**Projektuppgift**

*DT208G – Programmering i TypeScript*

**Moment 5 - Projekt**

Webbplats för det fiktiva universitetet Nordic Institute of Advanced Learning

**Emma Lorensson**

**MITTUNIVERSITETET  
Avdelningen för informationssystem och -teknologi**

**Författare:** Emma Lorensson, [emlo2302@student.miun.se](mailto:emlo2302@student.miun.se)  
**Utbildningsprogram:** Webbutveckling, 120 hp  
**Huvudområde:** Datateknik  
**Termin, år:** VT, 2024

# Sammanfattning

Projektets syfte var att med TypeScript som programmeringsspråk skapa en webbplats åt det fiktiva universitet Nordic Institute of Advanced Learning (NIAL). Webbplatsen skulle innehålla en startsida med allmän information om universitetet, en sida som listar samtliga av universitets kurser och en sida som listar det ramschema studenten själv väljer utifrån universitets kurssida.

Innehållsförteckning

[Sammanfattning ii](#_Toc175316560)

[Terminologi v](#_Toc175316561)

[**Definition av termer och förkortningar** v](#_Toc175316562)

[1 Introduktion 1](#_Toc175316563)

[1.1 Bakgrund och problemmotivering 1](#_Toc175316564)

[1.2 Övergripande syfte 1](#_Toc175316565)

[1.3 Konkreta och verifierbara mål 2](#_Toc175316566)

[2 Teori 3](#_Toc175316567)

[2.1 TypeScript 3](#_Toc175316568)

[2.2 Angular 3](#_Toc175316569)

[2.2.1 Komponenter 3](#_Toc175316570)

[2.2.2 Routing 4](#_Toc175316571)

[2.2.3 Services 4](#_Toc175316572)

[2.2.4 Data Binding 4](#_Toc175316573)

[2.2.5 Dependency Injection 4](#_Toc175316574)

[2.3 Angular Material 4](#_Toc175316575)

[3 Metod 5](#_Toc175316576)

[3.1 Utvecklingsmiljö och verktyg 5](#_Toc175316577)

[3.2 Validering och tillgänglighet 5](#_Toc175316578)

[3.2.1 Validering av kod 5](#_Toc175316579)

[3.2.2 Validering enligt WCAG 2.0 5](#_Toc175316580)

[3.2.3 Prestanda 6](#_Toc175316581)

[4 Konstruktion 7](#_Toc175316582)

[4.1 Planering av webbapplikation 7](#_Toc175316583)

[4.2 Skapande av moodboard och logotyp 7](#_Toc175316584)

[4.3 Skapande av designskisser 8](#_Toc175316585)

[4.4 Utveckling av webbapplikation 8](#_Toc175316586)

[4.4.1 Struktur 8](#_Toc175316587)

[4.4.2 Responsivitet 9](#_Toc175316588)

[4.4.3 Globala element 9](#_Toc175316589)

[4.4.4 Startsida 10](#_Toc175316590)

[4.4.5 Om oss 11](#_Toc175316591)

[4.4.6 Menyer 12](#_Toc175316592)

[4.4.7 Administrationsgränssnitt 14](#_Toc175316593)

[5 Resultat 17](#_Toc175316594)

[6 Slutsatser 18](#_Toc175316595)

# Terminologi

Nedan presenteras de akronymer som används i rapporten.

|  |  |
| --- | --- |
| **Definition av termer och förkortningar** | |
| NPM | Node Package Manager |
|  |  |
|  |  |

# Introduktion

Den här rapporten kommer att steg för steg gå igenom och motivera de tillvägagångssätt, kod och designval som gjorts för att skapa en webbplats åt det fiktiva universitetet Nordic Institute of Advanced Learning, nedan kallat NIAL. I detta första kapitel behandlas *bakgrund och problemmotivering, syfte, avgränsningar* samt *konkreta och verifierbara mål.*

## Bakgrund och problemmotivering

NIAL är ett universitet beläget i en medelstor stad i Sverige som erbjuder sina befintliga och blivande studenter att bygga sitt eget ramschema. För att studenterna enkelt ska kunna göra detta vill NIAL ha en webbsida där samtliga av deras kurser listas i en sök- och filtreringsbar lista. Utifrån listan ska studenterna sedan snabbt kunna söka fram relevanta kurser och lägga till dem i sitt ramschema, som visas på en egen undersida.

Samtliga av universitets kurser och information om dessa finns lagrat i ett API som universitetet administrerar. Detta API kommer även ligga till grund för de kurser som listas på webbplatsen. För att universitetet enkelt ska kunna kontrollera att samtliga kurser är tillgängliga i listan på webbplatsen önskar de att det i anslutning till listan står skrivet hur många kurser som finns tillgängliga i den. De önskar också att studenten, i sitt ramschema, ska kunna se det sammanlagda antalet högskolepoäng som de tillagda kurserna har.

Projektet ska byggas i TypeScript med Angular som ramverk då skolan har erfarenhet av att arbeta i detta sedan tidigare. För att snabba på utvecklingstiden och därigenom minimera kostnader önskar de att, i så hög grad som möjligt, ett komponentbibliotek används för webbplatsens olika komponenter.

## Övergripande syfte

Projektets övergripande syfte är att skapa en webbplats byggd i ramverket Angular åt det fiktiva universitetet NIAL. Studenter ska kunna söka och filtrera bland universitets samtliga kurser och lägga till kurser till ett eget ramschema. Studenterna ska också kunna plocka bort kurser från ramschemat och se det totala antalet högskolepoäng som är tillagda i ramschemat.

## Konkreta och verifierbara mål

Projektets mål är att nedan punkter ska vara uppfyllda vid färdigställande:

* Webbplatsen ska skapas med Angular och TypeScript och komponenter och routing ska användas.
* Webbplatsen ska bestå av en startsida och två undersidor, en för att söka och visa information om kurser - och en som visar skapat ramschema.
* Minst två stycken services skall skapas, en för kursdata och en för hantering av ramschema.
* Skapat ramschema skall lagras med hjälp av localStorage, och läsas in vid inladdning av webbsidan.
* Kurserna ska gå strukturera genom att filtrera på ämne, sorteras i samtliga kategorier och fritextsöka.
* Det ska gå att se antal kurser när såväl samtliga kurser visas som i respektive sökträff.
* Det sammanlagda antalet högskolepoäng för samtliga kurser i ramschemat ska visas.
* Webbplatsen skall vara snygg och prydlig och fungera väl på stora som små skärmar med bra responsiv design.

# Teori

## TypeScript

TypeScript är ett programmeringsspråk som är baserat på JavaScript, men lägger till statisk typning med möjlighet typannoteringar. När TypeScript körs transpileras det till JavaScript vilket innebär att det går att köra överallt där JavaScript går att köra [1]. Det är främst utvecklat för skapandet av stora applikationer, då den statiska typningen kan hjälpa till att minska antalet buggar och fel i koden, men kan även användas i mindre projekt [2].

Några av fördelarna med TypeScript är:

* **Statisk typning**Med statisk typning kan man definiera datatyper för variabler, funktioner och objekt. Det kan ge tidigare upptäckt av fel och förbättrar koden genom att göra den mer lättläst [3].
* **Klasser och arv**Objektorienterad programmering stöds genom klasser och arv, vilket ger ett mer strukturerat sätt att bygga och hantera kodbaser [3].
* **Gränssnitt (interfaces)**Det är möjligt att skapa gränssnitt som hjälper till att definiera kontrakt för olika delar av koden [3].
* **Moduler**Ett inbyggt modulstöd underlättar en tydligare organisering av koden och möjliggör effektiv återanvändning av kod [3].
* **Generics**Generics möjliggör skapande av flexibla och återanvändbara komponenenter genom att tilldela specifika typer när komponenterna används, snarare än när de skapas [3].

## Angular

Angular är ett ramverk för att bygga webbapplikationer som använder TypeScript som programmeringsspråk. Det utvecklades och underhålls av Google tillsammans med en community av utvecklare och företag [4].

### Komponenter

Angular använder en komponentbaserad arkitektur som gör det möjligt att utveckla modulära och återanvändbara delar i en applikation. Varje komponent inkapslar sin egen HTML-, CSS- och TypeScript-kod, vilket underlättar både underhåll och testning av de enskilda delarna av systemet [4].

### Routing

Angular kommer med en inkluderad router som möjliggör enklare hantering av navigationen i en single-page applikation [4].

### Services

I angular används services för att separera logiken från komponenterna, vilket gör det möjligt att återanvända funktionalitet. De används vanligtvis för att hantera serverkommunikation eller för att dela data och funktioner mellan olika delar av applikationen [5].

### Data Binding

Angular stödjer såväl en- som tvåvägs data binding, vilket innebär att ändringar i användargränssnittet automatiskt uppdaterar koden och ändringar i koden automatiskt uppdaterar användargränssnittet. Envägs data binding innebär att information bara skickas åt ena hållet, t. ex från koden till gränssnittet. Medan tvåvägs data binding innebär att information skickas åt båda hållen samtidigt [5].

### Dependency Injection

Angular har ett inbyggt system för dependency injections, vilket gör det enklare att hantera beroenden mellan komponenter och services. Detta skapar modularitet och förenklar testning av koden [4].

## Angular Material

Angular Material är ett komponentbibliotek för Angular som är framtaget av Google. Material har en mängd olika tillgängliga komponenter, så som knappar, formulärfält, tabeller etcetera. Dessa komponenter är anpassningsbara och är lätta att integrera i angularapplikationer [4].

# Metod

I detta kapitel beskrivs vilka metoder som kommer att användas för att uppfylla de mål som angetts i kap. 1 Introduktion.

## Utvecklingsmiljö och verktyg

De system som används för att utveckla universitetets webbapplikation är följande:

* **Figma**  
  I Figma skapades de designskisser som låg till grund för webbapplikationens layout.
* **Visual Studio Code**Den kodeditor som används är Visual Studio Code – i den utvecklas all HTML, CSS och TypeScript-kod.
* **Angular**Angular används som ramverk.
* **Angular Material**Flertalet av de komponenter som finns i webbapplikationen är hämtade från Angular Material.
* **Canva**Canva används för att ta fram universitets logotyp.
* **GitHub**Git och GitHub används för att versionshantera sidan.
* **Netlify**Webbapplikationen är publicerad hos Netlify.

## Validering och tillgänglighet

### Validering av kod

För att säkerställa att webbplatsens kod uppfyller krav på kvalitet och funktionalitet kommer samtliga HTML- och CSS-filer att valideras med hjälp av W3C Web Validator i webbläsaren.

### Validering enligt WCAG 2.0

Webbplatsen kommer att granskas i enlighet med WCAG 2.0-standarderna med hjälp av AChecker Web Accessibility Checker, för att säkerställa en tillgänglig och inkluderande användarupplevelse.

### Prestanda

För att säkerställa att webbplatsen upprätthåller hög prestanda kommer tester att genomföras med hjälp av Lighthouse.

# Konstruktion

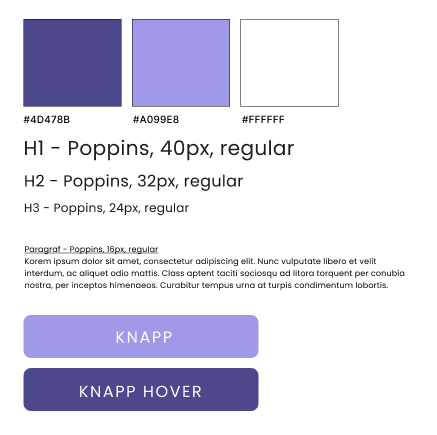
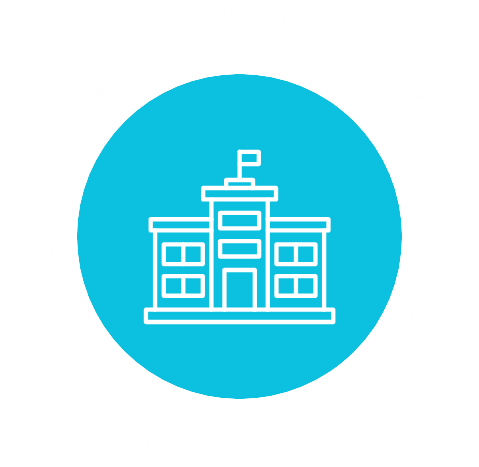
## Planering av webbapplikation

Projektet inleddes med att planera för de sidor och komponenter som behövde finnas på webbplatsen för att projektets mål skulle uppfyllas.

Det beslutades att webbplatsen skulle bestå av en startsida och två undersidor, varav den ena listar universitets samtliga kurser och den andra listar studentens valda ramschema.

## Skapande av moodboard och logotyp

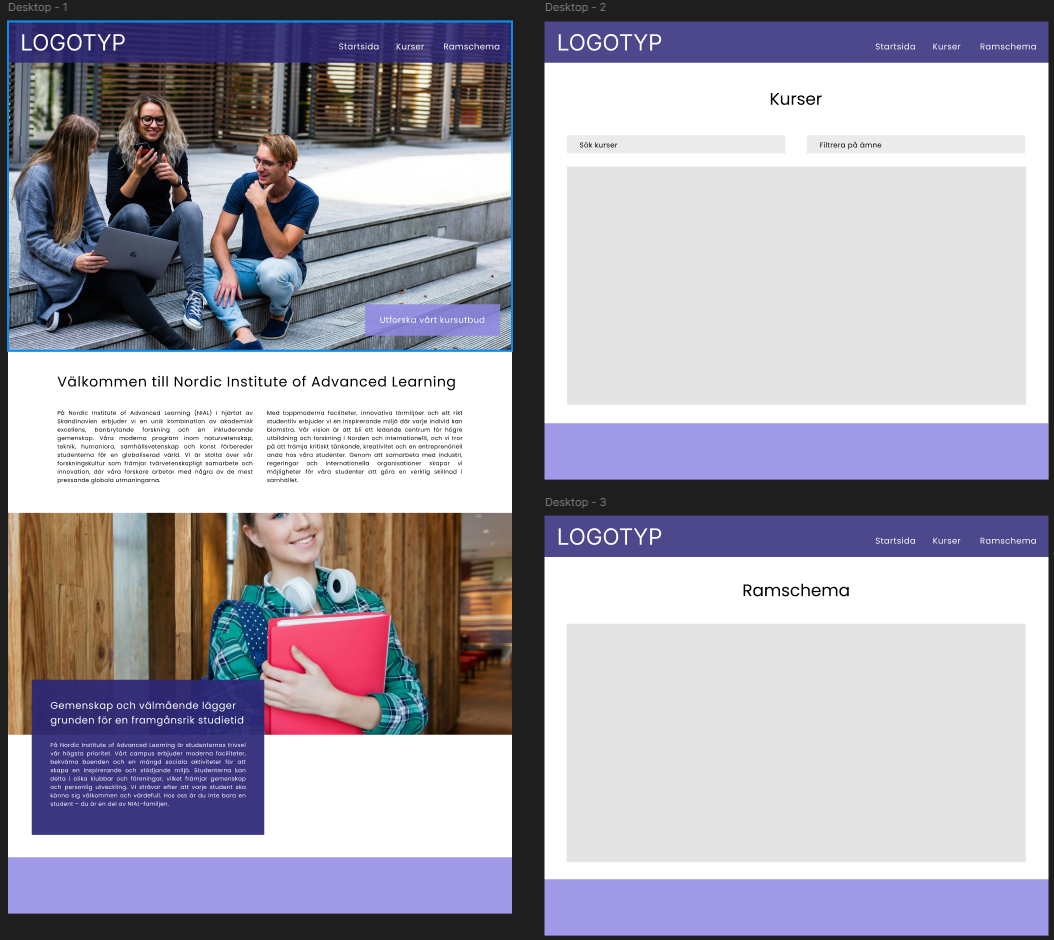
När strukturen på webbplatsen beslutats fortskred arbetet med att ta fram en grafisk profil till universitetet. Färgvalen gjordes utifrån en önskan om att framställa universitet dels seriöst, dels lekfullt och modernt. Den grafiska profilen inkorporerades sedan i en Moodboard där även typsnitt, textstorlekar och knapplayout beslutades (se figur 1).

I samband med att moodboarden skapades togs även universitets logotyp fram i verktyget Canva (se figur 2).

*Figur 1. Moodboard för universitetet NIAL Figur 2. Logotyp*

## Skapande av designskisser

I nästa steg skapades enkla designskisser över samtliga av webbplatsens sidor och komponenter i Figma (se figur 3). I designskisserna implementerades de designval som tagits i samband med att moodboarden togs fram.

*Figur 3. Designskisser föreställande universitetet NIALs webbplats.*

## Utveckling av webbapplikation

För att utveckla webbapplikationen användes Visual Studio Code för att skriva HTML, SCSS och TypeScript-kod. Utvecklingsarbetet inleddes med att starta upp ett nytt angularprojekt genom kommandot *”ng new project”.* I och med att Angular själv installerar alla nödvändiga paket och sätter upp strukturen för projektet behövdes inget ytterligare göras i detta skede.

### Komponenter

Inledningsvis skapades tre komponenter, en för respektive startsida, genom kommandot *”ng generate component ’component-name’”.* Därefter skapades komponenter allt eftersom sidorna byggdes. I slutändan blev det totalt nio komponenter.

I src-katalogen skapades ytterligare tre kataloger, en som hanterar media-filer, en som hanterar SCSS-filer och en som hanterar JavaScriptfiler. Samtliga HTML-filer placerades direkt i huvudkatalogen src.

Totalt skapades 12 HTML-filer i projektet. Sju av sidorna går att nå från webbplatsens frontend och fem av sidorna tillhör administrationsgränssnittet och kräver att användaren är inloggad för att dessa ska kunna nås.

För frontend-delen skapades en startsida, en om oss-sida, en bokningssida, en login-sida och tre menysidor, en för respektive meny. För administrationsgränssnittet skapades en dashboard för att nå samtliga undersidor, samt tre olika sidor för att uppdatera de olika menyerna och en sida för att hantera bordsbokningar.

Respektive HTML-sida har en egen JavaScript-fil knuten till sig med all JavaScript-kod som behövs för den specifika sidan. Därtill skapades en main.js fil som innehåller den JavaScript-kod som är generisk för samtliga sidor på webbplatsen.

I samtliga HTML-filer infogades därtill en main.scss-fil som genereras från tre andra SCSS-filer, varav den ena innehåller samtliga variabler, den andra innehåller samtliga animationer och den tredje all övriga SCSS-kod.

### Responsivitet

Webbplatsen utvecklades med responsiviteten i åtanke genom användandet av flexbox och dynamiska måttenheter i de fall där detta passat. Dock behövde webbplatsen trots detta anpassas med en relativt många media querys för att få de olika sidornas element att se bra ut på samtliga skärmstorlekar.

### Globala element

#### Header

Headern i HTML-filerna består av en logotyp som är placerad i det övre vänstra hörnet genom att tilldela den egenskaperna top: 0, left: 0. Då logotypen också har en animering tilldelad sig är left satt till -700px på elementet och efter att animationen körts tilldelas left: 0 som standard. Den animation som används på logotypen är ”slideInLeft” (se bilaga 1).

Därtill ligger även huvudnavigeringen i headern. Huvudnavigeringen är placerad i webbplatsens högra hörn och har tilldelats en fixed positionering för att den ska följa med vid scroll på sidan. Även huvudnavigering har tilldelats en animering, och elementet har inledningsvis egenskapen right: -900px men efter att animeringen körts sätts right till 0. Animationen som används på huvudnavigeringen är ”slideInRight” (se bilaga 1).

#### Footer

Footern innehåller tre containrar där den ena innehåller restaurangens logotyp, den andra innehåller länkar till webbplatsens olika sidor och den tredje innehåller kontaktuppgifter till restaurangen. Textfärgen sattes till vit och bakgrunden till svart. De olika containrarna positionerades genom att sätta footer-elementet till display: flex och justify-content: space-evenly. Footern har inga animeringar kopplade till sig.

### Startsida

#### Video

Det första användaren möts av på startsidan är en video som startar automatiskt och spelas på repeat genom attributen loop autoplay. Videon är placerad i vänsterkant med egenskapen left: 0. Videon är ett montage av flera olika videoklipp hämtade från Pexels.com. Montaget klipptes ihop i Adobe Premiere Pro. Animeringen ”slideInLeft” har lagts till på videoelementet (se bilaga 1).

#### Introducerande text

Ovanpå videon ligger en rubriktext i storlek 1 som är positionerad med en relativ positionering och en top-margin satt till -120 pixlar för att den översta delen av texten ska ligga ovanpå videon. Animeringen ”fadeInSkew” har använts på rubriktexten (se bilaga 1).

#### Citat

Efter rubriktexten kommer en paragraf med ett citat från restaurangens ägare. Denna är positionerad med en top-margin på 200 pixlar från föregående element. Citationstecknet ligger inom en span-container och är deklarerad med en font-storlek på 1400% för att sticka ut från texten.

#### Menyintroduktion

Ytterligare 150 pixlar från föregående element kommer en sektion som beskriver restaurangens menyer. Sektionen är deklarerad som en flexbox och innehåller i sin tur två containrar, där den första består av en rubrik i storlek 1, tre paragrafer och en knapp och den andra består av en bild. Samtliga element i den första containern har tilldelats animationen ”fadeInUp” och bildelementet har tilldelats animationen ”increaseWidth” (se bilaga 1).

#### Instagramflöde

Nästa sektion skapades för att lista restaurangens senaste inlägg på instagram. Det skapades ingen autentisk koppling till Instagram, utan istället har olika element satts samman för att efterlikna ett instagramflöde så som de generellt visas på webbplatser. Sektionen är också en flexbox som består av en rubrik i storlek 1, en container innehållandes ytterligare två containrar som vardera innehåller två ”kort” som ska motsvara instagraminläggen och en knapp.

Sektionen som innehåller samtliga av ovan nämna containrar är satt till flex med column som flex-direction. Rubriken är positionerad till höger med float: right. De två containrarna som innehåller ”instagraminläggen” och dess föräldracontainer är satta till flex. På större skärmar har båda dessa row som flex-direction, men allt eftersom skärmstorleken minskas så får först föräldra-containern column som direction och slutligen ändrar även kort-containrar direction.

Knappen är positionerad till höger genom egenskapen align-self: flex-end.

Samtliga element i instagram-containern är animerade med animationen ”fadeInUp” (se bilaga 1).

#### Öppettider

Efter instagramsektionen skapades en sektion som listar restaurangens öppettider. Sektionen består av en h1-rubrik positionerad till vänster med egenskapen align-self: flex-start. Därefter följer en container med tre ytterligare containrar som lista öppettider för olika dagar på veckan. Dessa containrar tilldelades en border på 1px solid. Föräldracontainern är satt till flex och de inre containrarna positioneras därefter med justify-content: space-between.

#### Boka-animering

Den sista sektionen som skapades för startsidan innehåller en animerad text som säger ”Boka”. I HTML-filen skapades en container som innehåller en SVG, som i sin tur innehåller ett länkelement med ett textelement inuti, där x är satt till ”0” och y ”80%”. Teckenstorleken på denna text är satt till 300 pixlar, vilket sedan minskas med hjälp av media querys för mindre skärmstorlekar. Animationen som används på texten är döpt till ”textAnimate” och använder egenskaperna stroke-dasharray och stroke-dashoffset för att få till den rörliga linjen run om. Vid animeringens start är stroke-dasharray satt till 0 50%, stroke-dashoffset är satt till 20% och fill är satt till den ljusare röda färgen i färgschemat. När animeringen avslutas är stroke-dasharray satt till 50% 0, stroke-dashoffset är satt till -20% och fill är satt till samma som bakgrundsfärgen på sidan (se bilaga 1). Animeringen körs sedan med attributen infinite alternate för att den ska köras hela tiden och alterera framlänges och bakländes.

Vid hover på texten ändras muspekaren till pointer och animeringen pausas, samtidigt som linjen runtom animeringen tilldelas den mörkare röda färgen i färgschemat.

### Om oss

”Om oss”-sidan liknar till stor del startsidan som gås igenom i kap 4.5.5 Startsida. Det första som skapades var en h1-rubrik med samma egenskaper som rubriken på startsidan. Därefter följer tre sektioner som samtliga är satta till display: flex och som positionerar versin sektion med en h2-rubrik och en paragraf på ena sidan och en bild på andra sidan. Dessa positioneras i föräldracontainern med justify-content: space-between.

Efter dessa tre sektioner följer sektionen för öppettider som nämns i kap. 4.5.5.6 Öpettider och sektionen för den animerade bokningsknappen som beskrivs i kap. 4.5.5.7 Boka-animering.

### Menyer

Vid klick på ”Meny”-länken i huvudnavigeringen landar användaren på sidan för middagsmenyn. Det första som dyker upp på sidan är en kompletterande navigering specifik för meny-sidorna. Där kan användaren klicka sig mellan middagsmeny, lunchmeny och takeaway-meny, som består av tre separata HTML-filer med samma kompletterande navigering.

Den kompletterande navigeringen består av en h1-rubrik som sedan fått en border-bottom på 1 px solid. Under denna ligger en oordnad lista innehållandes länkar till de olika menysidorna. Dessa är formaterade genom en förlängning (@extend) av egenskaperna från huvudnavigeringen, men för att de ska hållas på plats vid samtliga webbläsarstorlekar har de tilldelats en absolut positionering. Därtill har de row som flex-direction istället för column. Vid hover över någon av länkarna ändras textfärgen från vit till svart och bakgrundsfärgen från mörkgrå till den ljusare av de roströda färgerna i färgschemat. Denna bakgrundsfärg visar även vilken av länkarna som är aktiv, genom att en klass lagts på den aktiva länken och tilldelats samma egenskaper som vid hover.

Gundlayouten för menylistningen på de olika menysidorna är relativt simpel och består av dels en h1-rubrik som beskriver vilken av menysidorna användaren är inne på. Därefter följer två till fyra containrar som innehåller antingen information om menyn eller menyalternativen.

#### Middagsmeny

Middagsmenyn består av fyra containrar, en för respektive kategori på menyn. Den enda statiska texten som ligger i dessa containrar är rubrikerna smått, förrätter, huvudrätter och efterrätter. I övrigt läses all text in från webbtjänsten som beskrivs i kap. 4.4 Utveckling av webbtjänst enligt texten som följer.

Samtliga av de fyra containrarna hämtas in i JavaScript-filen för sidan och lagras i variabler.

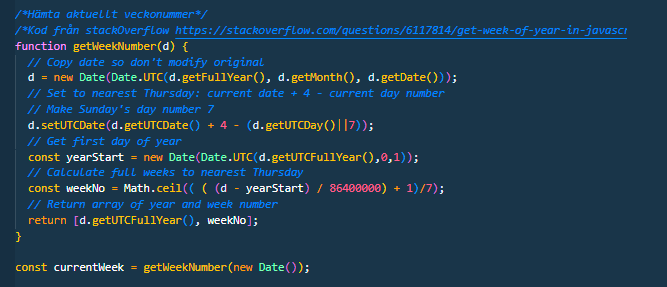
En funktion med try/catch skapades för att läsa in datan från api/dinners (se figur 4) genom att använda fetch. I funktionen kontrolleras det om svaret från API:et är okej och om det inte är det ”kastas” ett fel till catch-delen av funktionen som skriver ut felet i konsolen. Om svaret är okej omvandlas det till JSON-format och lagras i en variabel. Därefter anropas en funktion som loopar igenom middagsrätterna med datan från den lagrade variabeln.

Funktionen som loopar igenom datan är skapad för att ta emot ett argument. I funktionen deklareras först en tom variabel, därefter körs en forEach-loop som loopar igenom varje rätt i middagsmenyn. En switch-sats används för att kontrollera vilken kategori rätten tillhör för att kunna skriva ut den till rätt container i DOM. Det första caset i switch-satsen kollar om dinner.category är lika med ”Förrätt.”. Om så är fallet tilldelas den variabel som lagrat DOM-containern för förrätter till variabeln som deklarerades först i funktionen innan switch-satsen avbryts. Därefter hoppar funktionen vidare till en if-sats som kontrollerar variabeln som lagrar kategorin och skriver ut relevant information från API:t till rätt container via innerHTML.

För att få koden att köra anropas den första funktionen som hämtar datan och kör funktionen för att loopa igenom den.

#### Lunchmeny

Lunchmenyn består utav två containrar, en med statisk text som informerar om vad som gäller vid lunch och en som listar veckans luncher. JavaScript-koden som används för att skriva ut lunchmenyn till DOM liknar till viss del den som används för att skriva ut middagsmenyn. En skillnad är dock att personalen i administrationsgränssnittet kan lägga in flera lunchmenyer, en för respektive vecka på året, men endast den lunchmeny som är för innevarande vecka ska visas på webbplatsen. En funktion har därför lagts till för att hämta aktuellt veckonummer för att kunna kontrollera vilken veckas meny som ska skrivas ut till DOM. Koden för denna funktion är hämtad från Stack Overflow (se figur 5). Aktuell vecka lagras därefter i en variabel.

Figur 5. Bild av kodstycke som är hämtat från Stack Overflow.

Variabeln som lagrar den aktuella veckan används sedan forEach-loopen i funktionen som itererar datan genom att jämföra om den angivna lunchveckan i webbtjänsten är samma som aktuell kalendervecka, om de är samma, så loopas datan igenom och skrivs ut till DOM.

En annan skillnad i koden, jämfört med koden som skriver ut middagsmenyn, är att switch-satsen kontrollerar veckodagarna som är angivna för respektive lunch, istället för att kontrollera kategori.

#### Takeaway-meny

Koden som skriver ut Takeaway-menyn till DOM är i stort sett identisk med den som skriver ut middagsmenyn till DOM.   
Enda skillnaden är att det endast finns två Takeaway kategorier, till skillnad mot middagsmenyns fyra kategorier.

### Administrationsgränssnitt

Samtliga delar av administrationsgränssnittet är skyddade med jsonwebtoken och kan alltså endast nås om användare genererar en giltig token via inloggningsformuläret, eller om en giltig token redan finns lagrad i localStorage.

Samtliga sidor i administrationsgränssnittet har ett script lagt direkt i HTML-filen dör att verifiera om en giltig token finns eller inte. Anledningen till att scriptet ligger i HTML-filerna är för att koden ska köras direkt när sidan laddas. Under utvecklingen testades en variant där koden lades i JavaScript-filerna, men vid försök att nå någon av de skyddade routes laddades sidan i några millisekunder innan användaren omdirigeras. Genom att lägga koden direkt i HTML-filerna kunde detta problem undgås.

#### Dashboard administrationsgränssnitt

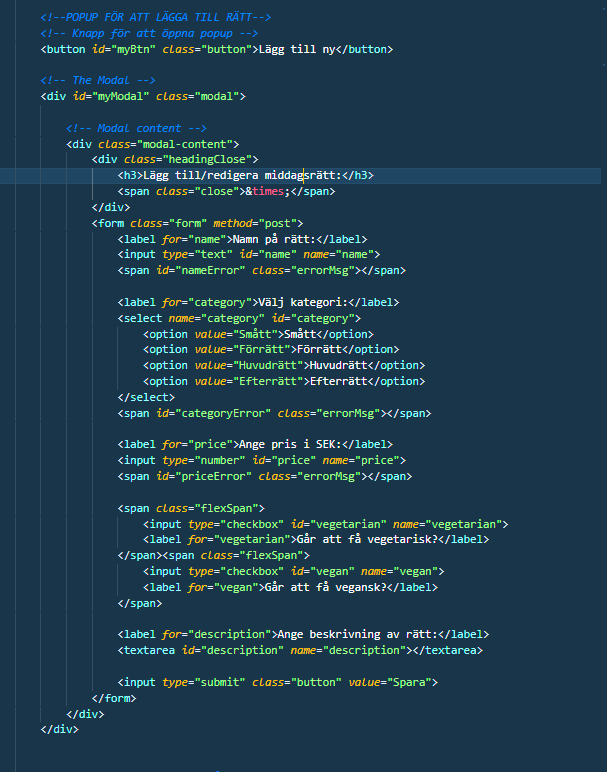
Dashboarden som är det första användaren kommer till efter inloggning i administrationsgränssnittet består av en HTML-fil som i princip endast innehåller fyra containrar innehållandes varsin h3-rubrik. En JavaScript-fil som hanterar omdirigering till skyddade routes vid klick på respektive container har importerats. I den filen hämtas de fyra containrarna och lagras i varsin variabel. Eventlyssnare har sedan lagts till för respektive container som vid klick omdirigerar användaren till korrekt sida. Från denna fil hanteras även koden för logga ut-knappen, som kort och gott är en eventlyssnare som hämtar elementet för logga ut-knappen och vid klick tar bort token från localStorage för att sedan omdirigera användaren till startsidan.

#### Uppdatera menyer

Det finns ett antal element som är gemensamma för samtliga av undersidorna i administrationsgränssnittet. De fyra containrarna som återfinns på dashboarden, finns också på samtliga undersidor, men det har fått en mindre storlek med SCSS och placeras högst upp i mitten av sidan. Detta för att användaren lätt ska kunna navigera mellan de olika undersidorna utan att behöva klicka sig tillbaka till dashboarden.

Därtill har samtliga sidor som uppdaterar menyer en modal innehållandes det formulär som används för att uppdatera eller lägga till en ny rätt. Modalen öppnas genom klick på antingen knappen för att lägga till, eller knappen för att uppdatera menyalternativ. Modalen är skapad med en mängd olika containrar och ett form-element innehållandes relevanta formulärfält för respektive databasschema (se figur 5).

Samtliga HTML-sidor har också containrar med en tunn border runt om som är positionerade centrerat med flexbox oxh justify-content: center. Respektive container innehåller en h3-rubrik som talar om vilken menykategori som återfinns i containern. På samma sätt som menyalternativen skrivs ut till DOM på webbplatsens menysidor skrivs de även ut i administrationsgränssnittet. Skillnaden är dock att för respektive menyalternativ tillkommer också två knappar, en för att uppdatera och en för att radera menyalternativet.



Figur 5. Modal i HTML-filen för att uppdatera middagsmeny

# Resultat

Projektets övergripande mål var att skapa en ny webbplats år den fiktiva restaurangen Benvenuto Amico Trattoria. Webbplatsen skulle representera restaurangen på ett tydligt och lättanvänt sätt och skapas i enlighet med en tydlig grafisk profil. Funktionalitet för att boka bord skulle finnas på webbplatsen och den skulle också innehålla ett administrationsgränssnitt där restaurangens personal kan logga in och själva uppdatera och redigera valda delar av innehållet.

Detta mål bröts sedan ner i mer konkreta delmål som skulle vara uppfyllda vid projektslut.

Det första konkreta delmålet var att designskisser ska presenteras i enlighet med Benvenuto Amico Trattorias grafiska profil. Detta mål skulle jag säga uppfylldes delvis, med tanke på att designskisser för endast ett fåtal sidor togs fram.

Nästkommande mål var att det ska vara enkelt att boka bord och hitta restaurangens menyer via webbplatsen. Detta mål tycker jag blivit uppfyllet. Ett av de första element användaren ser på webbplatsen är boka-knappen som ligger i huvudnavigeringen och har en annan färg än övriga knappar. Därtill har en stor boka-länk lagts till i slutet av flertalet sidor, denna leder också till sidans bordsbokning.

Det tredje målet var att ett lättanvänt administrationsgränssnitt skulle skapas där restaurangens medarbetare ska kunna uppdatera de tre olika menyerna, samt hantera inkommande bordsbokningar. Detta mål har uppfyllts och personligen anser jag att det är lättanvänt. Det är tydligt för användare hur de ska gå tillväga för att komma till de olika menyernas uppdateringssidor och knapparna för att ändra och redigera är också tydliga.

Det sista målet var att webbplatsen ska validera enligt standarden för w3c och WCAG 2.0 för att säkerställa en god tillgänglighet och även detta mål har blivit uppfyllt.

# Slutsatser

Det är många aspekter av det här projektet som jag hade gjort annorlunda om jag hade fått börja om. Dels hade jag lagt ner mer tid på det planerande arbetet när det kommer både till webbtjänsten och webbplatsen. Databasen för webbtjänsten hade troligtvis kunnat läggas upp mer effektivt med lite mer planering. Webbplatsen i sin tur hade troligtvis gått betydligt snabbare att utveckla om jag hade haft tydliga designskisser för samtliga sidor. Jag önskar därtill att jag hade planerat mina JavaScript-funktioner bättre från start. När jag nu i efterhand går igenom koden tycker jag att jag har väldigt mycket kod som är avsedd för att göra samma sak. Om jag planerat webbtjänsten bättre i kombination med en tanke kring hur jag skulle lägga upp mina JavaScript-funktioner hade jag troligtvis kunnat återanvända funktioner på ett annat sätt än vad jag gör nu. Så som det här projektet har utvecklats har jag tagit ett steg åt gången, utan att planera för hela applikationen och dess funktioner.

Jag har också känt att jag har komplicerat det här projektet onödigt mycket, med alla olika animationer och liknande, vilket har gjort att jag känt en otrolig tidspress och därmed inte kunnat hålla den kvalitet jag önskat på projektet. Med tanke på att det är ett stort projekt i flera delar redan från start, borde jag ha hållit mig till en mer enkel layout och därefter lagt på extra funktionalitet om det hunnits med.

Jag är relativt nöjd med sidans prestanda och är nöjd över att den ser bra ut på samtliga skärmstorlekar och i de olika webbläsare jag testat. Jag är också mycket nöjd med sidans layout som jag personligen tycker känns lyxig och lite annorlunda mot en vanlig restaurang.

Allt som allt finns det många lärdomar som jag tar med mig från det här projektet.

Källförteckning

1. [TypeScript: JavaScript With Syntax For Types. (typescriptlang.org)](https://www.typescriptlang.org/)
2. [TypeScript - Wikipedia](https://en.wikipedia.org/wiki/TypeScript)
3. [Moment 1 - Teori (matdah.github.io)](https://matdah.github.io/DT208G---Programmering-i-TypeScript/Moment%201/moment_1_teori.html)
4. [Angular (web framework) - Wikipedia](https://en.wikipedia.org/wiki/Angular_(web_framework))
5. [Moment 3 - Angular I - Teori (matdah.github.io)](https://matdah.github.io/DT208G---Programmering-i-TypeScript/Moment%203/moment_3_teori.html)
6. ‌ dotenv [Internet]. npm. 2024 [cited 2024 May 24]. Available from: https://www.npmjs.com/package/dotenv#%EF%B8%8F-usage
7. ‌cors [Internet]. npm. [cited 2024 May 25]. Available from: <https://www.npmjs.com/package/cors>
8. body-parser [Internet]. npm. 2019. [cited 2024 May 25]. Available from: <https://www.npmjs.com/package/body-parser>
9. jwt. JWT.IO - JSON Web Tokens Introduction [Internet]. jwt.io. [cited 2024 May 25]. Available from: <https://jwt.io/introduction>
10. nodemon [Internet]. nodemon.io. [cited 2024 May 25]. Available from: https://nodemon.io/