Examensarbete, Högskolan på Åland, Utbildningsprogrammet för informationsteknik

EN APPLIKATIONSPROTOTYP I ANGULAR

- Karriärlångt lärande

Erica Sundblom

2021:02

Datum för godkännande: 02.03.2021
Handledare: Björn-Erik Zetterman

EXAMENSARBETE
Högskolan på Åland

Utbildningsprogram:	Informationsteknik
Författare:	Erica Sundblom
Arbetets namn:	En applikationsprototyp i Angular - Karriärlångt lärande
Handledare:	Björn-Erik Zetterman
Uppdragsgivare:	-

Abstrakt

Området informationsteknik är i ständig utveckling. Tidvis är den utvecklingen väldigt snabb. De som tänker sig en framtid inom informationsteknik kommer således att behöva fortsätta lära under hela sina karriärer.

I det här projektet har jag jobbat med att lära mig en ny utvecklingsmiljö, en för mig ny teknik, nämligen ramverket Angular och skapat en applikationsprototyp. Angular är ett ramverk för *frontend*. Så jag har fokuserat på applikationens *frontend*, men kopplat ihop den med ett separat, men parallellt utvecklat *backend*. Som en extra del har jag tittat på vad progressiva webbapplikationer (PWA) är och gjort en enkel implementation av det i min applikationsprototyp.

Nyckelord (sökord)

Angular, Typescript, npm, HTML, CSS, progressiva webbapplikationer

Högskolans serienummer:	ISSN:	Språk:	Sidantal:
2021:02	1458-1531	Svenska	43 sidor

Inlämningsdatum:	Presentationsdatum:	Datum för godkännande:
02.03.2021	02.03.2021	02.03.2021

DEGREE THESIS
Åland University of Applied Sciences

Study program:	Bachelor of Information Technology
Author:	Erica Sundblom
Title:	An Application Prototype in Angular - Career-long Learning
Academic Supervisor:	Björn-Erik Zetterman
Technical Supervisor:	-

Abstract

The field of information technology is in constant development. At times this development is very fast. Those who see a future for themselves in information technology will therefore need to continue learning throughout their entire careers.

In this project I have worked with learning a new development environment, a for me new technology, the framework Angular and created an application prototype. Angular is a framework for frontend. So I have focused on the frontend of the application, but I have connected the frontend to a separately, but parallelly developed backend. As an extra part I have looked at what progressive web applications (PWA) are and made a simple implementation of that in my application prototype.

Keywords

Angular, Typescript, npm, HTML, CSS, progressive web applications

Serial number:	ISSN:	Language:	Number of pages:
2021:02	1458-1531	Swedish	43 pages

Handed in:	Date of presentation:	Approved on:
02.03.2021	02.03.2021	02.03.2021

INNEHÅLLSFÖRTECKNING

1. INLEDNING **6**
1.1 Hypotes 6
1.2 Syfte 6
2. METOD **7**
2.1 Angulars handledningar 7
2.2 Angulars dokumentation 7
2.3 Generella sökningar på internet 8
3. ANGULAR **9**
3.1 Angulars gränssnitt för kommandotolken 9
3.2 Applikationens beståndsdelar 9
3.2.1 Några grundläggande filer 10
3.2.2 Komponenter 11
3.2.3 Gränssnitt 13
3.2.4 Tjänster 13
3.2.5 Injicera beroenden 13
3.3 Dynamiskt innehåll i applikationen 14
3.3.1 Templates 14
3.3.2 Direktiv 14
3.3.3 Strukturdirektiv 14
3.3.4 Attributdirektiv 15
3.4 Hur några av delarna i Angular hänger ihop 18
3.5 Ytterligare hjälpmedel i Angular 18
3.5.1 Observerbara 18
3.5.2 HTTP API 19
3.6 Beroenden 21
3.6.1 node_modules och npm 21
3.6.3 Web Storage API 22
4. SPRÅK OCH TEKNIK **23**
4.1 Objektorienterad programmering 23
4.2 Språk 24
4.2.1 Typescript 24
5.2.1.1 För- och nackdelar 24
4.2.2 JavaScript 25
4.2.3 HTML 25
4.2.4 CSS 26
4.2.5 JSON 27

4.3 Andra hjälpmedel 27
4.3.1 Visual Studio Code 27
4.3.2 GitHub 28

5. PROGRESSIVA WEBBAPPLIKATIONER 29
5.1 Allmänt 29
5.2 För- och nackdelar 30
5.2.1 Progressiv 30
5.3 PWA med hjälp av Angular 31
5.4 Tjänstearbetare (service workers) 32
5.5 Klara svag nätverkskontakt och helt offline 32
5.6 Push-notifikationer 33
5.7 Reflektion 33

6. RESULTAT 34
6.1 Grunduppdraget 34
6.2 Tilläggsuppdraget 36

7. SLUTSATSER 39

KÄLL- OCH LITTERATURFÖRTECKNING 40

1. INLEDNING

Området informationsteknik är i ständig utveckling. Tidvis är den utvecklingen väldigt snabb. De som tänker sig en framtid inom informationsteknik kommer således att behöva fortsätta lära under hela sina karriärer.

1.1 Hypotes

När studenten kommit så långt i sina studier att hen får påbörja sitt examensarbete innehar hen tillräckliga kunskaper för att på egen hand lära sig och tillämpa nya tekniker inom programmering.

1.2 Syfte

Under utbildningstiden har vi knappt jobbat med ramverk. Så det är ett uppenbart område att utforska mera. Angular det kanske största ramverket för *frontend* på Åland. Därför är det relevant att lära sig något om Angular inför arbetslivet. Också internationellt är Angular ett stort ramverk (*State of Frontend 2020 Report*, 2020).

Mitt syfte har varit att lära mig en ny utvecklingsmiljö, en för mig ny teknik, nämligen ramverket Angular och skapa en applikationsprototyp. Angular är ett ramverk för *frontend*. Så tanken har varit att fokusera på applikationens *frontend* (delen användaren ser), men att koppla ihop den med ett separat, men parallellt utvecklat *backend* (del med huvudsaklig logik för bakomliggande resurser). Som en extra del har jag haft att eventuellt göra applikationsprototypen till en progressiv webbapplikation (PWA). Den delen har utgjort en möjlighet att lära om en andra ny teknik.

2. METOD

Hur ska någon göra för att lära sig Angular? Den som vill lära sig något behöver information. En sökning på "Angular" på internet ger Angulars hemsidor som första träff. Den som läser på Angulars hemsidor upptäcker ganska snart att det finns både handledningar och annan slags dokumentation där.

Praktiskt kodande ger mig alltid mer känsla och förståelse för teori jag läser eller hör. Och inte minst kommer jag bättre ihåg vad jag tagit till mig när jag jobbat med det och på så vis bollat det i mitt huvud flera gånger. Jag har således följt en handledning tidigt i mitt lärande. I mitt arbete med att lära mig Angular och utveckla en applikation har jag använt Angulars handledningar och dokumentation, samt letat information på hela internet.

2.1 Angulars handledningar

Angular tillhandahåller olika handledningar (*tutorials*) och liknande. Jag har använt en sådan i början av mitt lärande av Angular. Det är ganska naturligt att härma i början. Men när förståelsen av hur olika element hänger ihop börjat komma har jag byggt kod i den riktning som passar min applikation. Mycket i ramverket tycks ytligt sett ske en smula magiskt. Det är svårare att se flödena i koden än i fall där koden ligger på en lägre nivå, liksom mer öppet istället för dolt av ett ramverk. Vartefter jag lyckats få till en sak har jag gått vidare till nästa för att bygga hela den tänkta applikationen. Ganska många gånger har jag återkommit till att snegla på handledningar för att sedan bygga något som har passat mina behov.

2.2 Angulars dokumentation

Angulars officiella dokumentation finns på https://angular.io. Angulars webbsidor innehåller mycket information och en del basexempel för hur olika tekniker är tänkta att användas. Jag har läst mycket på Angulars sidor och även när jag letat generellt på nätet, har sökresultaten ofta visat mig till en lämplig sida hos Angular.

2.3 Generella sökningar på internet

Angular har bra dokumentation. Men den är trots allt inte allomfattande. Jag har således också kontinuerligt gjort sökningar på internet för att förstå begrepp med mera.

I dagsläget hittar jag också oftare snabbt och aktuellt material på internet än i böcker. Vissa böcker inom informationsteknik är jättebra. Men utvecklingen inom informationsteknik går på många områden så snabbt att böcker ofta är föråldrade redan när de trycks. Det går förvisso ofta att nyttja kunskapen i böckerna ändå. Men för att få veta det senaste är internet i dagsläget ofta en bättre källa. Fast internet innehåller också fällor. Inte allt som skrivs på internet stämmer nödvändigtvis. Även gamla artiklar finns tillgängliga på internet och det är inte alla författare som vet tillräckligt mycket om vad de skriver om, alternativt skriver tillräckligt mycket för att råda folk att undvika göra dumheter. Således gäller det att vara uppmärksam och efter bästa förmåga bedöma om materialet på nätet är relevant och tillräckligt tillförlitligt. Böcker som återfinns på bibliotek kan i många fall vara föråldrade, men innehållet i dem har åtminstone i något skede blivit granskat. Eventuella buggar som upptäckts efter bokens tillkomst och eventuell kräver någon manuell åtgärd finns dock inte med i böcker.

3. ANGULAR

Angular är en plattform och ett ramverk för att bygga enkelsidiga klientapplikationer med HTML och TypeScript. Enkelsidig innebär att applikationen laddar in bara en webbsida och dynamiskt ändrar innehåll beroende på input från användaren (*SPA (Single-Page Application)*, n.d.). Angular är skrivet i TypeScript och använder således TypeScript-bibliotek. TypeScript är löst sagt ett striktare JavaScript.

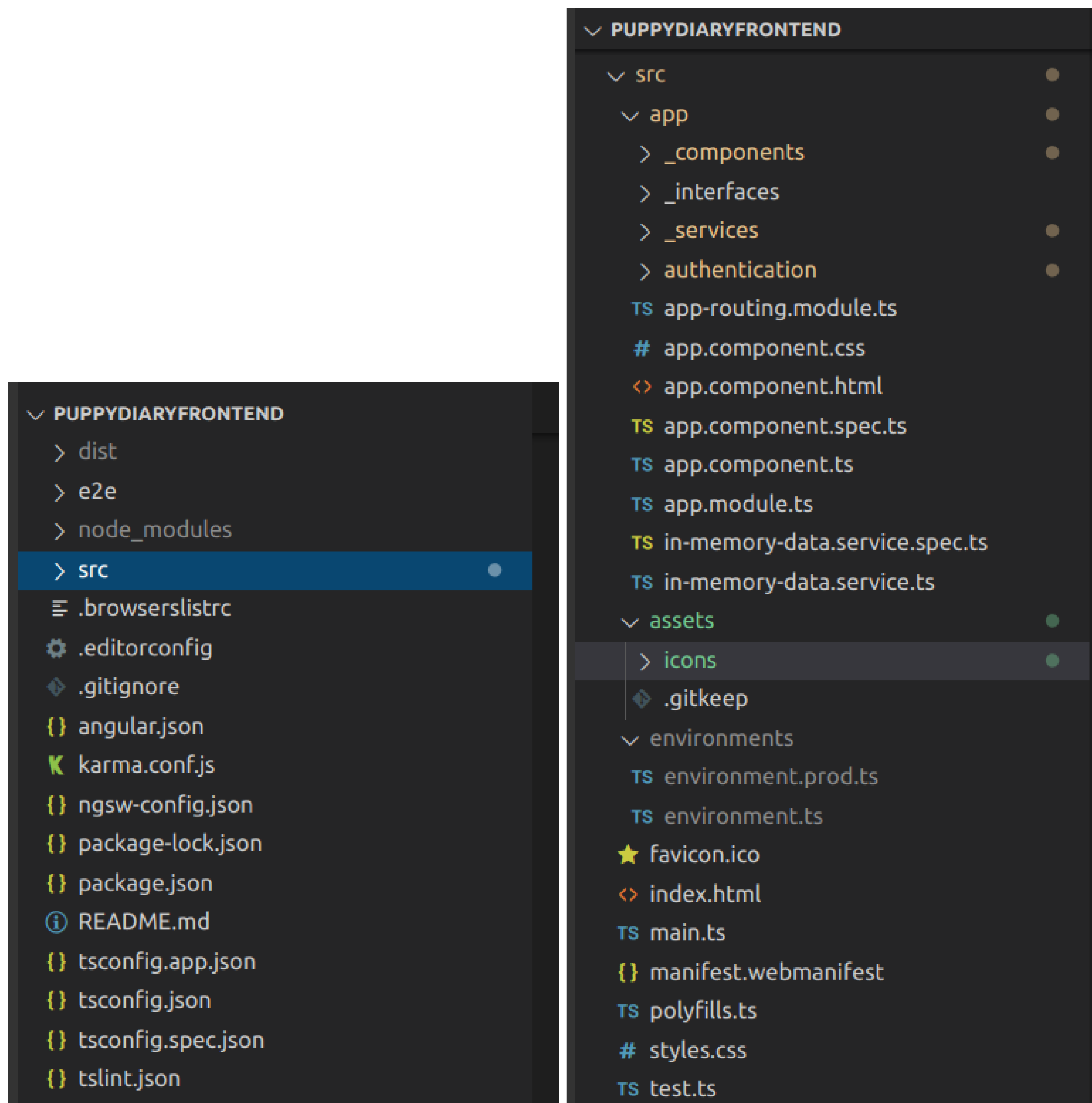
I det här kapitlet tar jag upp ett antal olika saker om Angular, vilka jag uppfattar som viktiga att notera för att jobba med Angular. Läser ni dokumentation eller liknande om Angular och något börjar med ng, så är det specifikt för Angular.

3.1 Angulars gränssnitt för kommandotolken

För att alls kunna jobba med Angular på ett smidigt sätt behöver programmerare installera Angular CLI (*Command Line Interface*) på sin arbetsdator. När gränssnittet är installerat kan det från inmatning i kommandotolken generera upp ett nytt projekt med kommandot *ng new appname*. Det ger en grundläggande struktur och förser projektet med vissa basfiler som varje projekt behöver. Angulars gränssnitt för kommandotolken är också praktisk att använda för att generera olika andra saker såsom komponenter och tjänster. Mer om olika delar i Angular nedan (*Angular*, n.d.-a).

3.2 Applikationens beståndsdelar

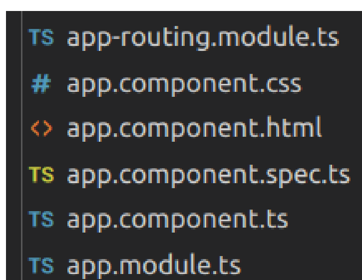
Såhär ser min applikationsstruktur ut i januari 2021, se figurerna 1 och 2. Det är en blandning av mappar och filer som autogenererats, som jag skapat, samt sådant som autogenererats och jag senare ändrat.



Figur 1 & 2. Aktuell projektstruktur, skärmdumpar från mitt projekt 14.01.2021

3.2.1 Några grundläggande filer

De kanske viktigaste filerna kommandot *ng new appname* genererar ur synvinkeln att komma igång med programmering, är ett antal filer namngivna app.någonting, se figur 3. De filerna är utgångspunkten för hur mycket i projektet jobbar.



Figur 3. app.filerna, skärmdump från mitt projekt 20.10.2020

Filen app-routing.module.ts (.ts är ändelse för typescriptfiler) bestämmer precis som namnet antyder hur ruttningen (*routing*) jobbar. Vilken exakt adress på webbsidan (*endpoint*) som jobbar mot vilken kod. I filen app.component.css finns formgivningsinformation. I filen app.component.html finns struktur för webbsidan. Här kan programmeraren placera logik för hur navigationen inom programmet sker. Ett viktigt element att placera är taggen `<router-outlet></router-outlet>` vilken är platsen app-routing.module.ts visar ruttat innehåll på. Filen app.component.spec.ts kan användas för att bygga upp tester. Filen app.component.ts innehåller logik för applikationens huvudsida (app.component.html). Slutligen innehåller filen app.module.ts förteckningar över importer, deklarationer, med mera applikationen behöver.

Utöver de här app-filerna finns också många andra viktiga filer såsom index.html och main.ts. De ligger på nivån högre än app-filerna och så att säga omsluter dem. Ännu en nivå upp finns ännu flera viktiga filer såsom package.json och angular.json. Filerna på de högre nivåerna bestämmer mycket över projektet, men jag går inte igenom allt om alla filer här eftersom det skulle bli en lång uppräkning och app-filerna är det jag närmast börjat att fokusera på för att komma igång.

Sammanhanget mellan några av de andra typerna av filerna i app-mappen kan till viss del generaliseras. Varje komponent i komponentmappen (*_component*) har ett begränsat uppdrag att sköta. De ska visa något särskilt och jobba med data för det. Det finns gränssnitt (i *_interfaces*) som fungerar för översättning av innehållet för att kunna kommunicera mellan *frontend* och *backend*. Och det finns tjänstefiler (i *_services*) som har som uppgift att hjälpa komponenterna med något specifikt, till exempel så att en tjänst hjälper till med kontakten med applikationens *backend* för ett gränssnitts innehåll.

3.2.2 Komponenter

Komponenter utgör de viktigaste byggstenarna för applikationer byggda med Angular. En komponent genererad med Angulars gränssnitt för kommandotolken med kommandot *ng g component componentname* består av fyra delar. Del ett är en HTML-fil som beskriver

strukturen på komponentens del av en webbsida. Den andra delen är en CSS-fil som bestämmer detaljer om formgivningen av komponentens HTML-del ifall det behövs något specifikt för aktuell komponent. Ifall *bootstrap* (här ett slags färdiga formgivningsregler som går att infoga) används för formgivningen är den här komponentspecifika CSS-filen ofta tom eller överhoppad. Del tre är en ts-fil som innehåller logik för komponentens beteende, hur den ska arbeta, vad den ska visa. Och slutligen så autogenererar kommandot för att skapa komponenter också en spec.ts-fil som kan användas för testning (*Angular*, n.d.-b). Ganska många av komponenterna i min applikation motsvaras av en tabell i databasen som mitt *backend* jobbar med.

Ts-filen som innehåller logik för komponentens beteende kan innehålla olika livscykelhändelser (*events*). Två av dessa är ngOnInit() och ngOnDestroy(). Båda är tillämpliga både för komponenter och direktiv.

Angular skriver att "ngOnInit() sköter initialiseringen efter att Angular visat data för egenskaper som är bundna till applikationens DOM och har satt de inskickade egenskaperna" (*Angular*, n.d.-c). DOM (*Document Object Model*) är datarepresentationen av innehållet i ett webbdokument (*Introduction to the DOM*, n.d.). Komponenternas konstruktorer ska hållas enkla. ngOnInit() kan med fördel användas för att sätta igång mer komplexa initialiseringar (*Angular*, n.d.-c). Till exempel kan applikationen hämta fram data från ett *backend* enligt instruktioner i ngOnInit().

ngOnDestroy() används för att försäkra sig om att det inte uppstår minnesläckor när Angular förstör komponenter och direktiv. ngOnDestroy() är också lämpligt att använda ifall någon annan del av applikationen behöver notifieras om att en komponent strax försvinner. För att undvika minnesläckor behöver resurser som inte automatiskt skärphanteras, frigöras manuellt istället. Således gäller det att avprenumerera på observerbara och DOM-händelser, stoppa tidtagarur och avregistrera återanrop (*callbacks*) som är registrerade med globala eller applikationsvisa tjänster (*Angular*, n.d.-c).

3.2.3 Gränssnitt

Gränssnitt används för att översätta information mellan *frontend* och *backend*. Angular rekommenderar att använda gränssnitt så långt som möjligt istället för regelrätta klasser för att mappa material.

3.2.4 Tjänster

En tjänst (*service*) är vanligtvis en specialiserad hjälpklass. Den kan göra en typ av uppgifter riktigt bra, till exempel förmedla kontakten med *backend* för ett enda gränssnitt. Många komponenter kan använda en tjänst (*Angular*, n.d.-d). Angulars gränssnitt för kommandotolken tillhandahåller ett kommando för att skapa tjänster, *ng g service servicename*. Angularkommandot generate har aliaset g. De kan fritt bytas ut mot varandra i kommandon. Kommandot för att skapa en tjänst genererar grunderna för både en service.ts-fil som ska innehålla själva tjänstfunktionalitéten och för en service.spec.ts-fil som är till för testning.

Att hålla komponenter skilt från tjänster främjar modularitet. Genom att hålla koden i mindre enheter blir det lättare att återanvända kodenheter även i lite olika sammanhang. Det blir också lättare att ändra kod, eftersom kodmassan att hålla reda på minskar.

3.2.5 Injicera beroenden

För att enkelt dra nytta av tjänster och andra kodenheter är det lämpligt att injicera beroenden (*dependecy injection*). I Angularprojekt ser det vanligtvis ut så att konstruktorerna har tjänster med mera som inparametrar. För att kunna skjuta in beroenden så använder vi annotationen `@Injectable()`. Använd i en tjänsteklass låter den annotationen Angular förstå att tjänsten ska gå att injicera i en komponent. Ramverket hjälper i bakgrunden till med att möjliggöra injektion av beroenden. Angular skapar en global injektor i uppstartsprocessen och flera vartefter det behövs. Injektorerna själva har hand om att skapa och hålla reda på beroenden. Det finns också något som kallas *providers*, som programmeraren i vissa fall behöver definiera i app.module.ts-filen. Dessa *providers* förklarar för injektorerna hur de ska få tag på eller tillverka beroenden (*Angular*, n.d.-d).

3.3 Dynamiskt innehåll i applikationen

3.3.1 Templates

Templates är enklast att tänka på som stycken av HTML-kod (*Angular*, n.d.-e). All HTML-kod i projektet utgör sammantaget applikationens strukturella skelett för det som ska presenteras. Ordet *template* förekommer på många ställen i Angulars dokumentation, men är inget särskilt magiskt i sig. Magin kommer in när programmeraren börjar dra nytta av funktionalitet Angular erbjuder för bindningar med mera mellan innehåll i HTML-filer och ts-filer.

För hålla koden redig och att kunna följa koden något sånär är det lämpligast att hålla HTML-kod i HTML-filer så långt som möjligt. Exempel som visar annat är inte ovanliga i *frontend*-kod. I Angular är index.html, app.component.html och komponenternas egna HTML-filer de typiska filerna för HTML.

3.3.2 Direktiv

I Angular går det att infoga direktiv i HTML-koden. Det finns två olika sorters inbyggda direktiv som kan infogas. Den ena sorten är strukturdirektiv, den andra sorten är attributdirektiv. Strukturdirektiv påverkar hur HTML blir arrangerad. De ändrar DOM-strukturen genom att påverka elementen de är ihopsatta med, till exempel genom att rada upp alla objekt i en lista oavsett listans storlek. Attributdirektiv påverkar innehållet i (och betendet för) element, attribut, egenskaper och komponenter (*Angular*, n.d.-f).

3.3.3 Strukturdirektiv

NgIf är som det låter ett villkorsdirektiv. Det lägger till eller låter tar bort undervyer från visning. NgIf är användbart istället för *null*-skrivningar i koden (*Angular*, n.d.-f). Genom att lägga till en asterix före ngIf, alltså *ngIf tolkar Angular det som ett helt `<ng-template>`-element paketerat runt det utpekade elementet. På så vis blir till exempel koden:

`<div *ngIf="hero" class="name">{{hero.name}}</div>` att betyda:

```
<ng-template [ngIf]="hero">
   <div class="name">{{hero.name}}</div>
</ng-template>
```

NgFor är ett direktiv som upprepar en nod för varje föremål i en lista (*Angular*, n.d.-f). I det fallet rör det sig om microsyntax under ytan. Ett typiskt exempel på ngFor är:

```
<div *ngFor="let item of items">{{item.name}}</div>
```

För att fungera måste items motsvaras av ett lämpligt attribut med namnet items i den tillhörande ts-filen (i komponenten). Helt naturligt, men det gäller att hålla tungan rätt i munnen med vad som fritt kan namnges i HTML-kod och vad som måste matcha ts-kod i tillhörande komponent, eller gränssnitt eller både och. Noterbart är också att namngivningar måste stämma också med projektets *backend*, så att till exempel Spring (ett ramverk för Java) kan översätta material rätt och vice versa.

NgSwitch är en sätt att välja mellan olika HTML-kod beroende på vad i switchen som matchar (*Angular*, n.d.-f). Det är lätt att se att de här direktiven sparar en del kodskrivande. Kodexemplen i det här kapitlet är tagna från Angulars egen dokumentation (*Angular*, n.d.-g).

3.3.4 Attributdirektiv

Bland attributdirektiven tycker jag NgModel är viktigast. NgModel ger en tvåvägsbindning till ett HTML-element. Det här gör att data aktivt kan bollas mellan HTML-filer och ts-filer och att bollandet visas genom ändrade värden i HTML-elementet. För att använda NgModel på det här sättet behöver programmeraren importera FormsModule till aktuell NgModule, i mitt fall app.module.ts. Många NgModules reglerar sina egna attributdirektiv. RouterModule och FormsModule är två exempel på det.

För att NgModel ska kunna översätta informationen för ett HTML-element behöver NgModel stöd av en ControlValueAccessor. Behändigt nog är Angular försett med sådana här tillbehör för att hantera värden för alla grundläggande HTML-formulärelement. Men om programmeraren vill använda ett element som inte hör till de vanliga formulärelementen eller om kodaren vill använda något tredjeparts HTML-element, så är programmeraren tvungen att skapa eller infoga en fungerande ControlValueAccessor (*Angular*, n.d.-f).

Det går att gå runt ngModel och använda Angulars tvåvägsbindning genom att nyttja namngivna värden och egenskaper. Separerade ngModel-bindningar kan ändå göras enklare med [(ngModel)]-syntax (*Angular*, n.d.-h). Figurerna 4, 5 och 6 visar hur några olika filer så att säga sitter ihop. Figur 7 visar syntax och tillhörande information för bindningar.

```
<label>type:
    <input [(ngModel)]="deworming.medication" placeholder="medication"/>
</label><br>
```

Figur 4. Del av deworming.component.html ,skärmdump från mitt projekt 10.01.2021

```
@Component({
  selector: 'app-dewormings',
  templateUrl: './dewormings.component.html',
  styleUrls: ['./dewormings.component.css']
})
export class DewormingsComponent implements OnInit {

  deworming: Deworming;
```

Figur 5. Del av deworming.component.ts ,skärmdump från mitt projekt 10.01.2021

```
export interface Deworming {
    id: number
    puppyId: number;
    date: string;
    medication: string;
}
```

Figur 6. Del av deworming.ts ,skärmdump från mitt projekt 10.01.2021

Type	Syntax	Category

Interpolation
Property
Attribute
Class
Style
	`{{expression}}`
	`[target]="expression"`
	`bind-target="expression"`
		One-way
		from data source
		to view target

Event	`(target)="statement"` `on-target="statement"`	One-way from view target to data source
Two-way	`[(target)]="expression"` `bindon-target="expression"`	Two-way

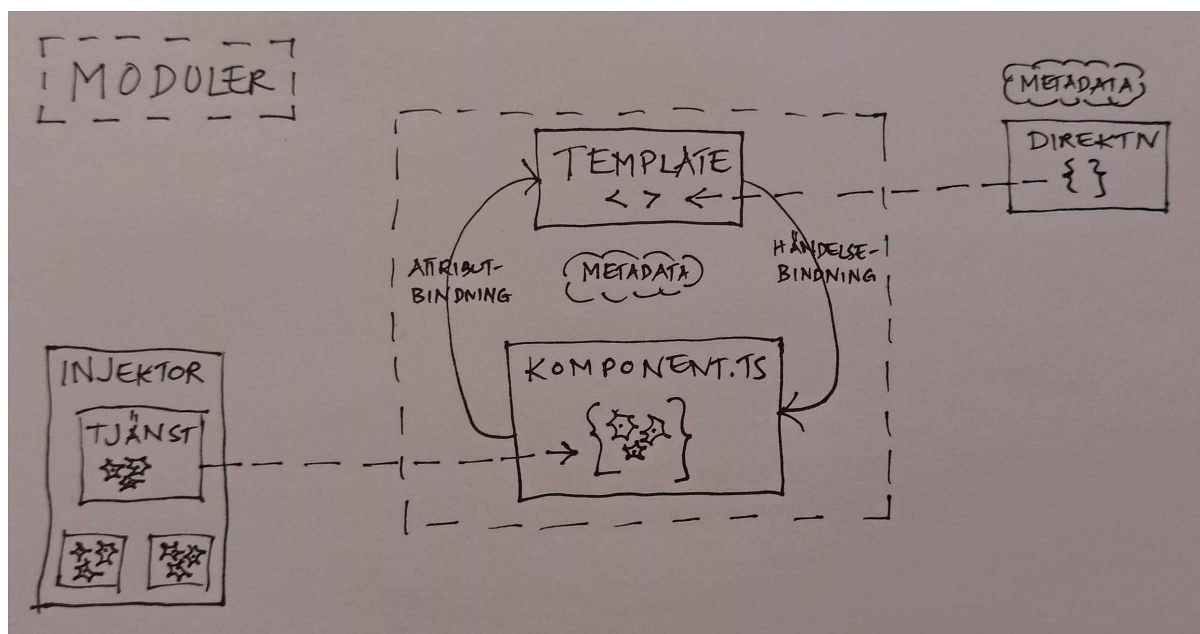
Figur 7. Angularbindningar. Bilden hämtad från https://angular.io/guide/binding-syntax 08.01.2021

NgForm är också ett mycket användbart attributdirektiv. Det är också möjligt att använda NgForm tillsammans med NgModel. På så vis blir det möjligt att reagera på inmatad data, till exempel informera om inkorrekt data.

Liksom NgModel kommer också NgForm från FormsModule. När FormsModule är importerat blir ngForm automatiskt aktivt på alla <form>-taggar. Det är ändå möjligt att lägga in ngForm direkt HMTL-kodstycken. Det är praktiskt för att komma åt formulärets innehåll som en helhet och för att kontrollera giltigheten för formuläret, samt egenskaper såsom *dirty* (ändrad) och *touched* (rörd). Också inom sådan här formulär finns det möjlighet att använda ngModel med namnattribut. På så vis går det att kontrollera delar av formuläret. För att bestämma vad som ska hända när formuläret skickas lägger programmeraren in kod i metoden onSubmit(). Ifall programmeraren absolut inte vill använda ngForm behöver programmeraren använda ngNoForm eller i fallet med *reactive forms* (reaktiva formulär) avstå från att använda FormGroup-direktivet (*Angular*, n.d.-i).

3.4 Hur några av delarna i Angular hänger ihop

Applikationer gjorda med Angular består av många element. Figur 8 visar hur komponenter jobbar med några andra element.



Figur 8. Så här beskriver Angular hur komponenter jobbar. Bilden baserar sig på Angulars material.

Lejonparten av en vy är definerad av en komponent inklusive dess HTML-kod. En dekoratör för en komponentklass lägger till metadata. Direktiv och bindningar i komponentens HTML-fil ändrar vyer enligt data och logik i applikationen. En injektor gör att en komponent kan använda tjänster (*Angular*, n.d.-j).

3.5 Ytterligare hjälpmedel i Angular

3.5.1 Observerbara

Observerbara (*Observables*) används mycket i Angular. Det observerbara gör är att de hjälper till att skicka meddelanden mellan olika delar av applikationer (*Angular*, n.d.-k).

Observerbara lämpar sig för händelsehantering (*event handling*). De passar också för att hantera flera värden. Observerbara kan nämligen leverera många värden av vilken typ som

helst. Observerbara lämpar sig även för asynkron (tidsmässigt oberoende interaktion) programmering. Det går ändå att jobba både asynkront och synkront (tidsmässigt beroende interaktion) med observerbara och använda samma API för bådadera (*Angular*, n.d.-k).

Att använda observerbara handlar om att utnyttja ett designmönster som heter *observer*. I mönstret finns ett objekt (tänk objektorientering), självklart nog kallat subjekt (tänk grammatik), som håller reda på dem som är beroende av subjektet. Subjektet meddelar automatiskt åskådarna (*the observers*), vilket är namnet på de som är beroende av subjektet, om förändringar (*Angular*, n.d.-k).

Observerbara är deklarativa, vilket medför att programmeraren aktivt måste prenumerera (*subscribe*) på dem för att de ska köra och meddela prenumeranten sin information. Prenumerationen fortsätter tills programmeraren lagt in ett funktionsanrop för att sluta prenumerera (*unsubscribe*) på den observerbara eller tills funktionen kört klart (*Angular*, n.d.-k). I sammanhanget kan nämnas att AsyncPipe avprenumererar självt (*Angular*, n.d.-l). Så observerbara levererade av tjänsteklassen HttpClient i nästa kapitel behöver inte avprenumereras manuellt eftersom de använder AsyncPipe.

3.5.2 HTTP API

För att ta sköta kontakten med mitt *backend* har jag tagit hjälp av Angulars HTTP API, tjänsteklassen HttpClient i @angular/common/http. Användning an HTTP API:et kräver att programmeraren lägger till import av HttpClientModule. Vanligtvis sker importen i AppModule (*Angular*, n.d.-m).

Tjänsteklassen HttpClient hjälper till med mycket. Den ger möjlighet att begära typade svarsobjekt och att genskjuta (*interception*) anrop (*requests*) och svar (*responses*) som skickas med HTTP-protokoll. Tjänsteklassen ger också smidig felhantering och testningsegenskaper (*Angular*, n.d.-m).

Tjänsteklassen HttpClient erbjuder möjlighet till get-, put-, post- och delete-anrop. Det finns många alternativ för att sätta upp regler för kommunikationen. Till exempel går det att ställa

in vad svaret borde innehålla, såsom något i json-format. Och det är lämpligt att bestämma vilken typ svaret ska utgöra. För att Angular ska kunna reda ut typen korrekt behövs gränssnitt som matchar den bestämda typen (*Angular*, n.d.-m). Figur 9 visar ett exempel på en metod för HTTP-anrop.

```
/** GET litters from the server */
getLitters(): Observable<Litter[]> {
  return this.http.get<Litter[]>(this.littersUrl)
    .pipe(
      tap(_ => this.log('fetched litters')),
      catchError(this.handleError<Litter[]>('getLitters', []))
    );
}
```

Figur 9. HTTP-anrop. Bilden är hämtad från mitt projekt.

HTTP-anrop kan misslyckas. Varje tjänst som använder tjänsteklassen HttpClient behöver hantera fel. Lämpligtvis lägger programmeraren till en privat metod för felhantering. Metoden bör kunna hantera objekt från HttpErrorResponse-klassen som input, för fel både hos *backend* och *frontend* resulterar i HttpErrorResponse-objekt (*Angular*, n.d.-m).

Genom att skapa genskjutare (*interceptors*) kan programmeraren få programmet att kontrollera och ändra HTTP-anrop och -svar. Det går också att länka ihop flera genskjutare. För att skapa en genskjutare behöver programmeraren skapa en klass som implementerar HttpInterceptor-gränssnittet och implementera metoden intercept(). De flesta genskjutare kontrollerar anropet och skickar anropet till metoden handle() hos nästa genskjutare i kedjan av objekt som implementerar HttpHandler-gränssnittet. Det är vanligt att genskjutare ändrar innehållet i anrop (*Angular*, n.d.-m). Jag har använt en genskjutare som en del av att hantera autentisering.

3.6 Beroenden

3.6.1 node_modules och npm

Angular använder som bekant TypeScript som bygger på JavaScript. Och Typescript transpilerar till Javascript. Mot den bakgrunden är det föga förvånande att hitta en mapp dedikerad till node_modules i Angularprojekt.

Npm är ett mjukvarubibliotek för JavaScript (*About Npm*, n.d.). När programmeraren hämtar material från npm, så hamnar det typiskt i en mapp kallad node_modules. Ramverket Angular, Angular CLI och komponenter som används av Angularapplikationer paketeras som npm-paket och distribueras via npm-biblioteket (*Angular*, n.d.-n). För att kunna använda npm behöver programmeraren installera npm CLI klienten. Den installeras och körs som en node.js-applikation. Det är också så att Angulars gränssnitt för kommandotolken använder npm-klienten ifall inget annat särskilt är konfigurerat. Alltså leder vägarna som standard till node.js. Node.js är en körmiljö för JavaScript och används för att köra JavaScript utanför en webbläsare. Node.js har öppen källkod och är användbart på olika plattformar (*Node.js*, n.d.), (*Node.js Introduction*, n.d.).

Node_modules-mappen innehåller både Angulars egna moduler, tredjepartsbibliotek och polyfillpaket. Polyfillpaketen bryggar luckor i webbläsares JavaScript-implementationer (*Angular*, n.d.-n).

Alla paket applikationen behöver listas i filen package.json. En del moduler behöver också importeras i app.module.ts-filen eller motsvarande för att Angularapplikationen ska fungera.

Några andra exempel på npm-material jag använt är @angular/core, @angular/material, @angular/router, angular-mydatepicker, angularx-social-login och http-server för att bara nämna några.

Ett paket som är bra att minnas är *angular-in-memory-web-api.* Före programmeraren har tillgång till eller vill använda ett *backend*, så är det möjligt att använda det för att sätta upp påhittad data för att pröva applikationen i inledningsfasen. Men författarna av API:et påpekar att det är experimentellt, att de kommer att göra drastiska förändringar och att det inte är avsett för produktion (*Npm: Angular-in-Memory-Web-Api*, n.d.). För att initialt pröva att applikationen börjar jobba ungefär som tänkt, är det här API:et ändå tillräckligt.

3.6.3 Web Storage API

Web Storage API tillhandahåller lokala förvar (jämför med databaser) i användarens webbläsare. Där lagras data som nyckel-värdepar. Data kan sparas, ändras, hämtas och tas bort. Den tillåtna storleken på förvaringen är begränsad. API:et har två olika sorts förvaring sessionStorage och localStorage. De fungerar skilt från varandra (*Web Storage API*, n.d.).

Det finns ett gränssnitt som heter Window. Det representerar ett fönster med ett DOM-dokument. Varje flik i en webbläsare har ett eget Window-objekt. Gränssnittet innehåller många olika saker. Ledstjärnan är global tillgänglighet (*Window*, n.d.). Web Storage API:et utvidgar Window-objektet.

sessionStorage når vi genom Window.sessionStorage. sessionStorage har skilda förvar för varje ursprung. Förvaren är tillgängliga lika länge som sidans session finns, alltså så länge webbläsarfliken är öppen, även om användaren laddar om eller återställer en sida. Datan i förvaret försvinner när sessionen avslutas (*Web Storage API*, n.d.). Nya flikar får nya sessioner med samma värde som toppnivåns kontext (*Window.sessionStorage*, n.d.). Jag har använt sessionStorage för att hantera information om genomförd inloggning.

localStorage når vi genom Window.localStorage. localStorage är annars som sessionsStorage, men datan blir kvar också när webbläsarfliken eller hela webbläsaren stängs. För att få bort data från localStorage behöver användaren antingen använda JavaScript (funktioner som removeItem() eller clear()) eller tömma webbläsarens *cache* (lokalt sparad data) (*Web Storage API*, n.d.).

4. SPRÅK OCH TEKNIK

Angular erbjuder bekväma sätt att göra mycket. Men att använda Angular blir mycket lättare med en del förkunskaper.

4.1 Objektorienterad programmering

I objektorienterad programmering (OOP) fokuserar vi på objekt som innehåller både data och beteenden. Datan finns i fält som kallas attribut eller egenskaper, beteendena finns i något som brukar kallas metoder.

I motsatsmetoden procedurell programmering är fokuset beteendet, där kallat funktioner. Funktionerna används för att påverka data. I procedurell programmering är funktionerna inte bundna inom objekt.

OOP:s objekt definieras i klasser. Och OOP har fyra omtalade principer. Principerna är arv, inkapsling, abstraktion och polymorfism. Arv betyder att en klass ärver data och beteenden från en annan klass. Det är praktiskt för att slippa skriva samma kod igen. Behöver applikationen trots allt en speciell lösning i den ärvande klassen, lägger programmeraren in kod för det i den ärvande klassen. Inkapsling handlar om att reglera vad som är tillgängligt för vem, till exempel tillgängligt bara för användning inom den egna klassen, privat eller tillgängligt för alla, publikt. Abstraktion handlar om representation utan att nödvändigtvis berätta all underliggande information. Till exempel en klass är en abstraktion av data och beetenden, i vilken bara vissa delar är publika och således uppenbart åtkomliga för alla. Polymorfism avser att klasser delar ett gränssnitt och har inom sig varierande implementationer. Det finns således likheter men också skillnader mellan klasserna (*What Is Object Oriented Programming? OOP Explained in Depth*, n.d.).

Angular har i grunden en uppbyggnad som är förenlig med OOP.

4.2 Språk

4.2.1 Typescript

Typescript är ett språk med öppen källkod. Typescript bygger på Javascript genom att lägga till statiska definitioner av typer (*Typed JavaScript at Any Scale*, n.d.). Typescript transpilerar (kompilerar källkod från ett språk till källkod i ett annat språk) till Javascript och är utvecklat av Microsoft.

Eftersom TypeScript bygger på JavaScript finns möjligheten för funktioner att finnas utanför klasser. Det avviker från normal OOP, där allting är bundet i klasser. Men TypeScript stöder många vanliga mönster såsom arv, att implementera gränssnitt och statiska metoder. TypeScript har strukturella typer vilka jobbar med strukturella jämförelser istället för nominella typer, vilka kräver exakta deklarationer. Typen är alltså inte lika strikt som inom exempelvis Java, men på kodskrivningsnivå betydligt striktare och användbarare än i JavaScript. Några saker i Typescript som kan förvåna en programmerare med bakgrund i OOP är att en tom typ (klass) kan bete sig annorlunda än väntat i och med att den strukturella typen utvärderas, att identiska typer (klasser) kan blandas och att reflektion inte fungerar (*Documentation - TypeScript for Java/C# Programmers*, n.d.).

5.2.1.1 För- och nackdelar

Finns det någon generell anledning att välja Typescript istället för JavaScript? TypeScript har statisk typning, vilket JavaScript saknar. Den statiska typningen är en fördel. Det är dock inget tvång att utnyttja statisk typning. Programmerare är även fria att välja att delvis använda statisk typning. När statisk typning väl har deklarerats för en variabel, så kan den variabeln bara ha vissa värden. Kompileraren fångar typrelaterade misstag. Det här gör koden mindre felbenägen i produktionsstadiet. Koden får också mer struktur och blir lättare att förstå med statisk typning. Det underlättar för grupper att arbeta tillsammans med kod, samt underlättar underhåll och vidareutveckling av kod (*The Good and the Bad of TypeScript*, 2020).

Till TypeScripts nackdelar hör att det inte är ett har helt klockren statisk typning på grund av att TypeScript transpileras om till JavaScript för att köras. Och jämfört med JavaScript blir den totala kodmängden större (*The Good and the Bad of TypeScript*, 2020).

4.2.2 JavaScript

JavaScript är ett programmeringsspråk vi kan förvänta oss hitta i webbsammanhang. JavaScript är ett skriptspråk (tolkat språk) som låter programmeraren skapa dynamiskt innehåll och interaktiva element. Ett exempel är att animera bilder. JavaScript baserar sig i sin tur på ett annat skriptspråk, ECMAScript (*JavaScript*, n.d.-a).

Javascript är ett tolkat eller *just-in-time* kompilerat programmeringsspråk. JavaScripts källkod behandlas på klientsidan, alltså av användarens webbläsare. JavaScript är ett högnivåspråk, som använder sig av prototypbaserad programmering, vilket har beskrivits som ett slags objektorientering i vilket återanvändande av beteenden använder prototyper av befintliga objekt. Javascript stöder flera paradigmer, både objektorienterad och procedurell programmering. Javascript använder dynamisk typning av variabler. Variabler utvärderas alltså först under körning (*JavaScript*, n.d.-b).

JavaScript återfinns dels i skilda js-filer, dels kan JavaScript finnas infogat i <script>-taggar i HTML-filer. Just blandning mellan HMTL och JavaScript i varandras filer gör applikationers kod mycket svårare att följa. Min erfarenhet är att det är bäst att hålla sig till separation av ansvarsområden så långt som möjligt i detta. Koden blir betydligt mer självdokumenterande och enklare att jobba vidare med så.

4.2.3 HTML

HTML står för *HyperText Markup Language*. Det är standard märkspråk för webbsidor. HTML-element utgör byggstenarna för strukturen på webbsidor. Elementen ser ut som <> med något emellan, till exempel <br>. Många av elementen utgör par med <> i början och </> i slutet av elementet. HTML-elementen kan ha olika attribut (*What Is HTML*, n.d.). Figur 10 visar ett exempel på HMTL.

```html
<div class="noprint">
  <label>New puppy name: <input #puppyName /> </label>
  <button (click)="add(puppyName.value); puppyName.value=''"> add </button>
</div>

<ul *ngIf='puppies' class="puppies">
  <li *ngFor="let puppy of puppies">
    <a routerLink="/puppydetail/{{puppy.id}}">
      <span class="badge">{{puppy.id}}</span> {{puppy.name}}</a>
    <button class="delete" title="delete puppy" (click)="delete(puppy)">x</button>
  </li>
</ul>
```

Figur 10. Exempel på HTML. Bilden är från mitt projekt.

4.2.4 CSS

CSS är en förkortning av *Cascading Style Sheets*. CSS har hand om utseendet på HTML-element. CSS kan påverka utseendet för vad som syns både på skärmen, på utskrifter och andra media. CSS kan tillämpas på en generell nivå och bli att gälla en hel applikation (*CSS Introduction*, n.d.). Detaljer i någon del av applikationen kan utformas skilt genom att peka ut dem särskilt i HTML-koden.

Något om mina CSS-val. Jag har valt att göra en egen CSS-formgivning. När jag undersökte Angulars uppbyggnad med bland annat en generell CSS-fil och egna CSS-filer för varje komponent föll det sig naturligt att undersöka vad det skulle leda till. Jag har främst tittat på att webbsidan skulle se okej ut från en dator, eller större skärm, men jag har gjort också en viss anpassning till mindre skärmar. Jag har också lagt in anpassningar för att få lämpligare utskrifter, som inte innehåller header, footer, knappar etc.

Det finns möjlighet att använda färdiga formgivningar genom att använda *bootstrap CSS*. (Installera *bootstrap* och lägg till en tagg <link href..> till önskad *CSS bootstrap* i index.html-filen och sätt vald CSS-stil i angular.json-filen. Beroende på hur omfattande användningen av *bootstrap* blir kan det också vara nödvändigt att ta med skripttaggar som hänvisar till vald *bootstrap* (Otto et al., n.d.).) För att använda *bootstrap* effektivt behöver dock programmeraren kunna klasserna i *bootstrap* väl. För *bootstrap* utgår från att HTML-koden innehåller klasshänvisningar till *bootstrap* för att skapa kopplingar till klasserna som ligger till grund för att generera utseende genom *bootstrap*. Det här är givetvis

ett extra moment att lära sig. Och jag har redan undersökt CSS tillräckligt för ett projekt, så jag har inte gått längre med *bootstrap* CSS. Nu i ett senare skede när jag kommit så långt att jag lärt mig lite om progressiva webbapplikationer, kan jag konstatera att det varit fördelaktigt att nyttja *bootstrap* och strunta i komponentbunden CSS så långt som möjligt. Progressiva webbapplikationer framhåller anpassning för olika skärmstorlekar.

4.2.5 JSON

JSON står för *JavaScript Object Notation* och är ett format för att utbyta data. Att använda JSON för att skicka data mellan klient och server och andra vägen är det typiska användningsområdet för JSON. Min applikation använder JSON för att kommunicera data med sitt *backend*.

Som namnet antyder baserar sig JSON på en särskild version av JavaScript. Trots det är JSON ändå språkoberoende. Formatet bygger på två strukturer som i någon form återfinns i de flesta moderna programmeringsspråk. Den ena strukturen är en samling namn-värdepar. Den andra strukturen är en lista av värden. JSON är självbeskrivande och lätt att begripa för både människor och maskiner (*JSON*, n.d.).

```
{"puppies":[
        {"name":"Clarissa", "gender":"female", "ribbon":"pink"},
        {"name":"Clark", "gender":"male", "ribbon":"blue"},
        {"name":"Claire", "gender":"female", "ribbon":"yellow"}
]}
```

Figur 11. JSON-exempel. Exempel jag hittat på 18.01.2021.

4.3 Andra hjälpmedel

4.3.1 Visual Studio Code

Jag har använt Visual Studio Code (VS Code) vid utvecklingen av mitt projekt. VS Code är en välkänd programutvecklingsmiljö. Den har öppen källkod och fungerar för Windows, Linux och Mac OS. VS Code är erbjuder sådant som *IntelliSense*, som ger smart komplettering av koden, *Run and Debug* för att köra och debugga kod direkt i VS Code, samt inbyggda Git-kommandon. VS Code ger också möjlighet att lägga till många utvidgningar,

även tredjepartsutvidgningar, för stöd för olika språk med mera (*Visual Studio Code - Code Editing. Redefined*, 2016). Jag upplever att VS Code har bra dokumentation och har kunnat hjälpa mig göra allt jag velat på ett smidigt sätt.

4.3.2 GitHub

För att säkerhetskopiera mitt projekt, så har jag skapat ett konto på GitHub och kontinuerligt laddat upp kod dit. Det skulle varit hemskt att förlora hela arbetet om min arbetsdator fallerat.

GitHub är en webbaserad värdplattform för versionshantering av programmeringsprojekt och för samarbete mellan programmerare. Versionshanteringen sker med versionshanteringssystemet Git. Det är gratis att bli användare hos GitHub, men det går att få tillgång till tilläggsfunktionalitet som betalande användare (*Hello World · GitHub Guides*, n.d.). Det är lätt för användaren att dela sina förvar, vilka innehåller användarens projekt, antingen med utvalda personer eller med alla som har tillgång till internet.

5. PROGRESSIVA WEBBAPPLIKATIONER

5.1 Allmänt

Jag kan konstatera att progressiva webbapplikationer (PWA) inte har någon formell standard. Så vad exakt som krävs för att kalla något för en PWA kan säkert diskuteras. Mozilla nämner vad de anser vara de tekniska minimikraven, nämligen att sidan tillhandahålls med HTTPS-protokoll (säker kontext), att applikationen har en eller flera tjänstearbetare (*service workers*) och att applikationen har en webbmanifestfil (*Progressive Web Apps (PWAs)*, n.d.). I en artikel från Googles utvecklingssidor på webben 2015 ser förklaringen till vad en PWA är lite annorlunda ut. Google listar 10 punkter en PWA bör vara. Enligt Google ska en PWA vara progressiv, alltså fungera oavsett val av webbläsare. En PWA ska också vara responsiv, anpassa sig till olika skärmar. En PWA ska fungera även vid svag eller otillräcklig nätverkskontakt. Vidare ska en PWA påminna om en traditionell lokalt installerad applikation avseende navigation och interaktion. Uppdateringsprocessen ska vara automatisk. Kontakt över nätverk ska vara krypterad. En PWA ska gå att hitta med sökmotorer. En PWA ska locka till fortsatt användning, till exempel genom att använda *push*-notifikationer. Det ska gå att installera en PWA och den ska gå att dela genom att ge URLen (*Getting Started with Progressive Web Apps*, n.d.). Google har således en beskrivning som inriktar sig mer på funktionalitet som bör finnas.

När jag letat runt på internet, så har jag hittat lite varierande uppfattningar om vad en PWA bör klara av. Några saker som återkommit har varit, att applikationen ska gå att installera (kunna starta från en ikon på klienten), att applikationen bör klara av att fungera även om kontakten med internet är svajig och att *push*-notifikationer bör ingå.

För att kunna vara installerbar behöver programmeraren utöver att fylla de tekniska minimikraven Mozilla beskriver också tillhandahålla en ikon för att representera applikationen i den lokala miljön (*How to Make PWAs Installable*, n.d.). Praktiskt nog går det för utveckling trots kravet på HTTPS (krypterad anslutning) att använda localhost (*Angular*, n.d.-o).

5.2 För- och nackdelar

Varför göra en applikation till en PWA? Det kräver rätt mycket arbete att göra en PWA som utnyttjar mer än de mest rudimentära möjligheterna hos PWA.

PWA:n påstås vara en kombination mellan webbsidor och lokalt installerade applikationer (*native applications*). Några fördelar kan vara att PWA:n klarar dålig kontakt med internet och laddar snabbt. De ska locka användare till återbesök genom att kunna startas i ett dedikerat fönster direkt från en ikon på hemskärmen/-miljön. De ska kunna göra saker som traditionellt bara klassiska lokalt installerade applikationer kan, till exempel använda enhetens sensorer (*What Are Progressive Web Apps?*, n.d.). Och en PWA är lätt att dela. Det räcker att ge URLen (IQUII, 2019).

Vad är möjliga nackdelarna med PWA:n? Nuförtiden är folk lärda att bara installera saker från betrodda källor såsom Google Play eller App Store. Det finns inget övervakningsorgan för installerbara PWA:n. PWA:n laddar användare bara ner direkt från webbsidor. Det är folk varnade för. Webbläsare har begränsningar på hur mycket som kan lagras i dem. Inte alla enheter, operativsystem, webbläsare stöder PWA:n fullt ut (IQUII, 2019). Åtkomsten till funktionalitet i den lokala miljön varierar (Poot, 2020).

5.2.1 Progressiv

Progressiv är ett viktigt koncept för PWA. Det finns med i namnet för PWA och det är den första punkten Google nämner i artikeln från 2015 jag läste för att på något sätt kunna definiera PWA. Men ännu idag är det ett ord som inte är fullt stött av alla på marknaden.

Att en PWA ska vara progressiv innebär att den ska fungera oavsett val av webbläsare. De flesta stora operativsystem och webbläsare har numera åtminstone ett grundläggande stöd för PWA. Men det finns undantag. Tjänstearbetare plus möjlighet att installera kan sägas vara två grundstenar för PWA. Tjänstearbetare stöds inte av ett antal äldre webbläsare som fortfarande är i användning. Möjlighet att installera (lägga till enhetens skärm) saknas för vissa fall såsom Safari på Mac OS, Firefox på *desktop*, Chrome på iOS och iPadOS, IE/gammal Edge,

Opera på *desktop* och några andra webbläsare (*Progressive Web Apps in 2021*, n.d.). Hur mycket stöd och hurdan implementation de olika operativsystemen har för PWA i övrigt varierar. Till exempel hur mycket lokalt lagringsutrymme en PWA kan använda varierar (Poot, 2020).

5.3 PWA med hjälp av Angular

Det finns ett npm-paket avsett för att hjälpa till att bygga PWA med Angular. Det går att starta ett nytt projekt som en PWA eller att lägga till PWA-funktionalitet senare. Här är kommandot för att lägga till PWA i ett senare skede, *ng add @angular/pwa --project* (--project ska vara projektnamnet som det står i angular.json) (*Angular*, n.d.-p).

Det kommandot gör följande (*Angular*, n.d.-p):

- Lägger till paketet för tjänstearbetare
- Sätter igång stöd i gränssnittet för kommandotolken för att bygga applikationen med tjänstearbetare
- Lägger till en tjänstearbetare i filen app.module.ts
- Uppdaterar filen index.html
- Lägger till ikoner att använda för installation. De här ikonerna vill programmeraren säkert byta ut till egna ikoner. För det är Angulars logo på alla ikoner Angular genererar.
- Skapar en konfigurationsfil kallad ngsw-config.json för tjänstearbetaren. Den filen bestämmer bland annat hur *cachning* sker.
- Tillför filen manifest.webbmanifest

Det här ger applikationen grundläggande stöd och funktionalitet för PWA. Angular gör så att vi efter lite konfiguration kan köra bygga och vår applikation som en PWA som är installerbar, laddar snabbt och uppdaterar sig själv i bakgrunden. Med ytterligare lite konfiguration kan vi få vår applikation att *cachea* material, så att vi får ett visst motstånd mot svag nätverkskontakt.

Men för att göra en applikation till en PWA som använder den fulla kraften hos en PWA, behöver programmeraren koda mer.

5.4 Tjänstearbetare (*service workers*)

Tjänstearbetare (*service workers*) är en nödvändig och central del av en PWA. Enligt Angular är en tjänstearbetare i grunden är "ett skript som körs i webbläsaren och tar hand om *cachning* för en applikation" (*Angular*, n.d.-o). En tjänstearbetare kan dock programmeras att göra mera.

Tjänstearbetare snappar upp alla utgående HTTP-anrop och kan välja hur anropen hanteras. En tjänstearbetare fungerar alltså som en nätverks-*proxy*. Tjänstearbetare kan utnyttja API:n eller projektspecifikt skriven kod. Tjänstearbetare är helt programmerbara. En tjänstearbetare blir kvar i en webbläsare även efter att webbläsaren stängts ner. Vanliga användningsområden är att vara en del av är att hantera svag nätverkskontakt och att snabba upp uppstarter (*Angular*, n.d.-o).

För Angulars del är de viktigaste filerna att känna till för tjänstearbetare en webbmanifestfil med namnet manifest.webmanifest och en konfigurationsfil som heter ngsw-config.json.

5.5 Klara svag nätverkskontakt och helt *offline*

För att klara av svajig nätverkskontakt behöver applikationen allra minst *cachea* material. Om det handlar om en sida som jobbar med möjlighet att påverka material i en databas behöver det finnas ett lokalt mellanförvar som synkroniserar med servern. Mellanförvaret tar över när nätverkskontakten är otillräcklig.

Till exempel är IndexedDB API eller localForage API möjliga att använda som mellanförvar. IndexedDB API:et är ett förvar på klientsidan och kan spara avsevärda mängder strukturerad data, även filer/blobbar. IndexedDB API:et är kraftfullt, men lite komplicerat. localForage fyller i princip samma syften, men är enklare att använda. localForage är en paketerare (*wrapper*) runt IndexedDB. localForage använder IndexedDB i bakgrunden för de

webbläsare som tillhandahåller IndexedDB. För övriga webbläsare använder localForage webSQL och localStorage (*IndexedDB API*, n.d.).

5.6 *Push*-notifikationer

Push-notifikationer är enkla meddelanden. Meddelandena gör mottagaren uppmärksam på något, såsom att avsändaren önskar att mottagaren ska göra något speciellt (PushPro, n.d.). Företag använder *push*-notifikationer bland annat för att göra sina kunder mer engagerade och i marknadsföringssyften.

En server skickar ut *push*-notifikationer och information för att nå rätt mottagare till en webb-*push*-notifikationtjänst. Webb-*push*-notifikationer använder en tjänstearbetare för att lyssna på *push*-händelser applikationens webb-*push*-notifikationstjänst skickar (PushPro, n.d.).

Webb-*push*-notifikationer stöds inte av iOS (Poot, 2020). Det är något av en motsägelse med hänsyn till att jag på ett antal ställen sett *push*-notifikationer lyfta i PWA-sammanhang.

5.7 Reflektion

PWA erbjuder intressanta möjligheter och med kunskap om vad en PWA är, ser jag ofta PWA när jag surfar på nätet. Men det finns ännu betydande luckor i möjligheterna att använda PWA. Beställare och programmerare behöver vara medvetna om det när de väljer vad som ska implementeras. Det är liksom ingen vits att lägga ner arbete på något folk inte kan använda och inte vill betala för.

6. RESULTAT

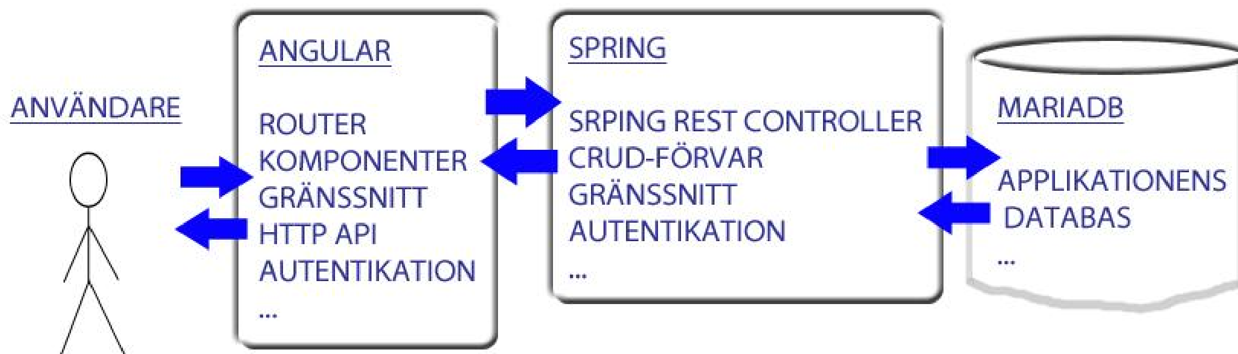
Som jag tidigare skrivit har mitt syfte varit att lära mig en ny utvecklingsmiljö, en för mig ny teknik, nämligen ramverket Angular och skapa en applikationsprototyp. Tanken har varit att fokusera på *frontend* av applikationen, men att koppla ihop den med ett separat, men parallellt utvecklat *backend*. Som en extra del har jag haft att eventuellt göra applikationsprototypen till en progressiv webbapplikation. Den delen har utgjort en möjlighet att lära om en andra ny teknik.

6.1 Grunduppdraget

Inledningsvis har läst jag på om Angular (*Angular*, n.d.-q), ordnat igång miljön på min dator och undersökt handledningar.

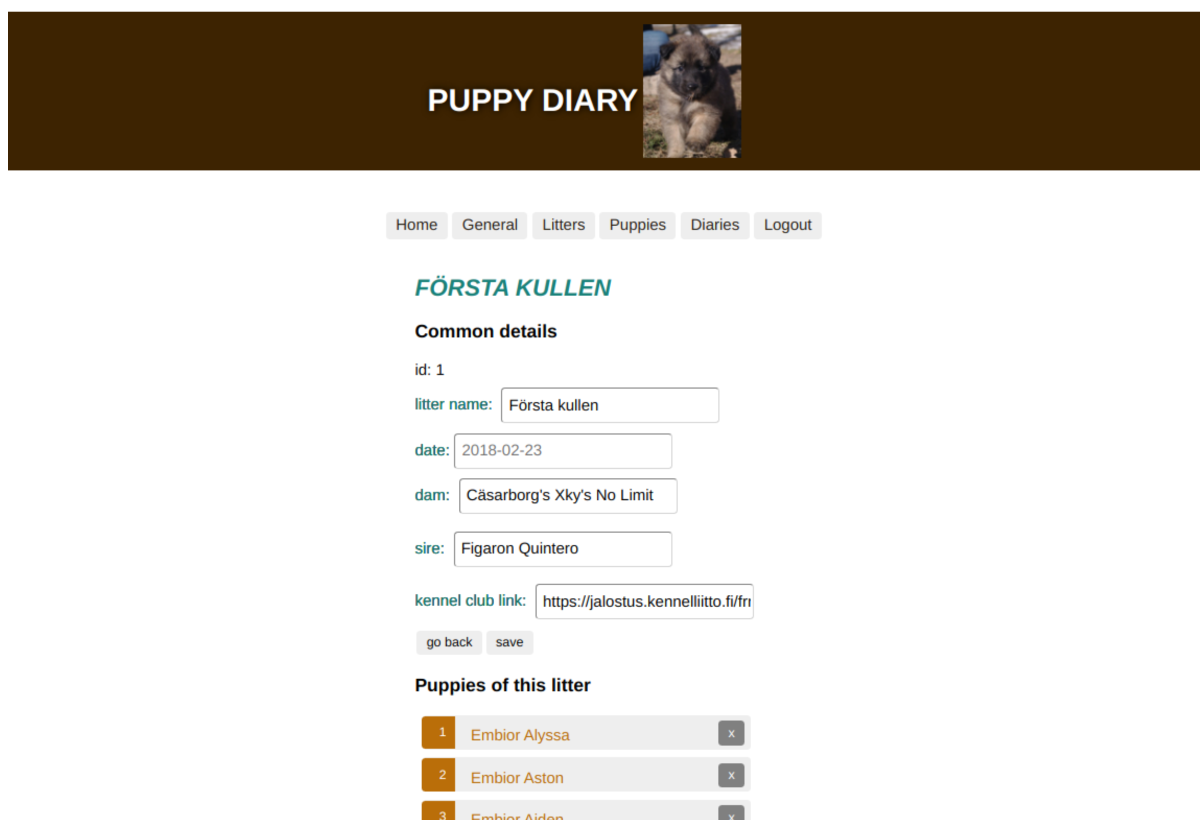
Efter att ha kommit igång lite smått med de mest grundläggande sakerna i Angular, har jag gått vidare till att bygga upp den grundläggande funktionaliteten jag tänkt min applikation ska ha. Min övningsapplikation går under namnet "Puppy Diary". Det är tänkt att bli en applikation för hunduppfödare för att bland annat spara, läsa, ändra och ta bort olika information om valpkullar och individuella valpar. I det här andra skedet har jag jobbat mot en intern databas (*angular-in-memory-web-api*).

När jag kommit så långt att *frontend* är försett med komponenter, gränssnitt och tjänster som jobbar ihop i en grundläggande helhet för *frontend*, har jag kopplat ihop mitt *frontend* med ett riktigt *backend.* Jag har byggt upp mitt *backend* i en annan kurs i skolan. Det är gjort med Springramverket och använder en MariaDB, en relationsdatabas att spara data i. Jag använder Crud-förvar (*CreateReadUpdateDelete*-förvar) för interaktion med databasen. Figur 12 visar hur mitt projekt fungerar.



Figur 12. Användaren interagerar med det frontend (Angular) visar. Frontend jobbar med ett backend (Spring), som jobbar med en databas (MariaDB).

För att koppla ihop Angular, mitt *frontend*, med mitt backend har jag använt Angulars HTTP API (*Angular*, n.d.-m). Jag har satt bas-URLen till mitt *backend* i mappen *environment* för att lätt kunna ändra på ett enda ställe. Jag har också hanterat CORS, alltså *Cross-Origin Resource Sharing* (dela resurser mellan olika ursprung) mellan *frontend* och *backend*. I figur 13 ser ni applikationen under körning när alla delar är sammankopplade.



Figur 13. Beskuret fönster från min övningsapplikation under körning.

Med *frontend* och *backend* arbetande tillsammans, har jag stannat till och förfinat min applikation en del. Att ta ett steg tillbaka och förbättra är ett återkommande tema i det här projektet, liksom i den mesta programmering jag erfarit. I nästa steg har jobbat med säkerhet. Vem får se och påverka vilket material? Jag har letat information om inloggning och läst på. Angular har flera sätt att hantera säkerhet, ett är route guards, ett annat är gränssnittet HttpInterceptor tillsammans med JWT:n (Json Web Tokens). Min applikation använder det andra alternativet (*Angular*, n.d.-m).

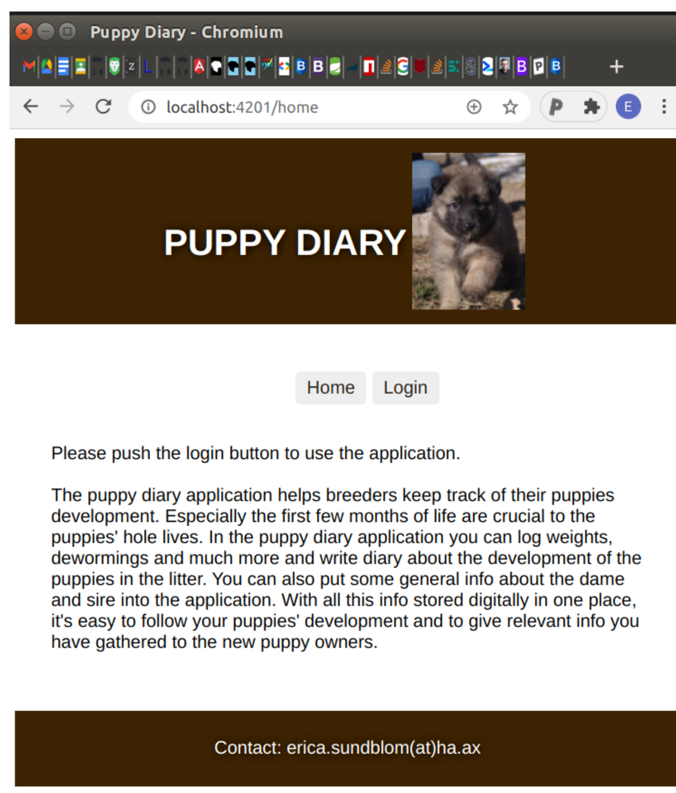
Det finns olika alternativ för hur vi kan göra inloggning. Det viktigaste finns på *backend*-sidan. *Frontend*-sidan behöver dock tillhandahålla något sätt att registrera sig som användare och inloggningsformulär och/eller någon social inloggningsfunktion. Jag har valt att både ha ett eget inloggningsformulär och att implementera inloggning med Google. För att kunna bygga en inloggning med Google har jag skapat ett konto hos Google och registrerat min applikation hos dem. Eftersom jag jobbar mot ett REST API i *backend* och valt HttpInterceptor och JWT, så behöver webbläsaren hålla reda det här JWT:t *frontend* får från *backend* vid lyckad inloggning.

I det här skedet av examensarbetet är grunduppdraget gjort. Det finns möjligheter till ytterligare utveckling, men alla projekt behöver göra delavslut och för den delen helavslut i något skede. Jag måste hinna göra examensarbetets text och jag har ännu inte undersökt PWA.

6.2 Tilläggsuppdraget

Avslutningsvis i utvecklingsdelen av mitt examensarbete har jag läst på om och undersökt PWA, samt implementerat en enkel version av PWA. Jag har utnyttjat Angulars hjälpmedel för att påbörja en PWA. Angular har hjälpt till att göra applikationen installerbar, försedd med en tjänstearbetare och ett webbmanisfest. Filen manifest.webmanifest i src-mappen är webbmanifestfilen i Angularprojekt. Ngsw-config.json i projektets root innehåller inställningar för tjänstearbetaren.

Jag har både lyckats installera och avinstallera min applikation från min dator. Notera plusset till höger i URL:en i figur 14 nedan. Pluset indikerar att applikationen går att installera.



Figur 14. Min övningsapplikation under körning.

Jag har en tjänstearbetare som sparar undan data, som används vid framtida starter och uppdaterar vartefter istället för att applikationen väntar på färskt material vid varje uppstart. Det innebär att det som först erbjuds användaren erbjuds snabbast möjligt, men kanske inte är det senaste. Det senaste kommer i sinom tid. Jag har också lagt på *cachning* av material från mitt *backend*, så vid förlorad nätverkskontakt går det fortsättningsvis att titta på det material användare har lagrat lokalt. Det skulle gå att utveckla applikationen ytterligare för att kunna jobba helt utan nätverkskontakt, alltså även göra ändringar i material och synkronisera mot applikationens *backend* vid återupprättad kontakt med internet. Men det skulle kräva att jag implementerade ett helt mellanförvar till webbläsaren, samt logik för att synkronisera det med det verkliga *backendet*. PWA utgör en tilläggsdel i examensarbetet, så eftersom tiden börjat tryta och jag har implementerat tillräckligt för att tekniskt uppfylla kraven på en PWA och visa ett smakprov på PWA-funktionalitet, så avstår jag det tills vidare. Jag har inte heller sett några uppenbara lämpliga användningsområden för *push*-notifikationer i min applikation

ännu, så jag har heller inte jobbat med *push*-notifikationer i min PWA-implementation. Om jag i framtiden lägger till funktionalitet för flera användare att dela material såsom kullar och valpar med varandra, så skulle *push*-notifikationer vara intressanta att lägga till.

En praktisk sak för programmeraren att veta är att det smidiga ng serve-kommandot som tillåter automatisk uppdatering av applikationen under programmering inte fungerar med tjänstearbetare. När programmeraren tillverkat en PWA med hjälp av Angular och vill köra den, så behöver programmeraren använda en skild HTTP-server (*Angular*, n.d.-p).

7. SLUTSATSER

Jag kan konstatera att det går långsammare att lära sig på egen hand än med strukturerad vägledning. Orsakerna till det är säkert många. Tre av orsakerna tror jag är:

1. Att det gäller att veta vad man behöver lära sig. Skolmiljöer ger vanligtvis en färdigt uttänkt struktur för vad som ska gås igenom under en kurs.
2. Att lyckas hitta relevant information. Det är lätt att hitta irrelevant, föråldrad eller rent felaktig information.
3. Att ha möjlighet att fråga någon. Det gör det lättare att snabbt uppfatta saker korrekt istället för att vara tvungen att söka ytterligare information för att förstå.

Inte desto mindre har skolan förberett mig såpass att jag klarat av att bygga ett *frontend* i Angular och få det att samspela med ett *backend* i Spring med en MariaDB bakom.

Om jag skulle fortsätta jobba med att utveckla min applikation vidare, så skulle jag försöka få några personer att pröva min applikation och be om feedback och förslag från dem. På så vis skulle jag försöka göra applikationen tilltalande för en bredare publik. Det är trots allt kunderna som i slutändan avgör om en applikation är bra eller inte. Jag skulle också ta hjälp av någon programmerare som jobbat några år för att kvalitetsgranska min applikation med andra ögon än mina egna.

Men för mina syften och för mig att använda applikationen hemma, så fungerar den utmärkt redan nu. Det är bra mycket trevligare och överskådligare att sitta och jobba med en applikation som serverar information grafiskt överskådligt än att sitta och hantera material i en databas med direkta SQL-kommandon. Och när jag vill delge någon information om en kull eller valp, så är det bara att skriva ut min samlade information.

KÄLL- OCH LITTERATURFÖRTECKNING

About npm. (n.d.). Retrieved January 13, 2021, from https://docs.npmjs.com/about-npm

Angular. (n.d.-a). CLI Command Reference. Retrieved January 15, 2021, from https://angular.io/cli

Angular. (n.d.-b). Angular Components Overview. Retrieved January 15, 2021, from https://angular.io/guide/component-overview

Angular. (n.d.-c). Component Lifecycle. Retrieved January 15, 2021, from https://angular.io/guide/lifecycle-hooks

Angular. (n.d.-d). Intro to Services and DI. Retrieved January 15, 2021, from https://angular.io/guide/architecture-services

Angular. (n.d.-e). Template Syntax. Retrieved January 15, 2021, from https://angular.io/guide/template-syntax

Angular. (n.d.-f). Built-in Directives. Retrieved January 15, 2021, from https://angular.io/guide/built-in-directives

Angular. (n.d.-g). Structural Directives. Retrieved January 15, 2021, from https://angular.io/guide/structural-directives

Angular. (n.d.-h). Building a Template-Driven Form. Retrieved January 15, 2021, from https://angular.io/guide/forms

Angular. (n.d.-i). NgForm. Retrieved January 15, 2021, from https://angular.io/api/forms/NgForm

Angular. (n.d.-j). Intro to Basic Concepts. Retrieved January 17, 2021, from https://angular.io/guide/architecture

Angular. (n.d.-k). Observables Overview. Retrieved January 13, 2021, from https://angular.io/guide/observables

Angular. (n.d.-l). AsyncPipe. Retrieved January 16, 2021, from https://angular.io/api/common/AsyncPipe

Angular. (n.d.-m). HTTP Client. Retrieved November 25, 2020, from https://angular.io/guide/http

Angular. (n.d.-n). Npm Dependencies. Retrieved January 13, 2021, from https://angular.io/guide/npm-packages

Angular. (n.d.-o). Angular Service Worker Introduction. Retrieved January 15, 2021, from https://angular.io/guide/service-worker-intro

Angular. (n.d.-p). Getting Started with Service Workers. Retrieved December 8, 2020, from https://angular.io/guide/service-worker-getting-started

Angular. (n.d.-q). Retrieved September 14, 2020, from https://angular.io/

CSS Introduction. (n.d.). Retrieved January 11, 2021, from https://www.w3schools.com/css/css_intro.asp

Documentation - TypeScript for Java/C# Programmers. (n.d.). Retrieved January 11, 2021, from https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes-oop.html

Getting Started with Progressive Web Apps. (n.d.). Retrieved January 26, 2021, from https://developers.google.com/web/updates/2015/12/getting-started-pwa

Hello World · GitHub Guides. (n.d.). Retrieved January 12, 2021, from https://guides.github.com/activities/hello-world/

How to make PWAs installable. (n.d.). Retrieved December 16, 2020, from https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Installable_PWAs

IndexedDB API. (n.d.). Retrieved January 31, 2021, from https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

Introduction to the DOM. (n.d.). Retrieved January 7, 2021, from https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

IQUII. (2019, March 4). *Progressive Web App (PWA): what they are, pros and cons and the main examples on the market*. IQUII. https://medium.com/iquii/progressive-web-app-pwa-what-they-are-pros-and-cons-and-the-main-examples-on-the-market-318f4538c670

JavaScript. (n.d.-a). Retrieved January 12, 2021, from https://techterms.com/definition/javascript

JavaScript. (n.d.-b). Retrieved January 12, 2021, from https://developer.mozilla.org/en-US/docs/Web/JavaScript

JSON. (n.d.). Retrieved January 11, 2021, from https://www.json.org/json-en.html

Node.js. (n.d.). Retrieved January 13, 2021, from https://nodejs.org/en/

Node.js Introduction. (n.d.). Retrieved January 13, 2021, from https://www.w3schools.com/nodejs/nodejs_intro.asp

npm: angular-in-memory-web-api. (n.d.). Retrieved January 11, 2021, from https://www.npmjs.com/package/angular-in-memory-web-api

Otto, M., Thornton, J., & Bootstrap contributors. (n.d.). *Introduction*. Retrieved December 16, 2020, from https://getbootstrap.com/docs/5.0/getting-started/introduction/

Poot, A. (2020, June 10). *The state of PWA support on mobile and desktop in 2020*. https://simplabs.com/blog/2020/06/10/the-state-of-pwa-support-on-mobile-and-desktop-in-2020/

Progressive Web Apps in 2021. (n.d.). Retrieved January 27, 2021, from https://firt.dev/pwa-2021/

Progressive web apps (PWAs). (n.d.). Retrieved December 8, 2020, from https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

PushPro. (n.d.). *Push Notifications for Progressive Web Apps Explained*. Retrieved January 30, 2021, from https://www.pushpro.com/blog/pwa-push-notifications-for-progressive-web-apps

SPA (Single-page application). (n.d.). Retrieved January 10, 2021, from https://developer.mozilla.org/en-US/docs/Glossary/SPA

State of Frontend 2020 Report. (2020, August 5). https://tsh.io/state-of-frontend/

The Good and the Bad of TypeScript. (2020, February 14). https://www.altexsoft.com/blog/typescript-pros-and-cons/

Typed JavaScript at Any Scale. (n.d.). Retrieved October 20, 2020, from https://www.typescriptlang.org/

Visual Studio Code - Code Editing. Redefined. (2016, April 14). https://code.visualstudio.com/

Web Storage API. (n.d.). Retrieved January 12, 2021, from

https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

What are Progressive Web Apps? (n.d.). Retrieved January 14, 2021, from https://web.dev/what-are-pwas/

What is HTML. (n.d.). Retrieved January 11, 2021, from https://www.w3schools.com/whatis/whatis_html.asp

What is Object Oriented Programming? OOP Explained in Depth. (n.d.). Retrieved January 11, 2021, from https://www.educative.io/blog/object-oriented-programming

Window. (n.d.). Retrieved February 16, 2021, from https://developer.mozilla.org/en-US/docs/Web/API/Window

Window.sessionStorage. (n.d.). Retrieved January 12, 2021, from https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage