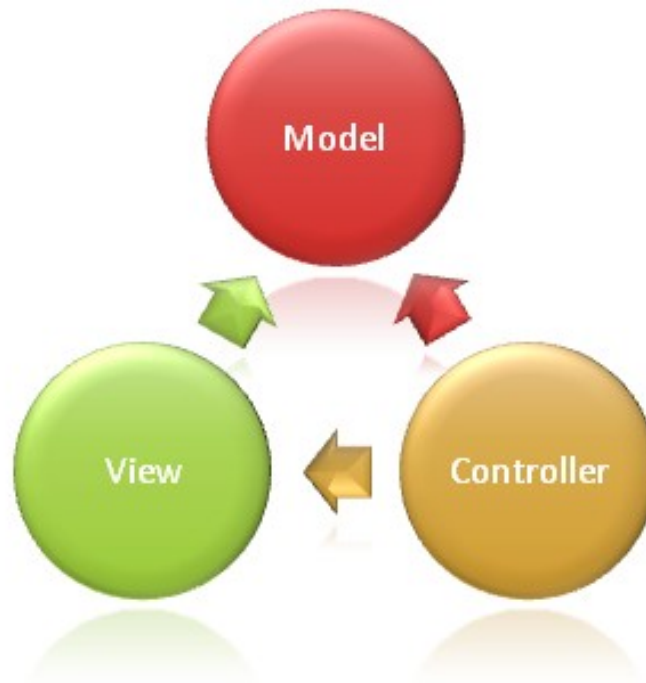


Figure 1: MVC-mönstret [2]

Views (vyer) är det användarna ser och det består av både C#-kod och HTML, där C#-koden blir renderad till HTML. Models (modellerna) beskriver till exempel databasens skelett när man använder sig av Entity Framework, som jag kommer att göra i detta projekt (se 2.3 för mer information om detta). Controllers (kontrollerna) tar hand om vad som händer när användaren gör vissa saker på webbplatsen, såsom att posta ett formulär, eller logga in. Det är i dessa controllers som C# kod läggs till för att kontrollera event och rendering i vyerna [2].

## 2.2 Synthetically Awesome Style Sheets

Synthetically Awesome Style Sheets (SASS) är ett förlängningsspråk till CSS. Den kod man skriver som SASS-kod kompileras om till CSS kod innan publicering till webbläsaren, och godtar många saker som inte går att använda i vanlig CSS kod såsom variabler och funktioner. Webbläsare kan inte ännu läsa SASS-kod, utan måste som sagt kompileras om innan det kan visas där. Som tur är finns det många sätt att kompilera om SASS till webbläsarvänlig CSS [3].

## 2.3 Entity Framework

Entity Framework är en “object-relational mapper” (O/RM) och används av utvecklare för att generera fram automatiserad kod med basen i .NET-objekt och skapar en databas baserad på dessa. Entity Framework stödjer både “code-first” och “database-first” sätt att skapa databas på, med det första menas att man skapar en modell först och sedan genererar databasen utifrån denna, och den andra betyder att databasen skapas först och context-klasser skapas i andra hand [4].

För att göra queries till databasen när man använder Entity Framework så använder man Language Integrated Query (LINQ) vilket är ett sätt att skapa queries från olika typer av källor, som till exempel SQL Server, men även för till exempel XML dokument [5].

Man kan använda många olika databastyper när man använder Entity Framework, och denna rapport kommer att behandla MSSQL (Microsoft SQL) eller SQL Server som det också kallas. Man kan dock använda andra typer av relationsdatabaser eller även NoSQL (non SQL/not only SQL) såsom MongoDB när man jobbar med Entity Framework [6].

## 2.4 Scaffolding

ASP.NET Scaffolding är ett ramverk som genererar kod till applikationer som använder sig av ASP.NET. Man scaffoldar kod till ett projekt när man vill lägga till kod som genereras automatiskt till sitt projekt, som interagerar med ens datamodeller (models) [8].

För att scaffolda in till ett projekt så högerklickar man på en mapp i projektet eller på själva projektet och väljer “Add→New scaffolded item”. Efter det får man lägga till några val såsom vilken modell man vill använda, och efter detta så kommer scaffolding-ramverket, om allt lyckas, att generera kod baserat på dessa val. Det är detta som kallas “scaffolding” [8].

## 3 Metod

### 3.1 Förarbete

För att nå upp till de mål som är satta kommer noggrant förarbete att göras. Detta kommer att påbörjas genom att en sitemap skapas, för att lägga upp en plan på vilka sidor som måste skapas. Efter detta skapas wireframes genom programmet/webbsidan wireframe.cc [7] för flera av dessa sidor för att få en generell idé över hur layouten på sidan ska vara. För att sedan få en känsla för färg och form kommer en moodboard att skapas som innehåller logotypen och möjliga färger samt typsnitt. Ett ER-diagram kommer sedan att skapas för att få en tydlig bild av hur databasen ska vara uppbyggd, för att minimera risken för misstag.

### 3.2 Utveckling

- För utveckling av startsidan kommer den wireframe som skapats under förarbetet och som visar på startsidan att användas. Denna kommer sedan att kodas som en vy, vid namn Index.cshtml.
- Efter att startsidan konstruerats, kommer administratörsgränssnittet att skapas. Detta kommer att ske genom att basera de modeller som ska skapas på ER-diagrammet som skapats under förarbetet, som sedan genom att använda Entity Framework och scaffolding kommer att konstruera en databas med tillhörande vyer. Dessa vyer ska sedan redigeras så att layout och design passar in den som planerats under förarbetet. Detta kommer baseras på ASP.NET Core MVC för backend och SASS för frontend.
- Efter detta kommer några objekt att läggas till i webbshoppen genom det administratörsgränssnitt som har skapats. Dessa objekt kommer jag att ha skapat själv i Photoshop. Dessa kommer vara det första jag gör när det kommer till utveckling, för att jag ska ha något att visa på startsidan. Övriga bilder som ska vara på startsidan kommer även dessa att redigeras i Photoshop.
- När objekten är tillagda i webbshoppen kommer själva gränssnittet för att visa dessa i webbshoppen att konstrueras, både på kategorinivå och per enskilt kort. På den enskilda kort-sidan ska en knapp som kodas med JavaScript att läggas till i ett formulär, där man kan välja om det är ett fysiskt eller ett digitalt kort man vill köpa. Vid klick på denna knapp kommer sedan kortet att läggas till i varukorgen, och numret som står bredvid Kundvagn i huvudmenyn kommer att, med JavaScript, uppdateras till det rätta antalet varor i kundvagnen.

- Själva kundvagns-sidan ska skapas så att man under två olika rubriker (digitala kort och fysiska kort) kan se vilka objekt man hittills lagt till i varukorgen. Här ska det finnas möjlighet att ta bort objekt som ligger i varukorgen, eller att tömma hela. Det ska även gå att gå vidare till checkout.
- Order som läggs via checkout ska sparas till databas och finnas tillgängliga för visning via administratörsgränssnittet.
- När en kund gått igenom checkout kommer ett bekräftelsemejl att skickas automatiskt till den e-post adress som kunden angivit i checkout-fältet för faktura-adressen. Som sagt kommer endast faktura att vara det val man kan göra på betalning. Ingen faktura kommer dock att skickas ut, utan enbart bekräftelse på att man gjort en order.
- Två statiska sidor kommer att skapas, som endast fyller funktionen att möjliggöra för presentation av kontaktuppgifter till företaget samt köpvillkor.

### 3.3 Design

Designen kommer att skapas med SASS-kod som sedan kommer att kompileras om till CSS kod via ett Nuget-paket (se 3.5). För att säkerställa att webbplatsen är användarvänlig och fungerar som det är tänkt samt har en tilltalande design, så kommer tester att utföras. Dels kommer koden att valideras för att se till så att den är så tillgänglig som möjligt, och dels så kommer jag själv att testa alla funktioner på siten innan publicering. Även användartester kommer att göras, för att se till så att inte enbart mitt eget omdöme bestämmer hur webbplatsens slutstadium ser ut, utan istället att jag får andras input. Se 3.4 Användartester.

### 3.4 Användartester

Under två zoom-intervjuer kommer jag att fråga två användare följande frågor för att se hur jag kan förbättra webbplatsen, främst när det kommer till design och layout.

- Vad gillar du mest med webbplatsen?
- Varför gillar du det bäst?
- Är det lätt att förstå vad som säljs på webbplatsen?
- Kan du hitta ett födelsedagskort och lägga till det i kundvagnen?
- Kan du hitta något annat typ av kort och lägga till det i kundvagnen?
- Kan du ta bort ett objekt från kundvagnen?

- Kan du ta bort allt i kundvagnen och börja om?
- Vad skulle du vilja förbättra?
- Är det något som förvirrar dig?

Baserat på de svar jag får kommer jag att göra ändringar i webbplatsens design och utförande för att säkerställa att den blir så bra den bara kan.

### 3.5 Nuget-paket och utvecklingsmiljö

Jag kommer att använda följande Nuget-paket för att få till min lösning.

- **BuildWebCompiler.**
- **Microsoft.EntityFrameworkCore.SqlServer.**
- **Microsoft.VisualStudio.Web.CodeGeneration.Design.**
- **Microsoft.EntityFrameworkCore.Design.**
- **Microsoft.EntityFrameworkCore.Tools.**
- **SendGrid.**

Det sistnämnda paketet, SendGrid, är ett e-posthanteringssystem som möjliggör automatiserade e-mail i .NET och kommer därför att användas för e-postbekräftelse.

Jag kommer att programmera i utvecklingsmiljön Visual Studio 2019, och sedan när det är dags för publicering kommer jag att använda mig av Microsoft SQL Server Management Studio 18 för databashantering, och jag kommer att publicera på ett webbhotell (Simply, tidigare Unoeuro, då jag testat det förut och det fungerar bra) som jag införskaffar en domän hos. Domänen ska heta sayhi.se.

## 4 Konstruktion

### 4.1 Sitemap

För att få en överblick över de sidor som skulle skapas för webbshoppen så gjordes en sitemap. Denna kan du se i figur 2.



Figure 2: Sitemap över SayHi

Det denna visar är hur själva webbshoppen – inte administratörssidorna – skulle byggas upp, och gav en bra bild över vad som behövdes för att webbshoppen skulle bli komplett. Alla kort/kategori-sidan var tillexempel bara en sida, men med .NET kod som reglerade vilken kategori kunden ser utefter vad de klickat på. Detsamma gäller enskilt kort-sidan som visade på det kort som valts under alla kort/kategori-sidan, trots att det bara var en och samma sida. Webbshoppen har alltså därför fler sidor än vad som syns i sitemapen, men själva vyerna är inte så många. Sidorna är nämligen dynamiska, något som går att läsa mer om längre fram.

### 4.2 Wireframes

Några wireframes skapades sedan för de viktigaste sidorna när det kom till layout. Du kan se dessa under Bilaga A: Wireframes.



### 4.3 Moodboard

För att få en känsla för siten och vad som skulle skapas designmässigt så skapade jag en moodboard. Det är ett simpelt, men effektivt sätt att få en överblick över webbplatsens design, och du kan se den i figur 3.



Figure 3: Moodboard som skapats under förarbetet

Jag valde att lägga in några nyckelord på vad jag ville förmedla med siten, som blev "friendly", "stylish", "simple" och "fun". Jag ville helt enkelt skapa en webbplats som tilltalade så många som möjligt då gratulationskort inte har någon specifik målgrupp enligt mig. Tanken var att alla som sökte gratulationskort skulle tilltalas av denna webbplats, och det var baserat på denna tanke som denna moodboard skapades.

### 4.4 ER-diagram

För att kunna skapa databasen på bästa sätt, som skulle ligga till grund för alla objekt på webbplatsen som var till salu, så skapade jag ett ER-diagram som visar de tabeller jag behövde skapa för administrationsgränssnittet. Se figur 4.

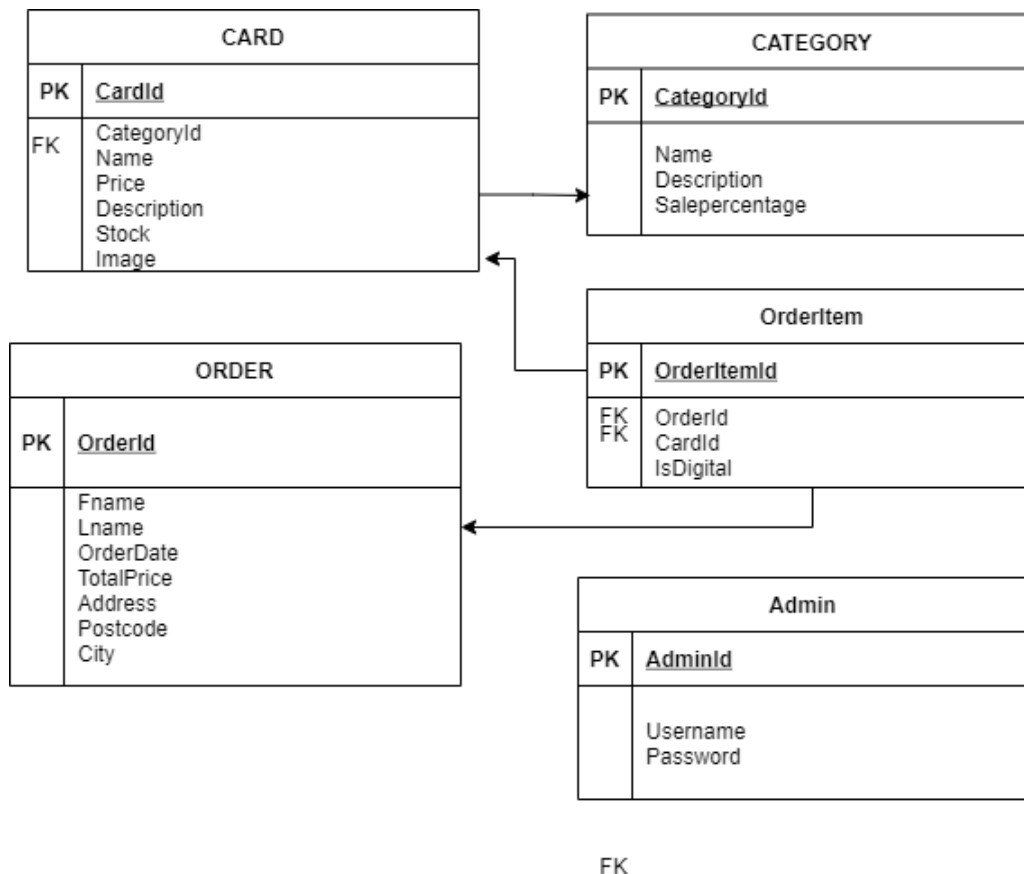


Figure 4: ER-diagram över databasen

Det detta ER-diagram visar är hur Card, OrderItem, Order och Category förhåller sig till varandra, nämligen att CategoryId är en foreign key i Card, så att man kan koppla ihop korten med den kategori de tillhör. Salepercentage i Category la jag till för att jag ville att det skulle gå att lägga till en rabatt på till exempel alla födelsedags-kort samtidigt, och jag tyckte att detta var det smidigaste sättet att göra detta på. I salepercentage skulle man som administratör då nämligen skriva ett värde mellan 0-1 och om man då skrev till exempel 0.1 så betydde det 10% rabatt. Admin-tabellen har ingen relation till de andra tabellerna, som man kan se i ER-diagrammet, men är en nödvändig tabell för inloggning och övrigt underhåll av resterande tabeller.

## 4.5 Skapandet av startsidan

Eftersom jag gjort en wireframe över startsidan samt en moodboard, så var det inte så svårt att fixa denna designen när det väl kom till att programmera den. Jag tog hjälp av min wireframe och skapade filen `_Layout.cshtml`. Jag skapade även `_LayoutAdmin.cshtml` som var den layout-fil jag skulle använda på administratörssidorna. Jag ville nämligen inte ha samma layout på både webbshoppen och administratörssidorna så jag skapade en för vardera. Jag fick sedan definiera vilken layout som skulle användas på de diverse sidorna, se figur 5.

```
@model SayHiStore.Models.Card  
  
@{  
    ViewData["Title"] = "Radera kort";  
    Layout = "~/Views/Shared/_LayoutAdmin.cshtml";  
}
```

Figure 5: Hur layout specificerades i projektet

I `_Layout.cshtml`, som då var för webbshoppen, la jag till logotypen i mitten av headern, och med en länk på vardera sida ("Alla kort" och "Kundvagn"). Övriga länkar valde jag att lägga i footern. Bredvid "kundvagn"-länken la jag in ett span-element som jag sedan via JavaScript skulle manipulera till att visa det antal objekt kunden har i sin kundvagn.

Jag fortsatte sedan med att skapa startsidan via den Index-fil som representerade denne. Jag la in text och bilder för att skapa ett helhetsintryck som jag var nöjd med och som såg ut som den wireframe jag skapat för startsidan. Se figur 6 för en skärmbild över hur det färdiga resultatet blev.

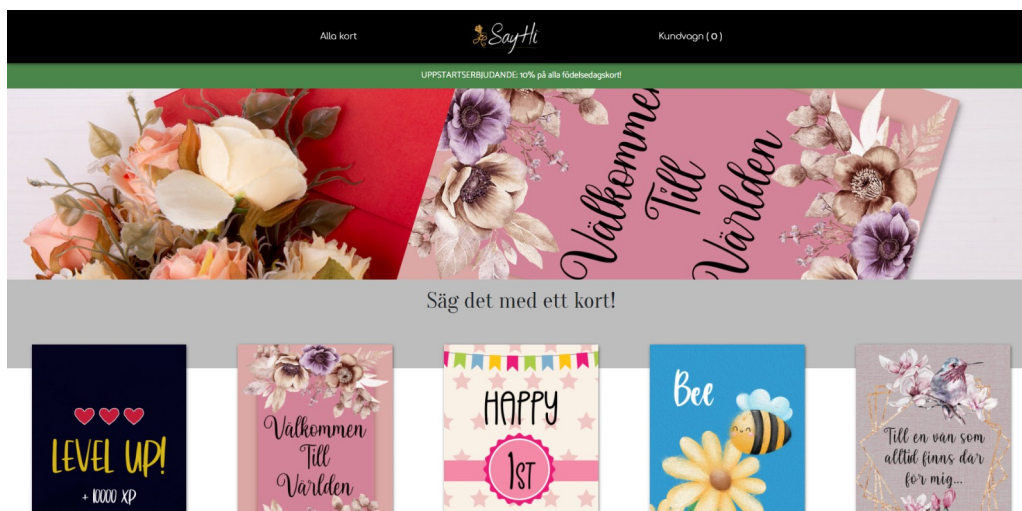


Figure 6: Webbplatsens startsida

Bilderna som visades på startsidan, samt korten under rubriken "Säg det med ett kort!" hade jag skapat i Photoshop med hjälp utav mockup-mallar, bildobjekt och olika typsnitt som jag ansåg bidrog till en snygg typografi. Det var meningen att det inte skulle gå att klicka på korten som låg på startsidan, då detta bara skulle vara ett litet urval av de kort som fanns tillgängliga för försäljning och inte faktiska länkar. För att visa korten var man istället tvungen att klicka på "Alla kort"-länken i huvudmenyn.

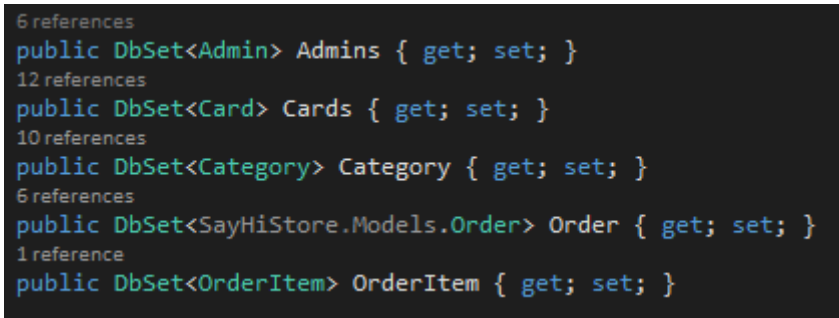
## 4.6 Administratörsgränssnittet

När jag skulle skapa administratörsgränssnittet så visste jag att jag skulle ha med en inloggning, så att bara verifierade administratörer skulle kunna komma

in på dessa sidor. Jag skulle också ha med diverse klasser som jag sedan via ett code-first tillvägagångssätt skulle skapa tabeller av i databasen med hjälp av Entity framework.

De klasser jag skapade var "Admin", som var till för att spara administratörsanvändare till databasen, "Card", "Category", "Order" och "OrderItem", de två sistnämnda var för att lite senare kunna spara alla ordrar som gjordes så att man visste vem som hade beställt vad.

Jag skapade en SayHiContext-fil som ärvde DbContext-klassen, och däri la jag till DbSet för varje tabell i databasen, se figur 7.



```
6 references
public DbSet<Admin> Admins { get; set; }
12 references
public DbSet<Card> Cards { get; set; }
10 references
public DbSet<Category> Category { get; set; }
6 references
public DbSet<SayHiStore.Models.Order> Order { get; set; }
1 reference
public DbSet<OrderItem> OrderItem { get; set; }
```

Figure 7: DbContext-klassens innehåll

Efter att detta var klart så scaffoldade jag in controllers och vyer till mitt projekt. Dessa vyer redigerade jag sedan till att passa min design och layout. Till exempel så ville jag för vyn som tillät att man skapade nya kort att man skulle se en dropdown-lista med de kategorier (då jag lagt CategoryId som en foreign key) som gick att välja, istället för att man skulle behöva veta vilket CategoryId som hörde till vilken kategori. Jag gjorde detta genom att skapa en metod i kontrollern för cards (CardsController) som skapade en lista av alla tillagda kategorier som jag sedan lät metoden returnera. För att använda denna metod så anropade jag den till en ViewBag från metoderna Create och Edit i CardsController så att jag skulle kunna komma åt den från vyn. I vyn tog jag sedan och loopade igenom den ViewBag jag skapat, och skrev ut ett alternativ per objekt i listan med namnet på kategorin som det som visades utåt, och med CategoryId som value så att enbart Id:t skulle sparas i databasen och inte hela namnet.

Jag la även till data annotations till de klasser jag skapat så att felmeddelanden skulle skrivas ut om användaren försökte lägga till något med felaktiga uppgifter, eller avsaknaden av uppgifter. Se figur 8.

```
[Display(Name = "Beskrivning")]
[StringLength(2000, MinimumLength = 45, ErrorMessage = "Beskrivningen måste vara mellan 45 och 2000 tecken lång")]
18 references
public string Description { get; set; }
[Display(Name = "Antal i lager")]
[Range(0, 999, ErrorMessage = "Måste vara ett värde mellan 0 och 999")]
17 references
public int Stock { get; set; }
[Display(Name = "Bild på kortet")]
17 references
public byte[] Image { get; set; }

[Required(ErrorMessage = "Du måste lägga till en bild")]
[Display(Name = "Bild på kortet")]
[NotMapped]
9 references
public IFormFile FormFile { get; set; }
```

Figure 8: DataAnnotations i klassen Card

Det som visas i denna figuren är en del av Card-klassen. Det går att se att jag längst ner i denna lagt ett "NotMapped" IFormFile-objekt, och detta var så att jag skulle kunna lägga till bilder i databasen för att visa de olika korten. Anledningen till att den är "NotMapped" är för att detta gör att den inte läggs till i databasen, och det var det jag ville. Det IFormFile-objektet gjorde var helt enkelt att jag använde detta för att omvandla bilden till något som gick att spara i databasen, nämligen en ByteArray. Genom en metod jag sedan skapade i kontrollern som jag kallade "GetImageFromByteArray" så omvandlade jag sedan tillbaka bilden till en Base64String, som gjorde att jag kunde visa bilderna från databasen på webbshoppen.

Med hjälp av TempData som jag sparade Base64String i kunde jag sedan möjliggöra för att användaren inte skulle behöva ladda upp en ny bild om de inte ville varje gång de redigerade ett kort. Genom att spara bilden, eller strängen som tillhörde bilden, i TempData, så kunde jag sedan konvertera det sparade värdet till en ByteArray och återigen spara den till databasen, utan att användaren behövde ladda upp en ny bild.

## 4.7 Inloggningsfunktionalitet

Jag ville att enbart registrerade användare skulle kunna komma in i administratörsgränssnittet, så jag skapade med hjälp av cookies en inloggningsfunktionalitet. Först skulle det gå för användaren att, om de hade den "administratörskod" som jag hårdkodade in i kontrollern, kunna skapa ett nytt administratörskonto. Denna administratörskod var bara ett värde jag hittade på för att inte vem som helst skulle kunna skapa ett konto, utan enbart personer som hade tillgång till koden.

För varje sida som hörde till administratörgränssnittet så kollade jag sedan om en cookie var satt till ett användarnamn, något som skedde vid inloggning eller skapande av konto. För att logga in så kollade jag de inmatade uppgifterna från login-formuläret jag skapade, gentemot de uppgifter som låg i databasen, och verifierade på så sätt att användaren var registrerad och använde rätt lösenord. Om inte uppgifterna matchade varandra vid försök till inlogg så skrev jag ut detta till användaren via backend. Se figur 9 för ett kodexempel på hur verifieringen av om användaren var inloggad eller ej fungerade.

```
2 references
public async Task<IActionResult> Index()
{
    var user = HttpContext.Request.Cookies["User"];
    ViewBag.Username = user;
    if (user != null)
    {
        return View(await _context.Admins.ToListAsync());
    }
    return RedirectToAction("Index", "Home");
}
```

Figure 9: Kontroll av användar-cookie

Som sagt kollade jag av i varje metod som hörde till en administratörsvy om en cookie var satt, och som man ser i figur 9 så sparade jag detta till en variabel som jag sedan kollade om den var skild från null eller inte. Om den inte var null, så returnerades vyn som användaren försökt gå in på, och om den var null så skulle användaren skickas tillbaka till webbshoppens startsida.

## 4.8 Design av administratörsgränssnittet

För att skapa ett lättanvänt gränssnitt så försökte jag hålla det så minimalistiskt som möjligt. Med SASS i en fil som jag kallade main.scss så designade jag layouten såsom jag ville ha den. Dova färger och lättlästa typsnitt som ändå var desamma som jag använde för webbshoppen fick ligga till grund för layouten. I main.scss la jag in allt som hade med administratörsgränssnittet att göra, och lämnade det som hade med webbshoppen att göra i en annan SASS-fil. Detta var för att skapa ett projekt som skulle vara lättare att få en överblick över och underhålla. Dock, eftersom jag arbetade med SASS, så kunde jag använda mig av partials, vilket jag gjorde. Jag skapade en partial för en css-reset och en partial för variabler såsom färger, och dessa partials kunde jag sedan importera i båda SASS-filerna (både den för administratörer och för webbshoppen) utan att behöva upprepa kod.

Jag validerade sedan administratörsgränssnittet för att se till så att allt var okej, och detta gjorde jag med hjälp av W3's "Markup Validation Service" [9] och "CSS Validation Service" [10]. Jag fick fel på CSS-valideringen, men det var i en bootstrap-fil som kom tillagd till projektet från början och inte något jag skapat själv, så det fick vara.

## 4.9 Gränssnitt för att visa ordrar

Det sista jag gjorde med administratörsgränssnittet fram tills det var dags för testning, var att jag skapade funktionalitet för att visa ordrar som gjorts. Detta så att de som arbetar med SayHi skulle veta vart orderarna skulle skickas, och vilka som hade lagts. Jag skapade som sagt klasserna Order och OrderItem, där OrderItem hade en foreign key som var OrderId. På så sätt kunde jag koppla ihop de objekt som fanns i ordern (orderItem) med den order de tillhörde.

Genom att via en metod som jag kallade för "OrderItemsList" kunde jag sedan på sidan "Details" för varje order (som jag scaffoldat fram), läsa in id:t på ordern och kontrollera vilka orderItems som hörde till den ordern, och sedan skriva ut i vyn vilka objekt det var. Se figur 10.

```
1 reference
public List<OrderItem> OrderItemsList(int id)
{
    var orderItems = _context.OrderItem.Where(m => m.OrderId == id).ToList();
    return orderItems;
}
```

Figure 10: Metod OrderItemsList()

Jag kallade sedan denna metod i metoden för "Details" genom att spara den till en ViewBag och sedan så kunde jag från vyn loopa igenom denna ViewBag och skriva ut vilka objekt som hörde till respektive order.

## 4.10 Webbshoppens uppbyggnad

För att inte manuellt behöva skapa en sida per objekt till salu, så skapade jag två dynamiska vyer, en för enskilda kort och en för visning av kort på kategori-nivå. Vid tryck på "Alla kort" i huvudmenyn på webbshoppen skulle man komma till den vy jag skapat för visning av kategorierna. Jag kontrollerade genom controllern om en specifik kategori hade valts, eller om användaren kollade på alla kort (vilket var default). Via denna sida kunde de sedan klicka sig in på den kategori de ville visa. Allt detta skedde i en och samma vy, och sättet jag gjorde detta på var att jag som sagt kontrollerade om en kategori valts, vilket jag gjorde genom att kolla efter ett id i adressraden, och sedan skriva ut de objekt som hörde till det CategoryId som jag hittat däri. Se figur 11 för en bild av hur denna sida såg ut.

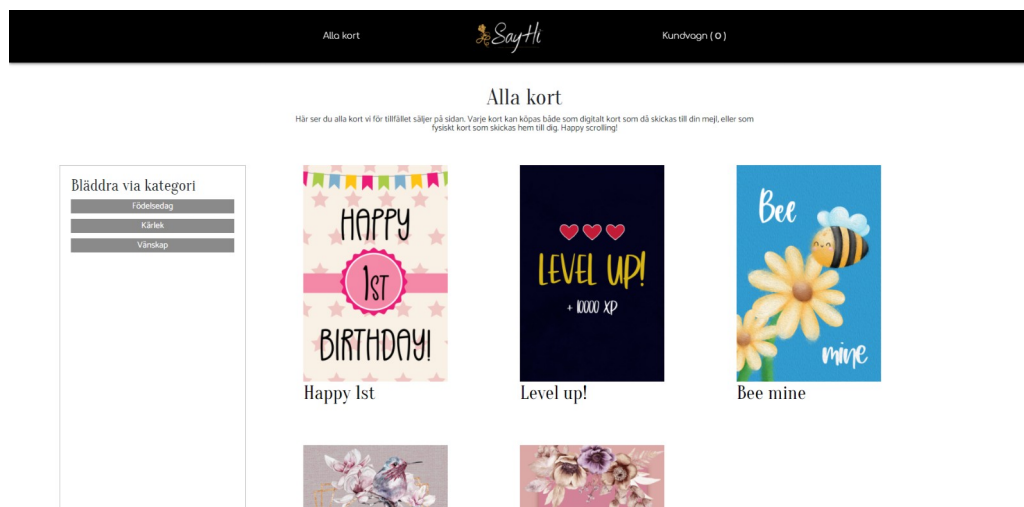


Figure 11: Uppvisande av alla kort på webbplatsen

Som man ser i figuren så valde jag att, på stora skärmar, lägga kategori-



visningen på sidan och sedan låta alla kort visas till höger. Under rubriken "Alla kort" (eller kategorinamnet, om en kategori hade valts), så skrev jag ut den beskrivning som var kopplad till antingen alla kort eller till respektive kategori.

Under "Bläddra via kategori" visade jag automatiskt vilka kategorier som blivit tillagda från administratörsgränssnittet, och dessa gick att trycka på så att man kom dit man ville. Tryckte användaren på "Födelsedag" så skulle bara födelsedagskort visas, till exempel.

Vid klick på ett av korten som visades i denna vyn, så skulle användaren komma till en sida för enskilda kort. Även denna var skapad dynamiskt så att i kod fanns det bara en vy, men denna vy uppdaterades baserat på vilket kort som valts. På sidan för enskilda kort skrev jag ut bilden som hörde till kortet, kortets namn, beskrivning och pris samt ett formulär där man skulle välja om det kort man köpte skulle vara digitalt eller fysiskt. Vid val av fysiskt kort skulle fem kronor läggas till på priset som man sedan kunde se i kundvagnen. Det fanns sedan en "Lägg till i kundvagnen"-knapp som vid tryck sparade objektet i den kundvagn jag sedan skapade med JavaScript. När man la till ett objekt i kundvagnen skulle även numret bredvid "Kundvagn" i headern uppdateras till det antal som man för tillfället hade i kundvagnen. Även detta gjorde jag med JavaScript.

## 4.11 Kundvagnen

Vid klick på "Lägg till i kundvagnen" så anropades en funktion jag skapat (AddToCart()). Denna funktion hämtade det objekt i localStorage som jag döpt till cart. Om cart var tom, så definierades den variabel som cart hämtats till som en tom array. Sedan använde jag mig av push() och la till objektet som jag skickat med när jag anropade AddToCart, nämligen det objekt som kunden lagt till i kundvagnen, i den variabel som jag sparat kundvagnen hittills i. Sedan uppdaterade jag localStorage cart till den nya arrayen. För att kunna spara datat och läsa ut det använde jag mig även av JSON.parse och JSON.stringify. Se figur 12.

```
function AddToCart(cardid, price, image, name) {  
    var cartArr = JSON.parse(localStorage.getItem("cart"));  
    if (cartArr == null) {  
        cartArr = [];  
    }  
  
    if (document.getElementById("digital").checked) {  
        cartArr.push({ id: cardid, type: "digital", name: name, price: price, image: image });  
    } else if (document.getElementById("physical").checked) {  
        cartArr.push({ id: cardid, type: "physical", name: name, price: price, image: image });  
    }  
    localStorage.setItem("cart", JSON.stringify(cartArr));  
}
```

Figure 12: En del av Javascript-kundvagn

I själva vyn för kundvagnen så visade jag sedan alla objekt som var tillagda i kundvagnen för användaren, också detta via JavaScript. Jag hade lagt tomma element i vyn som jag sedan skrev ut kundvagnen till. Jag hade även ett element



för det totala priset som jag räknade ut i min JavaScript-funktion som jag kallade showCart(). Då SayHi säljer både digitala och fysiska kort, så gjorde jag i denna funktion en kontroll om var kort var digitalt eller fysiskt. Var det digitalt så la jag till det i en array som hette nameListDigital, och annars i en som hette nameList. Sedan kunde jag under rätt rubrik skriva ut de objekt som hörde till vardera rubrik, se figur 13.

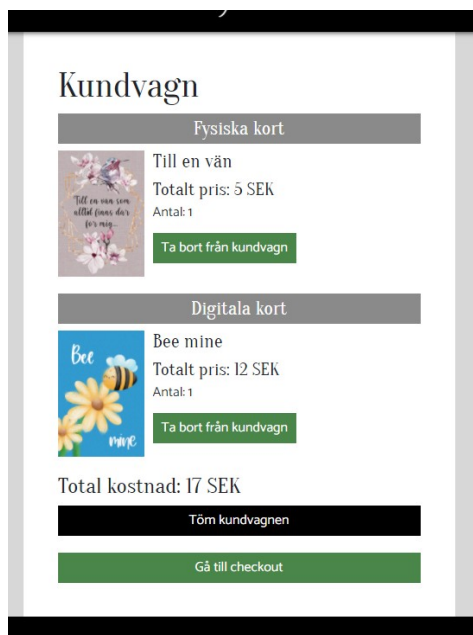


Figure 13: Kundvagnen i frontend

Detta visar en exempel-kundvagn och man kan se i bilden att det finns knappar för att ta bort ett objekt ur kundvagnen, som även detta sker via JavaScript, och för att tömma hela kundvagnen. Vid klick på att ta bort ett objekt ur kundvagnen så anropas funktionen deleteItem() som tar objektets namn som argument, och som sedan kollar vart i kundvagnen detta ligger, och sedan tar bort det elementet med splice(). Sedan sätts localStorage cart igen till de element som inte har blivit borttagna. Vid klick på "Töm kundvagnen" så görs en removeItem() på objektet cart i localStorage, och sedan laddas sidan om för att visa de nya ändringarna.

För att rätt nummer ska visas i huvudmenyn bredvid "Kundvagn" så hämtas cart återigen från localStorage, och sedan genom en for-loop så läggs 1 till en variabel per varje object i cart. Se figur 14.

```
function showNumber() {  
    var cart = JSON.parse(localStorage.getItem("cart"));  
    var cartNum = 0;  
    if (cart != null) {  
        for (var x = 0; x < cart.length; x++) {  
            cartNum = cartNum + 1;  
        }  
    }  
  
    document.getElementById("numCart").innerHTML = cartNum;  
}
```

Figure 14: Koden bakom elementet som visar antal objekt i kundvagnen

Detta nummer skrivs sedan in i span-elementet numCart som ligger i Kundvagns-länken.

## 4.12 Checkout

När en kund är färdig med sin kundvagn och är redo att gå till checkout, så kan denne klicka sig vidare från kundvagnen. Vid tryck på "Gå till checkout" så kommer användaren till en sida vid namn Checkout som kontrollerar via JavaScript om några element i kundvagnen är fysiska kort, och isåfall ombeds kunden att skriva in en leveransadress. Om det bara finns digitala kort i kundvagnen ombeds kunden istället bara att skriva in sitt förnamn, efternamn och email. Då det än så länge bara är möjligt att välja att betala med faktura så får kunden endast frågan vilken email fakturan ska skickas till. Dessa data, antingen bara email och namn eller email, namn och adress, sparas till Order-tabellen i databasen. Emailen sparas även som en cookie för att den ska kunna visas upp på nästkommande sida så att kunden kan försäkra sig om att denne har skrivit in rätt email utifall att bekräftelsemejlet som ska skickas inte kommer fram. Se figur 15 för ett flödesschema på hur flödet i kundvagn och checkout dels ser ut.

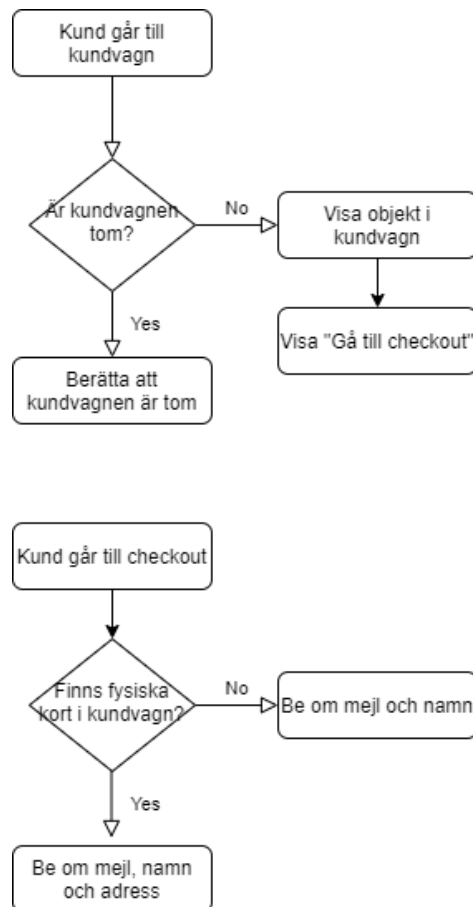


Figure 15: Flödesschema över checkout

När ordern sparas i databasen, så gör den det i backend i kontrollern i en metod som heter OrderCreate som anropas när användaren trycker på "Lägg beställning". Om det går bra att lägga till ordern, så skickas användaren vidare till en metod "OrderItemCreate" som är till för att alla OrderItems ska skapas och ha en foreign key till rätt Order. Via JavaScript sparas kundvagnen även till en cookie, som sedan läses i OrderItemCreate, decodas (då den encodats i JavaScript för att kunna sparas som cookie) och deserialiseras med hjälp av JsonConvert. Se figur 16.

```
var base64EncodedBytes = System.Convert.FromBase64String(HttpContext.Request.Cookies["theCart"]);  
var obj = System.Text.Encoding.UTF8.GetString(base64EncodedBytes);  
var cartItem = JsonConvert.DeserializeObject<List<CartItem>>(obj);
```

Figure 16: Inläsning och encoding av kundvagnen från cookie

Detta gör så att information om varje objekt, nämligen typen av objekt (fysiskt/digitalt) och kortets CardId, kan sparas i OrderItem-tabellen. Där sparas även OrderId:t som skapades i den förut nämnda metoden OrderCreate, så att OrderItem hör ihop med rätt Order.

## 4.13 Email-bekräftelse

För att användaren ska få en bekräftelse på att köpet är genomfört så görs detta via ett automatiskt utskickat email. För detta har jag använt mig av SendGrid [11] som är en email-tjänst som rekommenderades till mig av Microsofts dokumentation. Jag har där skapat ett konto och registrerat min webbplats domän och kopplat ihop SendGrid med min domän via min domäns DNS-inställningar. Sedan laddade jag ner Nuget-paketet SendGrid som är ett C# bibliotek för att kunna skicka email med SendGrid.

Med kod i kontrollern som anropas efter det att köpet genomförts så skickas sedan ett email till den email-adress som användaren angett som sparats som en cookie för inläsning i backend. Se figur 17.

```
var client = new SendGridClient(apiKey);
var from = new EmailAddress("hello@sayhi.se", "SayHi");
var subject = "Orderbekräftelse";
var to = new EmailAddress(emailTo, "Beställare");
var plainTextContent = "Tack för din order!";
var htmlContent = "<h1>Tack för din order! Vi är glada att du valt att handla hos oss!</h1>";
var msg = MailHelper.CreateSingleEmail(from, to, subject, plainTextContent, htmlContent);
var response = await client.SendEmailAsync(msg);
```

Figure 17: E-postbekräftelsekod

Grunden till koden har jag lånat från SendGrid [11] men lagt till mina egna variabler och texter. Emaillet som skickas är simpelt och innehåller ingen mer information än texten "Tack för din order! Vi är glada att du valt att handla hos oss!", men det skickas direkt och berättar för användaren att ordern har blivit mottagen.

I figur 17 kan man se att email-adressen det kommer stå att mejlet är från är "[hello@sayhi.se](mailto:hello@sayhi.se)" och detta är en email-adress som hör till den domän jag införskaffat för att kunna publicera webbplatsen på, nämligen sayhi.se.

## 4.14 Den responsiva designen

För att färdigställa webbplatsen innan testning krävdes några saker. En responsiv design och de statiska sidorna skulle skapas samt så skulle webbplatsen publiceras.

Med media queries anpassade jag webbplatsen så att den skulle likna de wireframes jag skapat. Övriga SASS-tillägg lades till för att webbplatsen skulle få den känsla jag önskade. Som sagt var min SASS-kod uppdelad i partials och två olika filer för huvud-designen, och detsamma gällde media queriesen som jag la till, nämligen att de hamnade i respektive SASS-fil, på slutet. Administratörsgränssnittet var simpelt att göra responsiv då den som sagt redan hade en simpel design för att inte distrahera administratörer från det de var där för att göra. Webbshoppens design gjordes också responsiv, och detta gällde även de två statiska sidor "Contact" och "Conditions" (köpvillkor) som jag skapade och länkade till från webbshoppens footer.

När det sedan var dags för publicering av webbplatsen så gjordes detta via en inställnings-fil som jag laddade ner från mitt webbhotell och sedan importerade i Visual Studio 2019. Jag fick sedan logga in med mina uppgifter för att verifiera anslutningen, och sedan publicerade jag webbplatsen till min domän <http://sayhi.se/>.

## 4.15 Funktionalitetstestning och användartestning

När webbplatsen var publicerad så fortsatte jag med ett arbete som pågått under hela tiden jag programmerade webbplatsen, nämligen att testa dess funktioner. Jag testade så att det gick att lägga till och redigera kategorier och kort, att det gick att visa och lägga till administratörsanvändare och att kundvagnen fungerade som den skulle inklusive den lilla räknaren bredvid kundvagns-länken. Jag testade så att det gick att checka ut, och att email-bekräftelsen skickades.

Efter det ville jag testa designen, och gjorde det i Google Chrome, Microsoft Edge och Mozilla Firefox. Jag kollade igenom webbplatsen i dessa webbläsare och det såg ut som de skulle, så jag gick vidare till att testa webbplatsen på faktiska användare.

Under två zoom-intervjuer så testade jag webbplatsen på två potentiella kunder. Jag ställde frågor för att ta reda på om de blev förvirrade av något, om de gillade något extra eller om det var något de inte gillade. Funktionerna på webbplatsen kunde de inte hitta fel på, men jag fick några synpunkter om designen på sidan. Till exempel så hade jag den gröna färgen som man kan se till vänster i figur 18 som bakgrund på varannan tabellrad på administratörsgränssnittet, men efter feedback så ändrade jag till den högra färgen i figur 18 då det inte var en lika påträngande färg.




Figure 18: Färger före och efter

Jag ändrade även några knappars design då de såg aningen tråkiga ut. Till exempel så ändrade jag i kundvagnen som du kan se i figur 19 där det högra är hur slutresultatet blev och den vänstra hur den var till en början.


## Kundvagn

Fysiska kort



Till en vän  
Totalt pris: 15 SEK  
Antal: 1  
[Ta bort från kundvagn](#)

Digitala kort



Bee mine  
Totalt pris: 12 SEK  
Antal: 1  
[Ta bort från kundvagn](#)

Totalt kostnad: 27 SEK

[Töm kundvagnen](#) [Gå till checkout](#)

## Kundvagn

Fysiska kort



Till en vän  
Totalt pris: 5 SEK  
Antal: 1  
[Ta bort från kundvagn](#)

Digitala kort



Bee mine  
Totalt pris: 12 SEK  
Antal: 1  
[Ta bort från kundvagn](#)

Totalt kostnad: 17 SEK

[Töm kundvagnen](#)

[Gå till checkout](#)

Figure 19: Kundvagn före och efter

När jag hade ändrat min layout och design utefter feedbacken från intervjuerna så publicerade jag den återigen via Visual Studio 2019 till min domän.

## 5 Resultat

Resultatet av arbetet blev en fungerande e-handel som möjliggör försäljningen av fysiska och digitala kort till användare i Sverige. Förarbete har gjorts som resulterat i wireframes, moodboard, ER-diagram och Sitemap. Detta förarbete har sedan legat som grund till utvecklingen av webbplatsen.

Bilder har redigerats och skapats för försäljning och visning på webbplatsen. För att kunna visa upp de objekt som är till salu har administratörsgränssnitt skapats med funktioner för ordrar, administratörer, kort och kategorier. Detta administratörsgränssnitt har testats för att säkerställa att det fungerar som det ska.

Gränssnitt för webbshoppen har skapats, både på kategori-nivå och enskilt kort-nivå. Detta gränssnitt har designats med responsiv design och enskilt-kort-vyerna innehåller en ”Lägg till i kundvagn” knapp som möjliggör för kunden att starta en kundvagn.

Kundvagns-sida har skapats och visar med hjälp av JavaScript upp de objekt som ligger i kundvagnen hos kund. En kundvagns-räknare finns bredvid Kundvagn-knappen. Checkout kan nås från kundvagnen och det går att lägga beställningar som registreras i order-registret som kan visas från administratörsgränssnittet. Vid lagd beställning skickas en e-postbekräftelse till av kund angiven e-post adress.

Testning av webbplatsen har gjorts. Användare har haft inflytande på designen på webbplatsen som har designats om därefter.

Webbplatsen är skapad med ASP.NET Core MVC som backend och JavaScript, SASS och HTML5 som frontend.

Webbplatsen fungerar som den ska och har validerats, och är publicerad till publikt tillgänglig webhost.

## 6 Slutsatser

En webbplats har skapats som möjliggör för start-up företaget SayHi att sälja egenskapade gratulationskort. Webbplatsen fungerar som det är tänkt och SayHi är nöjda med resultatet.

### 6.1 Analys och resultatdiskussion

Planen har alltid varit att skapa en fullt fungerande webbshop åt SayHi, och slutresultatet blev något snarare fullt fungerande som det rimligen gick att komma. SayHi kommer nämligen i framtiden om affärerna går bra att utveckla ett betalsystem som innefattar annat än faktura, men för tillfället fungerar det bra med fakturabetalning.

Webbplatsen har en fullt fungerande administratörsdel, som interagerar med webbshoppen på så sätt att de objekt som laddas upp för försäljning sedan visas på huvudwebbplatsen, och att ordrar som kommer från huvudwebbplatsen visas på administratörsdelen. Detta fungerar bra och möjliggör för faktisk försäljning från SayHi.

Jag har inte skapat så många gratulationskort som jag lagt upp för försäljning, men det är något SayHi kommer att arbeta vidare med allt eftersom webbplatsen växer.

Det arbete som har gjorts har nått målen, och jag känner mig nöjd när jag lämnar över webbplatsen till SayHi.

### 6.2 Projektmetod diskussion

Det var bra att jag valde att göra förarbetet som jag gjorde, då det underlättade själva utvecklingen av webbplatsen, speciellt ER-diagrammet och wireframes. Att jag valde att arbeta i Visual Studio 2019 anser jag var bra då det är rustat för att ta hand om ASP.NET applikationer (bland annat), och för att publicering därifrån är så pass enkel. När jag behövde uppdatera min kod efter testningen så var det bara att trycka på "Publish" och webbplatsen återuppladdades vilket var väldigt smidigt. Att jag valde att skicka e-postbekräftelse till kund vid genomförd order känns seriöst och som att det ger ett genuint intryck vilket är viktigt för nya företag såsom SayHi så att kunder väljer att handla där, och så att de kommer tillbaka.

### 6.3 Etisk och social diskussion

Vissa personer var inkluderade i skapandet av detta projekt, främst de användare som jag gjorde användartester på. Jag har valt att inte göra deras namn publika i denna rapport för att de ska få vara anonyma.



Detta projekt är starten på att hjälpa folk vara mer sociala och fira varandra under dessa tider när social distansiering är så pass viktigt och en så stor del av vår vardag på grund utav Covid-19.

Jag sparar vissa cookies med denna applikation, dock inget alldeles privat. Jag sparar till exempel orderns totala pris, men även användarens e-postadress. Detta är dock nödvändigt för att webbplatsen ska fungera, och funktionaliteten bygger mycket på dessa cookies. Vid vidareutveckling och med kontakt med personer som kan det bättre än jag så borde en fullständig GDPR-beskrivning finnas med på webbplatsen, men då jag inte besitter kompetensen för detta så har jag utlämnat det från min applikation.

Jag har inga bilder av människor på min webbplats, och jag använder ingen spårning eller liknande. Webbplatsen gör det enkelt för folk att nå ut till sina nära och kära och ingens integritet behöver kompromissas för att uppnå detta.

Allt som allt så är detta en webbplats som jag är nöjd med och som ligger rätt i tiden, och som har potential att få SayHi att nå sina mål och drömmar om att vara med i kampen om kunder som vill köpa gratulationskort på internet.

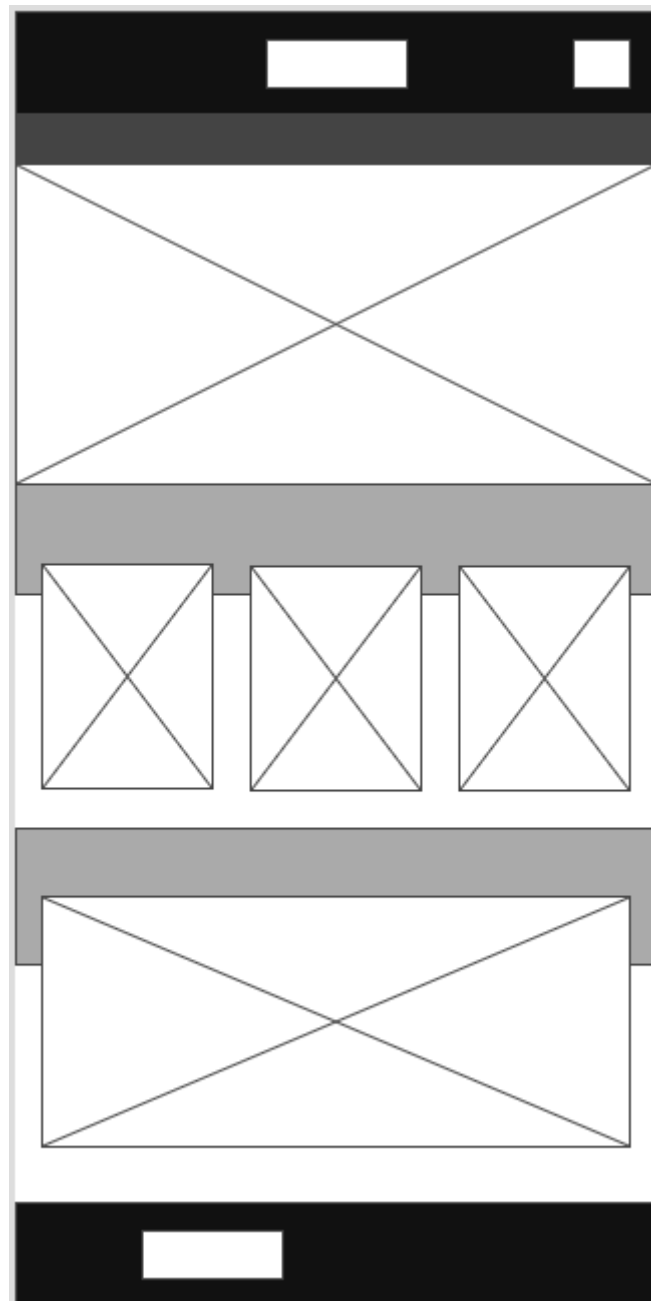
## Källförteckning

- [1] Microsoft, "Scaffold Identity in ASP.NET Core projects",  
<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/scaffold-identity?view=aspnetcore-5.0&tabs=netcore-cli#scaffold-identity-into-an-mvc-project-without-existing-authorization>  
Hämtad 2021-04-20.
- [2] Microsoft, "Overview of ASP.NET Core MVC",  
<https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0>  
Hämtad 2021-04-20.
- [3] Sass, "Documentation",  
<https://sass-lang.com/documentation>  
Hämtad 2021-04-20.
- [4] Microsoft, "Entity Framework Core",  
<https://docs.microsoft.com/en-us/ef/core/>  
Hämtad 2021-04-20.
- [5] Microsoft, "Language Integrated Query (LINQ)",  
<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/>  
Hämtad 2021-04-20.
- [6] Microsoft, "Database Providers",  
<https://docs.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli>  
Hämtad 2021-04-20.
- [7] Wireframe.cc, "Wireframe.cc",  
<https://wireframe.cc/>  
Hämtad 2021-04-20.
- [8] Microsoft, "ASP.NET Scaffolding in Visual Studio 2013",  
<https://docs.microsoft.com/en-us/aspnet/visual-studio/overview/2013/aspnet-scaffolding-overview>  
Hämtad 2021-05-11.
- [9] W3, "Markup Validation Service",  
<https://validator.w3.org/>  
Hämtad 2021-05-23.

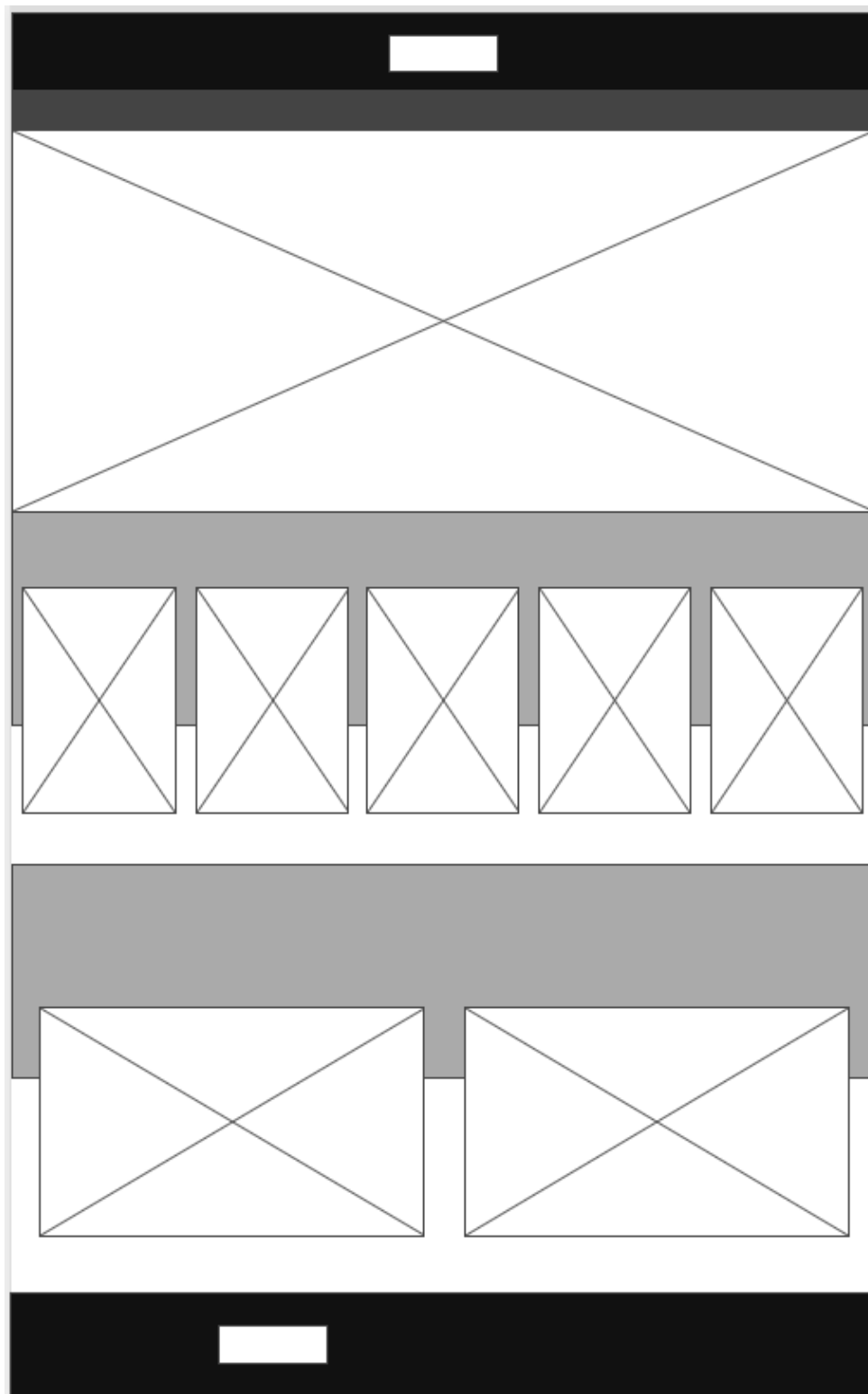
- [10] W3, "CSS Validation Service"  
<https://jigsaw.w3.org/css-validator/>  
Hämtad 2021-05-23.
  
- [11] SendGrid, "SendGrid"  
<https://sendgrid.com/>  
Hämtad 2021-05-23.

## Bilaga A: Wireframes

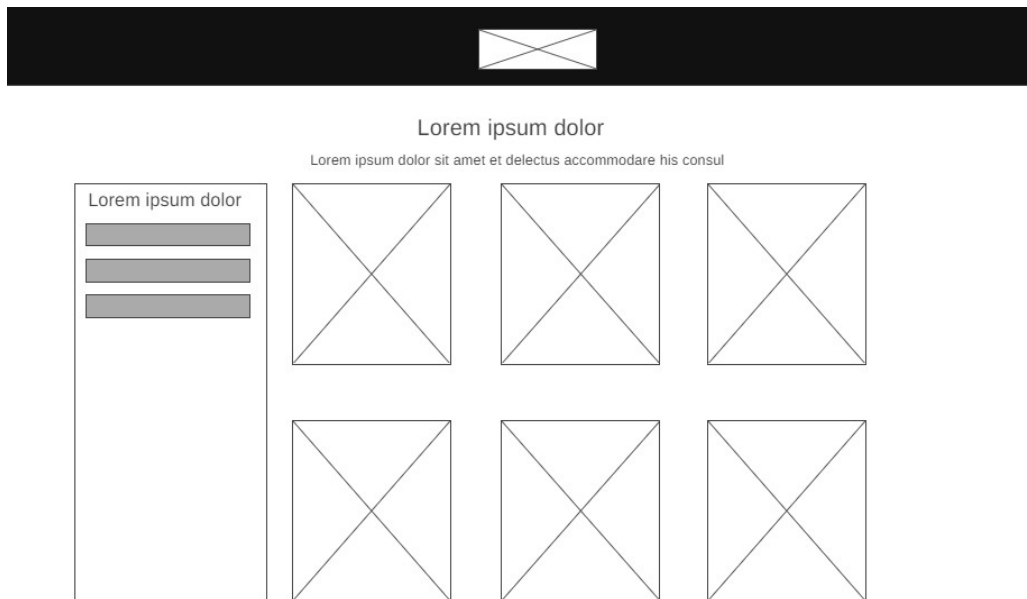
### Mobil-startsida



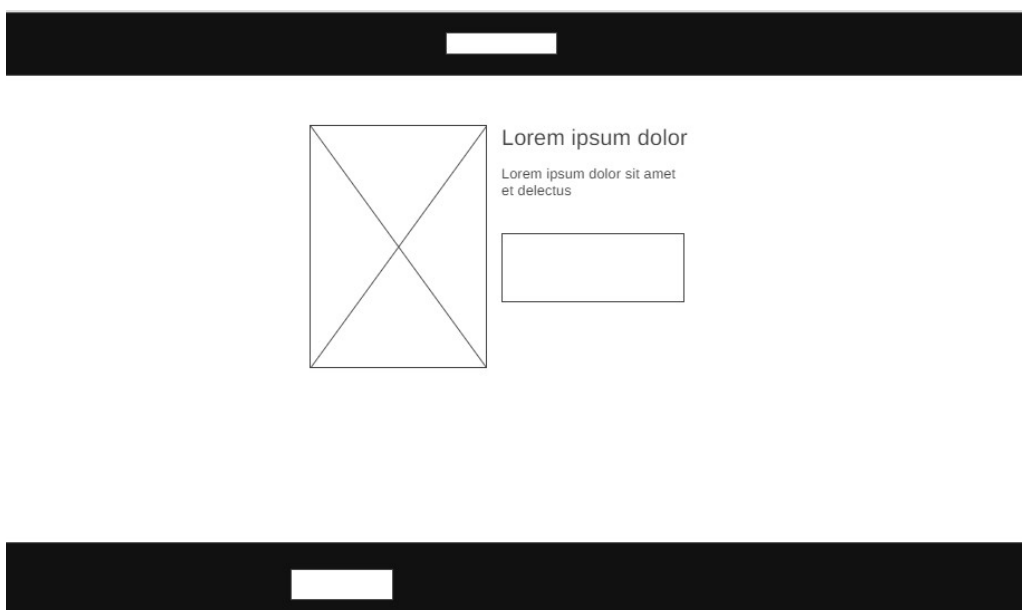
## Desktop-startsida



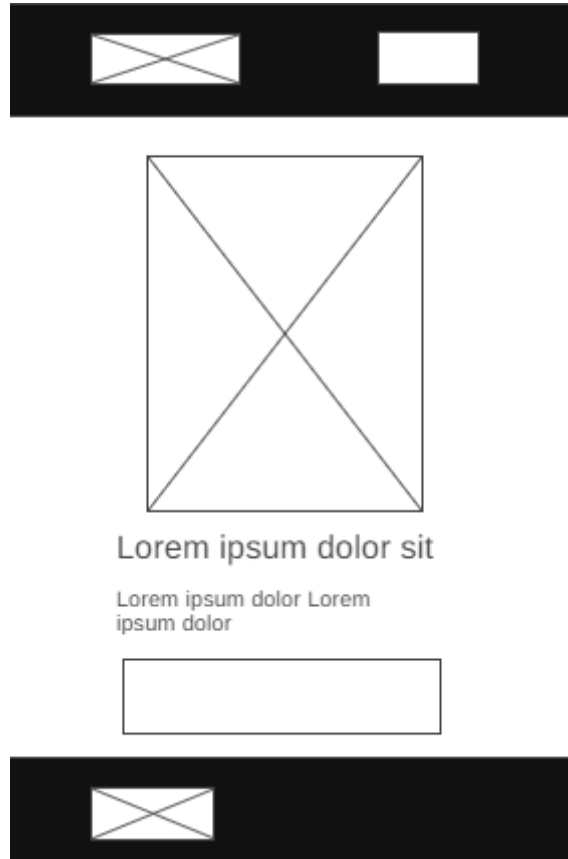
## Desktop-alla kort



## Desktop-enskilt kort



## Mobil-enskilt kort



## Mobil-alla kort

