Examensarbete, Högskolan på Åland, Utbildningsprogrammet för Informationsteknik

UTVECKLING AV ANVÄNDARGRÄNSSNITT TILL WEBBPORTAL

för Crosskey Banking Solutions

Kim Lindholm

2021:21

Datum för godkännande: 21.06.2021
Handledare: Björn-Erik Zetterman

EXAMENSARBETE
Högskolan på Åland

Utbildningsprogram:	Informationsteknik
Författare:	Kim Lindholm
Arbetets namn:	Utveckling av användargränssnitt till webbportal - för Crosskey Banking Solutions
Handledare:	Björn-Erik Zetterman
Uppdragsgivare:	Crosskey Banking Solutions

Abstrakt

Syftet med den här uppdraget är att skapa ett användargränssnitt till webbportalen crosskey.io, för att underlätta för applikationsutvecklare och banker att ha bättre överblick och lätt hitta rätt API, samt att förklara vad Open Banking är och vilka krav som ställs samt vilka EU-direktiv som gäller för ändamålet.

Nyckelord (sökord)

Angular, webbportal, användargränssnitt, frontend, crosskey.io

Högskolans serienummer:	ISSN:	Språk:	Sidantal:
2021:21	1458-1531	Svenska	33 sidor

Inlämningsdatum:	Presentationsdatum:	Datum för godkännande:
20.06.2021	12.05.2021	21.06.2021

DEGREE THESIS
Åland University of Applied Sciences

Study program:	Information Technologies
Author:	Kim Lindholm
Title:	Development of User Interface for Web Portal - for Crosskey Banking Solutions
Academic Supervisor:	Björn-Erik Zetterman
Technical Supervisor:	Crosskey Banking Solutions

Abstract

The purpose of this assignment is to create a user interface for the web portal crosskey.io, to make it easier for application developers and banks to have a better overview and easily find the right API, and to explain what is Open Banking and what requirements are set and what EU directives apply to the purpose.

Keywords

Angular, webportal, user interface, frontend, crosskey.io

Serial number:	ISSN:	Language:	Number of pages:
2021:21	1458-1531	Swedish	33 pages

Handed in:	Date of presentation:	Approved on:
20.06.2021	12.05.2021	21.06.2021

INNEHÅLLSFÖRTECKNING

1. INLEDNING **4**
1.1 Syfte 4
1.2 Metod 4
2. DEFINITIONER **5**
2.1 Crosskey.io 5
2.2 Open Banking 5
2.3 Bakgrund till PSD2 7
2.4 Säkerhetsaspekt 9
2.5 Angular 10
2.5.1 Skillnad mellan JavaScript och TypeScript 10
2.5.2 Skillnad mellan Angular och AngularJS 11
2.5.3 Dokumentmodell i AngularJS 12
2.5.4 Node.js och NPM 12
2.5.5 Komponent vs Modul 13
2.5.6 Pipes 14
3. KRAV **19**
3.1 Skallkrav 19
3.2 Börkrav 19
3.3 Design 20
4 IMPLEMENTATION **21**
4.1.1 Visa alla API:er 21
4.1.2 Sök-implementation 24
4.1.3 Responsiv sida 25
4.1.4 Hur API:er skall representeras (box eller lista) 26
5. SLUTSATS **30**
5.1 Framtida utveckling 30
KÄLL- OCH LITTERATURFÖRTECKNING **31**

1. INLEDNING

1.1 Syfte

Syftet med det här uppdraget är att skapa ett användargränssnitt till webbportalen crosskey.io. Webbportalen används av olika användare som typiskt är utvecklare av andra applikationer. Portalen används som informationskälla för olika integrationer för att hitta tekniska specifikationer och syftet med webbportalen är att tillgängliggöra information från en informationskälla som uppfylls enligt *Open Banking UK standard*.

Från ett användarperspektiv skall det vara lätt att hitta den eftersökta informationen. För att förenkla det skall det finnas sökfunktioner för att underlätta den processen, så att användaren ej behöver läsa all information och själv söka i text utan använda en sökfunktion för ändamålet.

Projektet strävar även till att förklara vad *Open Banking* är för något, vilka krav som ställs samt vilka EU-direktiv som gäller för ändamålet.

1.2 Metod

Projektet började med att jag hjälpte ett annat team inom Crosskey med att eliminera buggar inom användargränssnittet, vilket ledde till att jag hittade ett koncept att hjälpa utvecklarna med att hitta rätt applikationsprogrammeringsgränssnitt (API). Så jag talade med min handledare om idén att utveckla en till sida till Crosskey.io, där sidan skulle visa alla API:er som erbjuds. Detta skulle göra det enklare för både utvecklare och banker att hitta rätt API samt ha bättre överblick vad som erbjuds.

Metoden som jag använder för att utveckla användargränssnitt, är webbramverket Angular och utvecklingsmiljön är Intellij.

2. DEFINITIONER

2.1 Crosskey.io

Crosskey Banking Solutions Ab är ett åländskt företag inom finansiell IT och är dotterbolag till Ålandsbanken. Crosskey Banking Solutions Ab levererar IT-tjänster till banker och andra finansiella aktörer. För att lättare kommunicera olika metoder för integration har Crosskey Banking Solution skapat serviceplattform *Crosskey.io*.

Crosskey.io är en molnbaserad öppen webbplattform som används av utvecklare för att integrera med Crosskeys kunders data och uppfyller europeiska bankmyndighetens (*European Banking Authority*, EBA) krav för att reglera tekniska standarder (RTS, *Regulatory Technical Standards*).

Crosskey.io erbjuder API-marknad som då länkar banker och tredje parter med data, kanaler och funktioner. Det skall vara enkelt och effektivt att integrera samt minska komplexitet och sänka kostnader för alla parter (Crosskey, 2021).

2.2 Open Banking

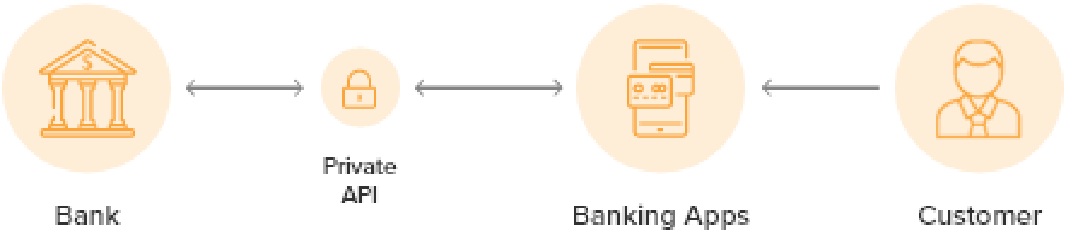
Open Banking är en process inom finansindustrin som har utformat för att ge mer konkurrens och innovation till finansiella tjänster som erbjuds av olika leverantörer som använder Open Banking för att kunna erbjuda sina produkter och tjänster. Detta förfarande regleras av *Financial Conduct Authority* (FCA) i England eller motsvarande inom EU (TW, 2021).

Figur 1 beskriver en modell utan Open Bank-integration, där banken som är kontohållare ej delger information till tredje part. Det skapar en monopolsituation som löses genom *Open Banking*. På grund av *Open Banking* har olika aktörer rätt att ta del av information, som de har behörighet till, såsom kontotransaktioner, kontoinformation, tillgängliga medel, initiera betalningar osv.

För varje bank eller finansiell aktör krävs skilda API då de hanteras olika och har olika beställare. Det ger upphov till en stor mängd API:er som skall vara sökbara och tillgängliga. Med det privata API och bankapplikationen (se figur 1 utan *Open Banking*) begränsas kunden att endast använda applikationen till den bank som har byggt upp hela systemet och för banken del så är det en dyr process att skapa systemet. En fördel med att ha *Open Banking* är att en applikation kan använda data från flera banker, eller en bank kan användas från flera applikationer.

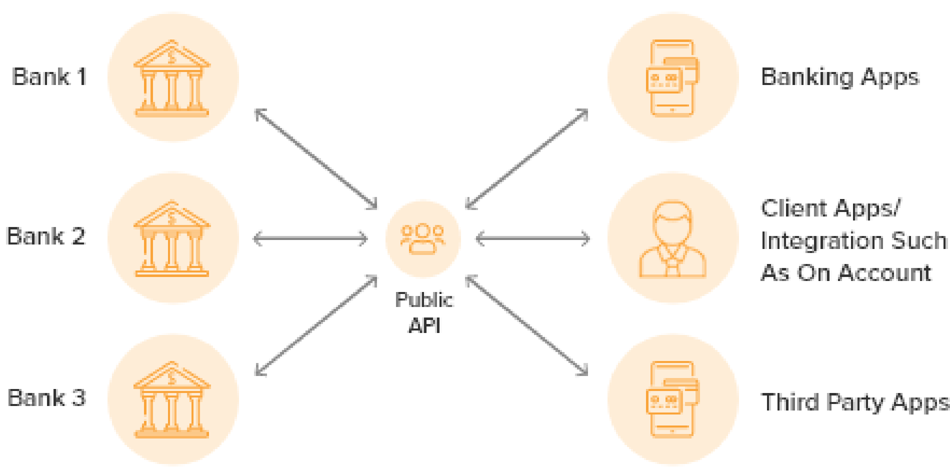
API MODEL

WITHOUT OPEN BANKING



Figur 1, modell av bankens egen API och applikationsgränssnitt (Hathaway, 2020).

I Figur 2 visas principen för Open Banking, där en bank offentliggjort sina API:er enligt *Open Banking*. Då kan utvecklaren använda dessa API:er genom att konsumera data via API till sina tredjepartsapplikationer som använder data från bankens gränssnitt. Detta leder till att utvecklaren kan koppla sig till flera banker och integrera data samt att andra affärer och finansierings-applikationer kan nyttja dessa API. Från bankkunders synpunkt har kunden bättre översikt om sin finansiella information och kan effektivare att hantera sina ärende hos banker (Hathaway, 2020).



Figur 2, om en bank offentligt gör sina API:er enligt Open Banking (Hathaway, 2020)

Man kan definiera Open Banking som en modell där bankuppgifter delas via ett API mellan banker och icke-banker. Open Banking främjar innovationer och det ger nya affärsmöjligheter och användningsmöjligheter då konkurrens råder mellan banker och icke banker.

Men det ställs också frågor om dataskydd och reglering kring hur mycket av data som skall delas via API:er. Inom EU har EU kommissionen fastställt regler för Open Banking-engagemanget genom att förnya den tidigare betalningstjänstdirektivet (PSD2, *Payment Service Directive 2*) (Brodsky & Oakes, 2017).

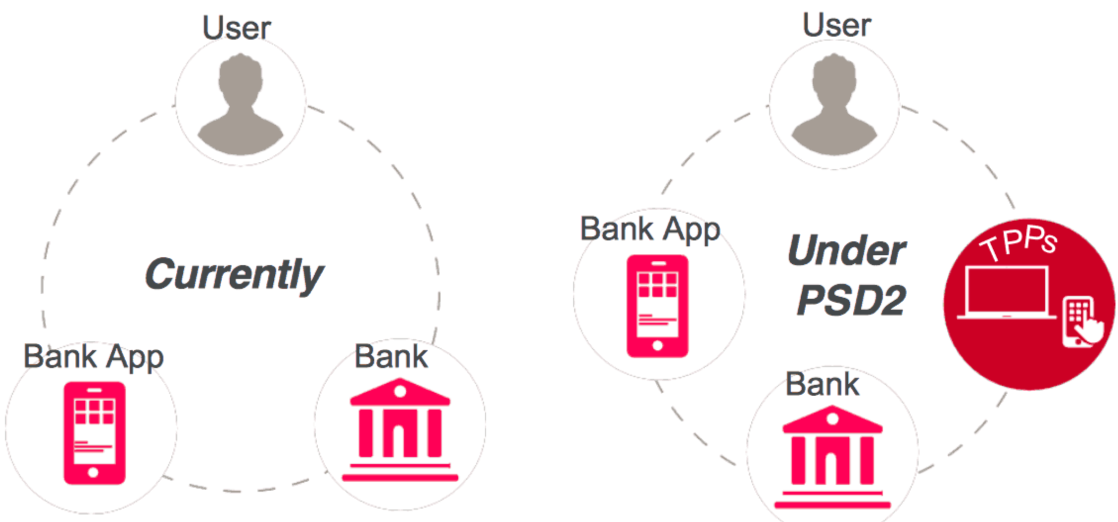
2.3 Bakgrund till PSD2

PSD2 (*Payment Service Directive 2*) är ett reviderat direktiv om betaltjänster som ersatte det tidigare direktivet PSD (*Payment Service Directive*) med utökat behov.

PSD är en reglering av marknaden som Europeiska unionen (EU) tog fram år 2007, för att bidra till en gemensam betalningsmarknad och för att främja innovation, konkurrens och effektivitet inom EU. Europeiska kommissionen föreslog förändring av PSD år 2013, med syfte till att förbättra dess mål och förbättra konsumentskyddet, öka konkurrensen, affärsidéer och tjänster i sektorn samt förstärka säkerheten på betalningsmarknaden. Förändringar som gjordes i PSD2, var att bankerna öppnade sina betalningstjänster till andra företag, även kallad tredje-partsbetalningstjänsteleverantör (TPP, *Third Party Provider*) och två typer av tjänster som har blivit populära de senaste åren är betalningsinitieringstjänst (PIS, *Payment Initiation Service*) och Konto-informationstjänster (AIS, *Account Information Service*), vilket PSD2 (*Payment Service Directive 2*) både reglerar och harmoniserar. Dessa två tjänster fanns redan i PSD som togs i bruk år 2007. Konto-informationstjänster (AIS, *Account Information Service*) samlar in och lagrar kundens olika bankkontoinformation på ett enda ställe, så att det underlättar för kunden att få en helhetsbild över sin ekonomiska situation samt underlätta analysering av utgifter och ekonomiska behov. Tjänst för initiering av betalning (PIS, *Payment Initiation Service*) används av nätbanken för att utföra betalningar online. Dessa två tjänster, AIS och PIS utgör ett betalningsgränssnitt för att överbrygga en betalning, genom att fylla in den information som behövs för en banköverföring vilket är transaktionsbelopp, kontonummer, meddelande och informera mottagaren om transaktionen (BBVA, 2019).

PSD (*Payment Service Directive*) gäller för alla länder inom EU (Europeiska unionen) och de länder som är inom europeiska ekonomiska samarbetsområdet (EES, *European Economic Area*, EEA).

Sammanfattningsvis kommer PSD2 och *Open Banking* att leda till att tredjeparts leverantörer kan skapa tjänster som fungerar med den information som i realtid tillhandahålls av banken. Det leder till att den leverantör av produkt (en bankapplikation) som kunden väljer kan tillhandahålla tjänster i samtliga banker, vilket visas i figur 3) (Global Leading Conferences, 2017).



Figur 3. Vänster sida beskrivs hur det ser ut idag och högra sidan hur det kommer att se ut när bankerna inför PSD2-direktivet (Global Leading Conferences, 2017).

2.4 Säkerhetsaspekt

När bankerna skall följa PSD2 med att öppna sina kunddata via API:er för transaktionskonton, sparkonto och kreditkortsuppgifter, krävs det från bankens sida att de förstärker sin riskhantering för att kunddata skall hanteras på säkert sätt (Sowden, 2019).

Man kan förvänta sig att nya metoder för nätfiskebedrägeri kommer att dyka upp, när hackers låtsas att vara ett FinTech-företag. Från hackers perspektiv så betraktas bankkunders transaktionsdata som högvärdig information och med denna information kan hackers studera beteendemönster, rutiner, scheman och ekonomisk status över bankkunden. Det innebär att hackers kommer att ha stort intresse för *Open Banking* (Trend Micro Incorporated, 2019).

Så fördelen med *Open Banking* är att förbättra kundupplevelse, nya intäktsströmmar och hållbar servicemodell. Ur bankens perspektiv blir det lägre kostnader att skapa och underhålla sina applikationer. *Open Banking* ökar marknadsmöjligheter för bankerna och leverera lönsamma tjänster (Brodsky & Oakes, 2017).

2.5 Angular

Angular är ett *TypeScript*-baserat webbramverk och leds av Googles Angular Team, som består av ingenjörer som delar en passion att göra webbutveckling enklare (Angular, 2021a). Angular har öppen källkod, så det är tillåtet att modifiera källkoden. Angular kallas också Angular 2+ eller Angular v2 och uppåt (Wikipedia, 2021a).

2.5.1 Skillnad mellan JavaScript och TypeScript

JavaScript och *TypeScript* är lika varann, men *TypeScript* erbjuder samma som *JavaScript* men också extra lager på *JavaScript*.

JavaScript tillhandahåller vanliga språkprimitiver som *string* (sträng), *number* (nummer) och objekt. *TypeScript* kontrollerar att data har tilldelats på rätt sätt, men det gör ej *JavaScript* (Typescript, 2021).

JavaScript är programmeringsspråk på klient-sidan men kan också användas på serversidan. När *JavaScript*-koden på serversidan växte och blev mera komplex blev det för tungt för en server att kunna köra *JavaScript* koden och ge snabba respons till klienten, vilket ledde till att *JavaScript* ej lyckades komma in på företagsnivå som en *serverside*-teknik och kunde ej uppfylla kravet på ett objektorienterat programmeringsspråk.

TypeScript är ett objektorienterat programmeringsspråk och överlappar glappet mellan server och klient-kommunikationen. *TypeScript* är ej specifikt för någon virtuell maskin, vilket innebär att *TypeScript*-koden kan köras på vilken webbläsare, enhet eller operativsystem, men för att webbläsaren skall förstå *TypeScript* så måste *TypeScript* kompileras och omvandlas till *JavaScript*.

Denna process är känd som *Trans-piled*. Så fördelen med *TypeScript* är att den stöder moduler, klasser, gränssnitt och *JavaScript*-bibliotek. Nackdelen med *TypeScript* ifall koden blir för komplex och stor, är att det kommer att ta tid, ifall det tar mera än 1 - 2 minuter för koden att kompilera och omvandlas till *JavaScript*, därefter så förstår webbläsaren och utför instruktionerna som har kodats i *TypeScript* (Dubey, 2018).

I figur 4 ses skillnaden mellan *JavaScript* och *TypeScript*. Om man skall addera två tal med TypeScript så kommer TypeScript kompilator att ge ifrån sig felmeddelande på rad 3 och 4, för att 1: orna är innanför apostrof och därefter tolka TypeScript som sträng. *JavaScript* så slår ihop 1: orna som är innanför apostrofer och visar 11 som resultat (Michels, 2018).

typescript.ts

```
const add = (x: number, y: number): number => x + y;

add('1', '1'); // compiler error
add(1, '1'); // compiler error
add(1, 1); // 2

// compiler error IF "strictNullChecks": true,
add(null, undefined);
```

javascript.js

```
const add = (x, y) => x + y;

add('1', '1'); // 11
add(1, '1'); // 11
add(1, 1); // 2


add(null, undefined);
```

Figur 4, skillnad mellan TypeScript och JavaScript med enkel addition (Michels, 2018).

2.5.2 Skillnad mellan Angular och AngularJS

AngularJS är föregångare och första versionen av Angular. AngularJS är ett *JavaScript*-baserat webbramverk med öppen källkod och underhålls för det mesta dels av Google och sen av grupper av individer och företag (Wikipedia, 2021b).

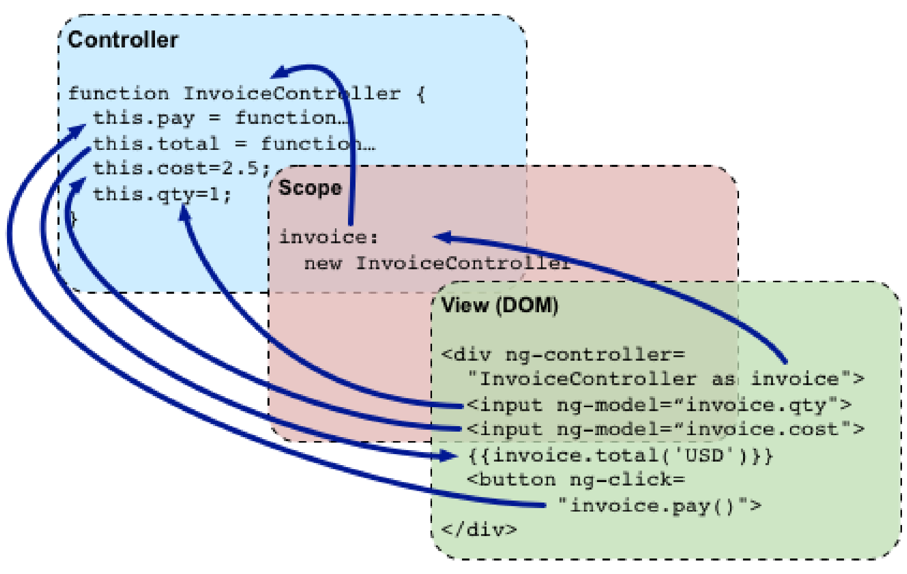
AngularJS använder sig av "*scope*" för att binda sig mellan vy och kontroller. *Scope* är ett objekt som hänvisar till applikationsmodellen och är ordnade i hierarkisk struktur som efterliknar DOM-strukturen i applikationen. Det finns alltid en *root scope* men flera underliggande *scope*. Kontroller (*Controllers*) är en *JavaScript*-konstruktor funktion och använder *scope* som argument. Hur *scope* och kontroller kommunicerar mellan varann, beskrivs mera i kapitel 2.5.4 (AngularJS, 2020).

Angular som är nyare och har vissa fördelar som ej AngularJS har. Angular har ej "*Scope*" eller "*Controllers*", utan Angular använder sig av (Wikipedia, 2021):

1. Hierarki av komponenter.
2. Annan syntax då det är TypeScript(se kap 2.5.2), ej *JavaScript*.
3. Modularitet, dvs. att alla stora funktioner är indelade i mindre moduler

2.5.3 Dokumentmodell i AngularJS

AngularJS använder sig av det traditionell HTML- och DOM-manipulering. Figur 5 förklarar hur “*scope*” fungerar mellan kontroller och vyn. I DOM finns det *ng-controller*, som berättar åt AngularJS att *InvoiceController* är ansvarig för elementet och elementets barn. *Invoice* i “*InvoiceController as invoice*” är självaste variabeln i “*scope*”:et som sedan används på de platser som behöver använda *InvoiceController*s variabler och funktioner. Denna *data-bindning* är *live*. För varje ändring som sker i *invcoice.qty* eller i *invoice.cost* skickas det data till *InvoiceController* för att uppdatera variabelnamnet med det nya värdet samt rendera ut det nya värdet med *ng-repeat* eller när användare klickar på knappen varvid funktionen löses ut som är kopplad till "*invoice.pay()*” (AngularJS, 2020).



Figur 5. Hur “scope” i AngularJS fungerar (AngularJS, 2020).

2.5.4 Node.js och NPM

I kapitel 2.5.2, nämndes att *JavaScript* ej lyckades komma in på företagsnivå som en serverside-teknik. Node.js blev däremot accepterat på föreetagsnivå ochblev introducerad år 2009 med sin första version som kunde skriva och exekvera *JavaScript* på serversidan. Node.js är öppen källkod, *cross*-plattform, mjukvaruutveckling som kör *JavaScript*-kod utanför webbläsare. Utvecklarna kan skriva *JavaScript*-kod på serversidan och skapa en dynamisk webbsida (Wikipedia, 2021c).

Node.js är utformad för att bygga skalbara nätverksapplikationer och använder sig av asynkron händelsestyrd *JavaScript-runtime*. Det innebär att flera anslutningar hanteras samtidigt och vid varje anslutning avbryts återuppringningar. Om Node.js ej har några arbeten att utföra, går Node.js i "sovläge". Node.js har nästan inga I/O-funktioner som kan blockera en process eller låsa processen. Många Node.js-bibliotek och applikationer är publicerade på NPM och fler läggs till (node.js, 2021).

NPM (*Node Package Manager*) är både *online* förvar och en kommandorad för att hjälpa med att installera paket av programvara/bibliotek som även sköter versionshantering och beroendehantering. NPM är standard pakethanterare för användning av Node.js (Node.js, 2021).

2.5.5 Komponent vs Modul

Komponent är en grundstomme i applikationen Angular och varje komponent har följande innehåll:

- HTML-mall, som renderas på sidan (mall)
- *TypeScript*-klass, hur allting skall fungera
- En CSS-*selector*, som definierar hur komponent skall visas i mallen.

Komponenter definierar vyer och komponenter använder sig av tjänster som ger specifika funktioner och är ej direkt relaterade till vyn.

Angular har ett eget modularitets-system som heter *NgModules* och varje Angularapplikation har en *root modul som* heter *AppModule*.

I *NgModules* innehåller

- Komponenter
- Tjänster som *JavaScript*-bibliotek
- Andra koder, som andra *NgModules*.

Moduler, komponenter och tjänster är själva klasser och använder sig av dekoratörer.
Dessa dekoratörer berättar åt Angular vad det är för någonting och hur det skall användas.

Komponent använder sig av @*Component*-dekoratör som identifikation, medan *Pipe* använder sig av @*Pipe*-dekoratör som identifikation (Angular, 2021).

2.5.6 Pipes

Angular har färdiga inbyggda *pipes* (rör) och dessa *pipes* så kan man omvandla strängar, valutabelopp och annan data som skall visas. Man kan använda *pipes* i hela applikationen och man behöver bara deklarera *pipe* en gång. Man kan skapa egna *pipes* som kan utföra andra saker som ex. söka på bokstäver eller andra saker som datum, år m.m. (Angular, 2021)

Vissa *pipes* har valfria parametrar för att kunna finjustera data. Ifall *pipes* accepterar flera parametrar så separeras man med kolon (**:**). För att kunna använda pipe i sin applikation, så behöver man använda lodstreck (|) som även kallas till *pipe*-operatör och man kan kedja ihop med flera olika pipes efter varann med att separera med *pipe*-operatör (|) (Angular, 2021d).

Om man implementerar ett eget filter, så skall man först skapa en *TypeScript*-fil där namnkonventionen för *pipe* blir: namn.*pipe*.ts detta för att veta vad den gör och vad det är för nånting. Man kan importera på två sätt.

1. Enskilda funktioner, som *Pipe* och *PipeTransform* från Angular/*core*
2. använder en stjärna (*) för att importera allt från en modul eller från *JavaScript* bibliotekMan använder stjärna (*) för att importera allt från en modul eller från *JavaScript*-bibliotek.

I figur 6 har jag skapat en fil för *pipe*-filter som skall filtrera nästlade JSON-matriser och importerar två funktioner (*Pipe* och *PipeTransform*) från Angular/core och alla funktioner från lodash med namngivning understreck (**_**). @*Pipe*-dekoration, med namngivning filter som sen används i de HTML-filer som behöver använda detta filter tjänsten.

```
filter.pipe.ts
1  import { Pipe, PipeTransform} from '@angular/core';
2  import * as _ from 'lodash';
3
4
5  @Pipe({
6    name: 'filter'
7  })
8
```

Figur 6, importering av Pipe, PipeTransform, lodash och namngivning.

Lodash, är *JavaScript*-bibliotek och hjälper utvecklaren att ta bort en massa onödigt arbete när utvecklaren skall hantera matriser (*arrays*), nummer, objekteter och strängar (John-David Dalton, 2012). JSON (*JavaScript Object Notation*) är populär dataformat med att skicka API-förfrågningar och svar mellan klienten och servern. JSON saknar ett standardiserat schema eller metadata-definition om hur en JSON-dokument skall struktureras, vilket betyder att det är upp till utvecklaren själva att specificera JSON-dokument strukturen. JSON:s datatyper så är baserat på *JavaScript* programmeringsspråk (Felipe Pezoa, Domagoj Vrgoč, Martín Ugarte, Juan L. Reutter, & Fernando Suarez, 2016).

På en kapslad JSON-matris i figur 7 ser man att API som har flera JSON-matriser är kopplade till rätt ID-nummer. Det som är ovanför API är nivå 1 och de som är innanför API är nivå 2.

```
// Nested JSON-array
[
  {
    name: 'A',
    id: 1,
    api[
        {
          id: 1,
          desciption: 'A is best 1.0',
          title: 'A is awesome 1.0'
        },
        {
          id: 1,
          desciption: 'A is best 1.1',
          title: 'A is awesome 1.1'
        },
        .
        .
        .
       ]
  },
  .
  .
  .
]
```

Figur 7, hur en JSON-Array ser ut i ren dataform

I figur 8 deklareras klassen *FilterPipe* (kommer från filnamngivning) och implementerar *PipeTransform*-gränssnitt, för att utföra ändringarna. Angular anropar metoden *transform* för att kunna binda första argumentet som då kommer från HTML filen och alla parametrar därefter, som kan vara ex. sökord. Sedan returneras det omvandlade värdet. *Value* (värdet) är min JSON-matris och *filterString* är mitt sökord, som jag kommer att beskriva lite senare mera om. I det första *if*-påstående så kollar jag bara upp om *filterString* är tom, returnerar vi *value* såsom det är.

```
export class FilterPipe implements PipeTransform {
  transform(value: any[], filterString: string): any {
    if (filterString === '') {
      return value;
    }
```

Figur 8, klassdeklarering, implementation och metodanrop.

Ifall *filterString* ej är tom, så skapar jag en variabel *resultArray* som skall spara matris-värdet och sedan returnera det. Det första som görs, är att använda *filter*-metod för att kolla upp ifall värdet innehåller (*includes*) det som är efterfrågas och returnerar den filtrerade JSON-matrisen, som sen sparas i *resultArray*. Jag använder *toLowerCase* funktion för att ej bli beroende ifall namnet eller sökordet (*filterString*) är skrivet med stora bokstäver eller med småa bokstäver, så gör funktionen *toLowercase* allting till små bokstäver, se figur 9.

```
let resultArray = [];
resultArray = _.filter(value, item => {
  return item.name.toLowerCase().includes(filterString.toLowerCase());
});
```

Figur 9, första nivå att filtrera

I Figur 10 ses hur ifall den första funktionen ej gav resultat observerar jag om *resultArray* har längden 0. Anropas nästa filterfunktion, men istället itererar funktionen djupare in i JSON-matrisen och använder *find*-metoden för att söka upp namnet på API:er och returnerar JSON-matrisen till *resultArray*.

```
if (resultArray.length === 0) {
    resultArray = _.filter(value, item => {
      return _.find(item.api, apiName => {
        return apiName.name.toLowerCase().includes(filterString.toLowerCase());
      });
    });
}
```

Figur 10, djup filtrering av JSON-Array för att söka API-namn

I figur 11 ses hur ifall den översta funktionen ej gav något resultat, så undersöker jag igen om *resultArray* har längden 0, och utför samma process som i figur 10 (se figur 10). Istället för att söka upp på API-namnet så går den på API-beskrivning och returnerar JSON-matrisen till *resultArray*.

```
if (resultArray.length === 0) {
  resultArray = _.filter(value, item => {
    return _.find(item.api, apiDescription => {
      return apiDescription.description.toLowerCase().includes(filterString.toLowerCase());
    });
  });
}
```

Figur 11, fortsättning av djupfiltrering men söker på API-beskrivning

I figur 12 returnerar *resultArray* till HTML med nya JSON-matris eller tom *resultArray*.

```
  return  resultArray;
}
```

Figur 12, returnering av resultArray

3. KRAV

När jag diskuterade med min handledare om olika krav som skall implementeras för att utveckla sidans gränssnitt på crosskey.io webbportal, så listade vi upp följande som prioritering.

3.1 Skallkrav

Följande krav ska uppfyllas till den nivån att primära funktionaliteten kan användas på sidan (Pabliq, 2021b).

1. Visa en lista av API:er
2. Man skall kunna söka upp API:er via:
    a. Sökruta
    b. Checkboxar, valmöjligheter att filtrera på olika bankaktörer (Flervalsboxar)
    c. Miljö (sandbox eller live)
3. Sidan skall vara responsiv, man skall kunna använda följande:
    a. Mobila enheter, såsom *tablets* och *smartphones*
    b. Desktop, beroende på skärmupplösning
4. Användaren skall kunna välja utseende på resultatvisning (lista eller box)

3.2 Börkrav

Följande krav är ej obligatorisk att uppfylla under utvecklingsinkrement, dock ger det mervärde till gränssnitt sidan om man uppfyller börkraven (Pabliq, 2021a).

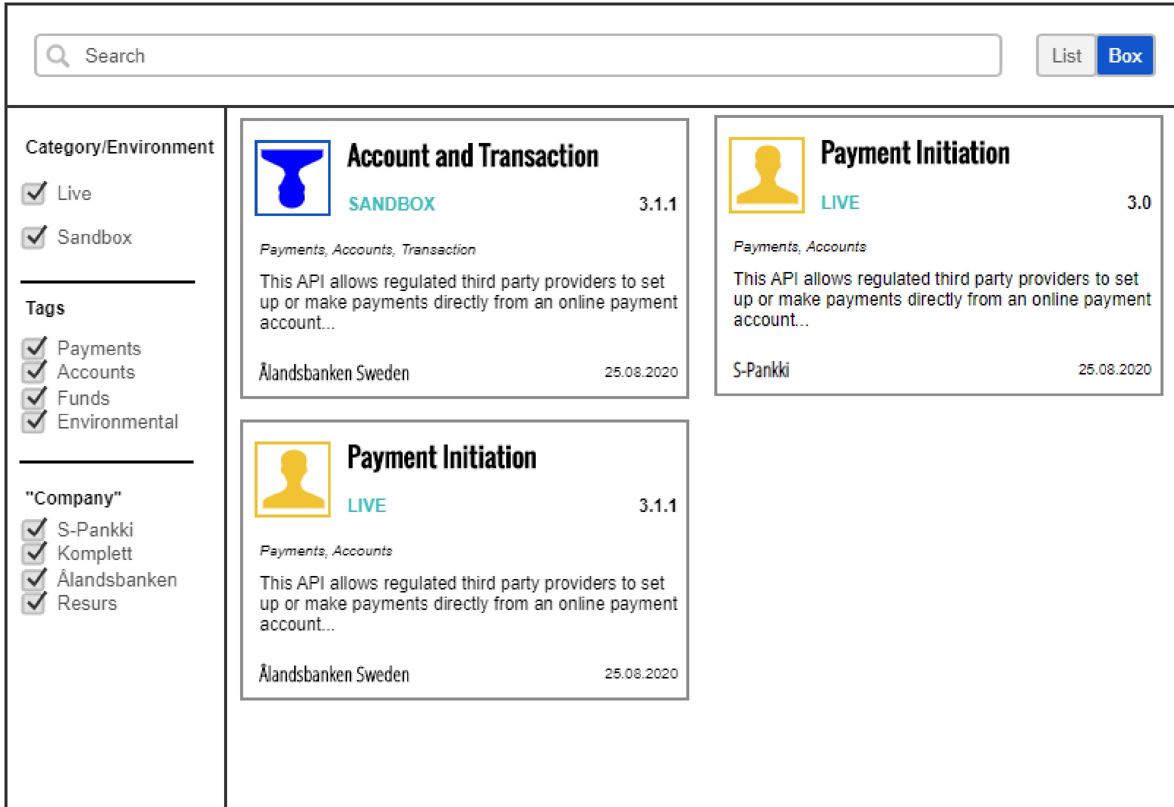
1. Uppfylla tillgänglighetsdirektivet[1]
2. Filtrera sökning på API-taggar (*Payment*, *Account*)

[1] Läs mer om tillgänglighetsdirektivet på Valtiovarainministeriö The Ministry of Finance. (2021). Tillgänglighet. Retrieved from https://vm.fi/sv/tillganglighetsdirektivet

3.3 Design

När jag och handledaren diskuterade om kraven (se kap. 3) så tog vi upp hur användargränssnittet skulle se ut på basen av kravspecifikation.

I figur 13 visas hur layouten är tänkt att se ut i klassisk desktop miljö. Högst upp kan man se en sökruta, följt av en väljare för box eller listvisning. Till vänster visas en filtrering av resultat på olika fördefinierade kategorier för att göra urval/begränsning av sökning. Till höger om den filtreringen finns resultatet, som i detta fall visas som en typisk box-stil.



Figur 13, Konceptdesign hur användargränssnittet skall se ut, när det är färdigt.

4 IMPLEMENTATION

I detta kapitel tar jag upp hur jag har implementerat en del av kravspecifikationen (se kap. 3), och så beskriver jag hur allting är uppbyggt i HTML DOM, hur funktionaliteten ser ut och hur slutresultatet blir efter jag har implementerat lösning.

4.1.1 Visa alla API:er

I HTML-filen (se figur 14) använder jag *ng-container* för att bibehålla HTML-strukturen när den går igenom JSON-matrisen. Den första *ngFor* itererar genom *collectStore*, som då har JSON-matriser och den andra *ngFor* itererar genom andra nivån av JSON-matriser (se figur 7). *NgClass.lt-lg* beskrivs mera i kapitel 4.1.3. *Mat-card* är Angulars färdig-implementerade material som jag använder, vilket får att API:er se ut som kort. *NgIf* undersöker om det finns en sträng i *apiData.logoResource*. Om det finns kommer bilden att renderas ut annars hoppar den över.

I *mat-card-title* och *mat-card-subtitle* skrivs det ut vad API heter och vilken miljö det är. *Mat-card-content* har API beskrivning och *mat-card-actions* berättar vilken bank som äger API: et.

```
<ng-container *ngFor="let data of collectStore | filter: searchWord">
  <ng-container *ngFor="let apiData of data.api ">
    <a ngClass.lt-lg="list"
       [ngClass]="activeButton ? 'list': 'box'">
      <mat-card>
        <mat-card-header>
          <div mat-card-avatar *ngIf="apiData.logoResource">
            <img src="{{apiData.logoResource}}" alt="">
          </div>
          <mat-card-title>{{apiData.name}}</mat-card-title>
          <mat-card-subtitle>{{apiData.environment}}</mat-card-subtitle>
        </mat-card-header>
        <mat-card-content>
          <p>
            {{apiData.description}}
          </p>
        </mat-card-content>
        <mat-card-actions>
          {{ data.name }}
        </mat-card-actions>
      </mat-card>
    </a>
  </ng-container>
</ng-container>
```

Figur 14, HTML-implementation av hur alla API:er visas.

I figur 15 initialiseras *collectStore* matris variabel som sparar JSON-matriser, som kommer från serverns svar på API-anrop baserat på de kriterier för sökning som angetts.

```
collectStore: [] = [];
```

Figur 15, collectStore initialiserat

I figur 16 visas när sidan börjar laddas in så. Då anropas funktionen *initCollection()* som då kör *getStoreData()*-funktionen och sedan funktion *getApiData*(*this.collectStore*). I *getApiData* används *collectStore* som parameter för att hämta de sista API från databasen.

```
async ngOnInit() {
  await this.initCollection();
}

private async initCollection() {
  this.collectStore = await this.getStoreData();
  await this.getApiData(this.collectStore);
}
```

Figur 16, Hur funktionen anropas när sidan börjar laddas

Async *pipe* observerar på en funktion och när senaste värdet som har angivits, kommer den att returnera värdet. När komponenten som har en async *pipe* förstörs kommer Async *pipe* att avslutas per automatik för att undvika minnesläckor (Angular, 2021b).

I föregående figur (se figur 16) såg vi hur funktioner anropas när sidan börjar laddas och i figur 17 går vi igenom hur allting sitter ihop så att i figur 14 kan *collectStore* iterera och rendera ut API-data. I *getStoreData*-funktionen har *marketplaceService*-tjänst som i sin tur har *getStores()*-funktion som då anropas till databasen för att hämta ut den första biten av JSON-matrisen. I *getApiData*-funktionen som har *collectStore* som parameter, itererar jag

igenom *collectStore*, för att sedan anropa databasen med *getApis*-funktion med id som parameter för att hämta resten av API:erna och därefter anropas *mergeData*-funktionen som tar nuvarande JSON-matris och dess API:er. I *mergeData*-funktionen sparar *collectStore.api* API:er som har hämtats från databasen, så att *collectStore* får samma uppbyggnad enligt figur 17.

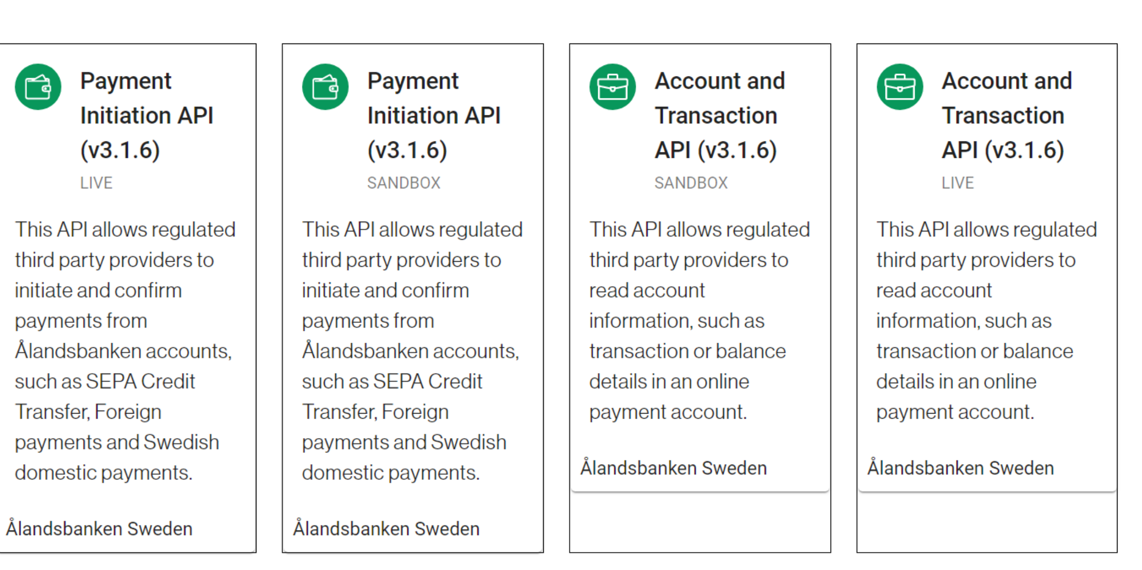
```
private async getStoreData() {
  return this.marketplaceService.getStores();
}

private async getApiData(collectStore: Store[]) {
  let item;
  for (item of collectStore) {
    this.mergeData(item, await this.marketplaceService.getApis(item.id));
  }
}

private mergeData(collectStore: Store, apis: Api[]) {
  collectStore.api = apis;
}
```

Figur 17, funktioner som hämtar API:er från databasen.

Resultatet ser ut som i figur 18.



Figur 18, en lista med API:er som har renderats ut med mat-cardmaterial.

4.1.2 Sök-implementation

I figur 19 visas hur filtreringen sker med *ngModel*, som är då en två-vägs data-bindning. Variabel *searchWord* sparar värdet, när användaren skriver i sökrutan. API:er filtreras på basen av sökordet (*searchWord*).

```
<input type="search" id="searchId" [(ngModel)]="searchWord">
```

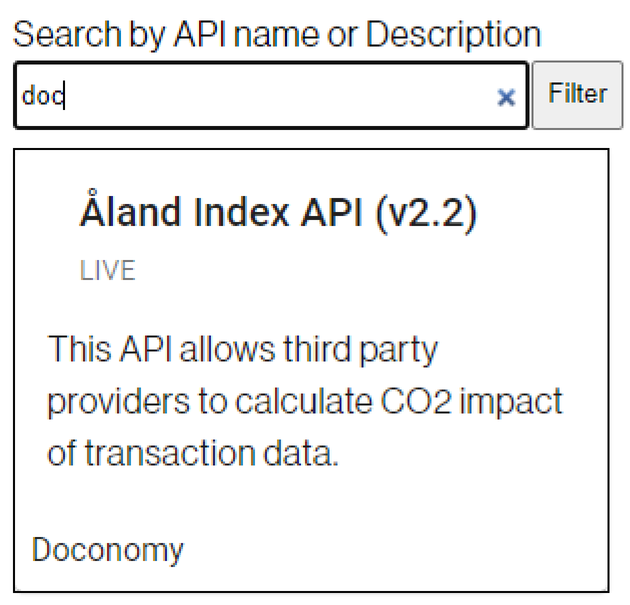
Figur 19 ngModel data bindning i sökrutan och ordet searchWord

I figur 20 lyssnar filter på sökordet (*searchWord*) som skickas till *pipe* (som beskrivs i kap 2.5.6).

```
<ng-container *ngFor="let data of collectStore | filter: searchWord">
```

Figur 20, filter-operatör med searchWord

Med 3 bokstäver får man resultat enligt figur 21.



Figur 21, resultat med sökning med 3 bokstäver.

4.1.3 Responsiv sida

I figur 22 har vi *ngClass.lt-lg* och list. List är klassnamnet som då berättar för HTML hur elementet skall se ut. NgClass är Angulars eget data-bindningsdirektiv som man kan använda med att lägga, ta bort eller byta CSS-klassnamn. *lt-lg* så kommer från Angular *Flex-Layout* API.

Figur 22, ngClass med lt-lg som ersätter med list som stilmall

Angular *Flex-Layout* är en *layout* API som ger sofistikerat utseende, beroende på vilken enhet man använder. I den här API använder man olika brytpunkter för att ge ett flexibelt rutnät och beskriver hur en applikation kan skala från små till extra stora skärmar.

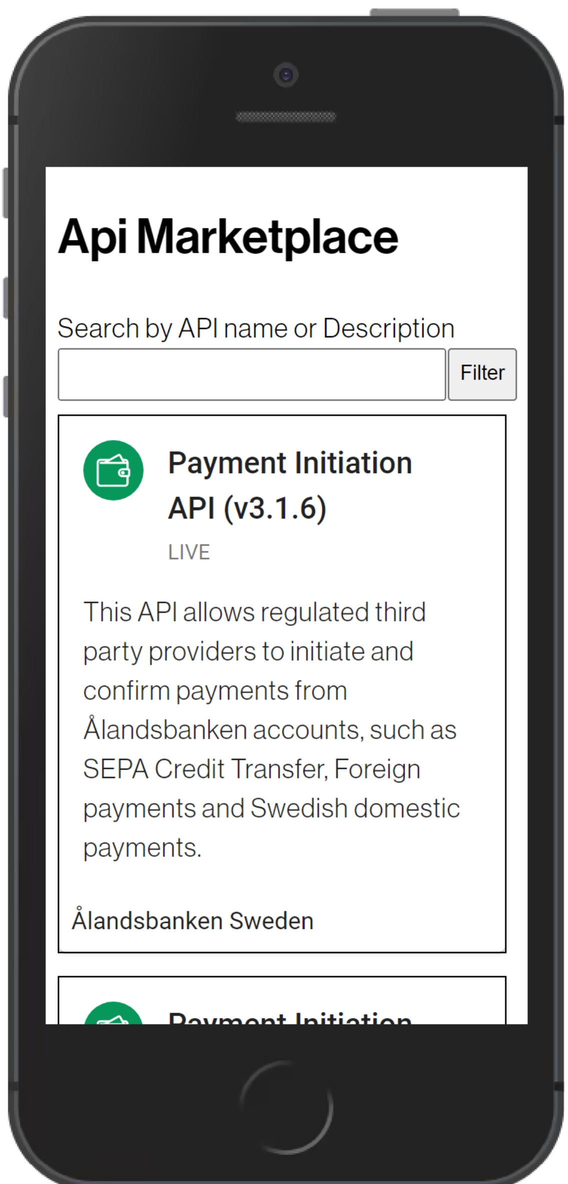
Med *mediaQuery* kan vi använda brytpunkt-alias som lt-lg. Lt-lg har färdigt värde vad max bredd på skärmen får vara och *mediaQuery* observerar hur stor skärm man använder och kontrollerar därefter på brytpunkt alias för att skala nedåt eller uppåt (Angular, 2019).

I figur 23 kan man gömma HTML-element om skärmbredden blir mindre än 1279 pixlar (px) och *fxLayout* berättar hur HTML-element skall ställas upp.

```
fxLayout="column" fxHide.lt-lg="true"
```

Figur 23, element-uppställning och gömma element efter visst antal pixlar

Resultatet ser ut enligt figur 24, när man använder IPhone 5 SE.



Figur 24, responsiv på IPhone 5 SE

4.1.4 Hur API:er skall representeras (box eller lista)

I kapitel 4.1.3 gick vi igenom vad *ngClass* är för någonting och hur man kan lägga till, ta bort och byta ut klassnamn, så i detta kapitel använder vi *ngClass* på annat sätt, med att använda booleska värdet för att ändra API-utseende, beroende på vilket status *activeButton* har när användaren klickar på knappen.

I figur 25 implementeras två knappar som hanterar hur API skall representeras och har id-attribut (box eller list). Knapparna är kopplade till en klickfunktion som kör funktion *classChange* och skickar in en händelse (*$event*) för att ändra API utseende.

```
<button mat-button id="box" (click)="classChange($event)">Box</button>
<button mat-button id="list" (click)="classChange($event)">List</button>
```

Figur 25, Knappar med klick-händelse som kontrollerar API-utseende

I figur 26 är *ngClass* kopplad till booleska uttrycket *activeButton* som övervakar om det är sant eller falskt. Om *activeButton* är falsk kommer API att representera enligt figur 29 (box) eller om användaren klickar på knappen med id *list*, så kommer API:erna representeras enligt figur 30 (list).

```
[ngClass]="activeButton ? 'list': 'box'">
```

Figur 26, ngClass lyssnar på booleskt uttryckt activeButton

I *TypeScript*-filen, så deklareras den booleska variabeln *activeButton* och färdigt klassnamn som skall representera API (*previousId*), när sidan har laddats färdigt (se figur 27). Dessa värden kommer att ändras i *classChange*-funktionen (se figur 28).

```
activeButton = false;
private previousId = 'box';
```

Figur 27, boolesk variabel och variabel för klassnamn

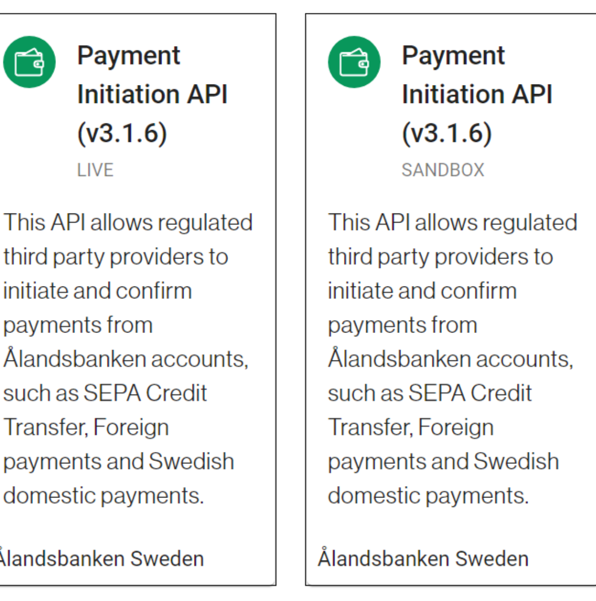
I figur 28 visas hur *classChange*-funktionen har *event* som parameter, där sparas vad det är för något (*target*), i dagligt tal en knapp (se figur 25). *Variabel idAttr* sparaknappens id attribut (se figur 25), och i variabel id så sparas knappens id-värde, om det är *box* eller *list*. I *if*-påståendet så kontrolleras om id och föregående id (*previousId*) ej har samma värde. Då ändras *activeButton*-värdet till sant eller falskt och så sparas den nya id värdet som föregående id (*previousId*). *ClassChange* funktionen kontrollerar på vilket tillstånd *activeButton* har och vilket var den senaste id-värdet var.

```
classChange(event) {
  const target = event.target || event.srcElement || event.currentTarget;
  const idAttr = target.attributes.id;
  const id = idAttr.nodeValue;
  if (id !== this.previousId) {
    this.activeButton = !this.activeButton;
    this.previousId = id;
  }
}
```

Figur 28, funktion classChange för att observera tillstånden på activeButton och id-värdet

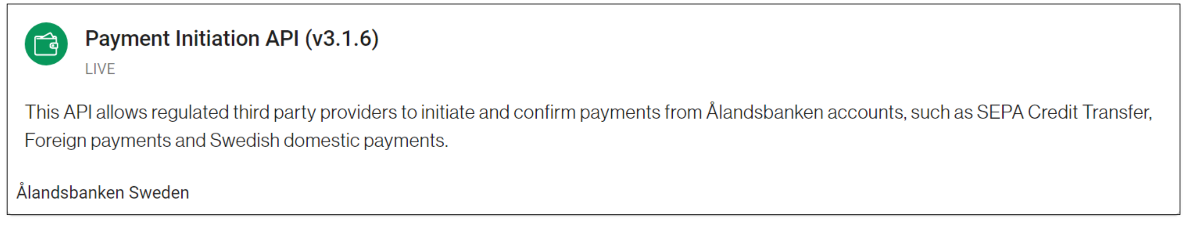
Variabel *const* innebär att när variabeln har tilldelats ett värde, kan man ej tilldela ett nytt värde till variabel *const*. Variabel *const* förblir oföränderligt. *Constant* (Konstant) är förkortat till *const* . Man kan deklarera variabler med *var* eller *let* ifall man vet att värdet på variabeln kommer att ändras och sparas i samma variabel (McGinnis, 2019).

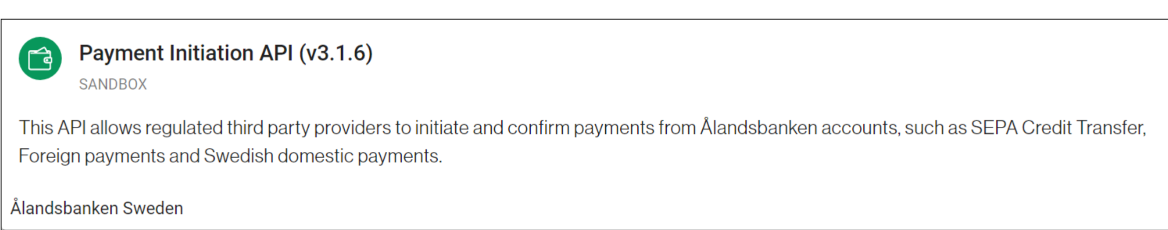
I *TypeScript* föredras det att deklarera variabler med *let*, för att *TypeScript* ger direkt felmeddelande ifall variabeln används före den har blivit deklarerad (Tektutorialshub, 2019). Som slutresultat när sidan laddas eller när användaren väljer box vy, representeras API:er enligt figur 29.



Figur 29, API i box vy

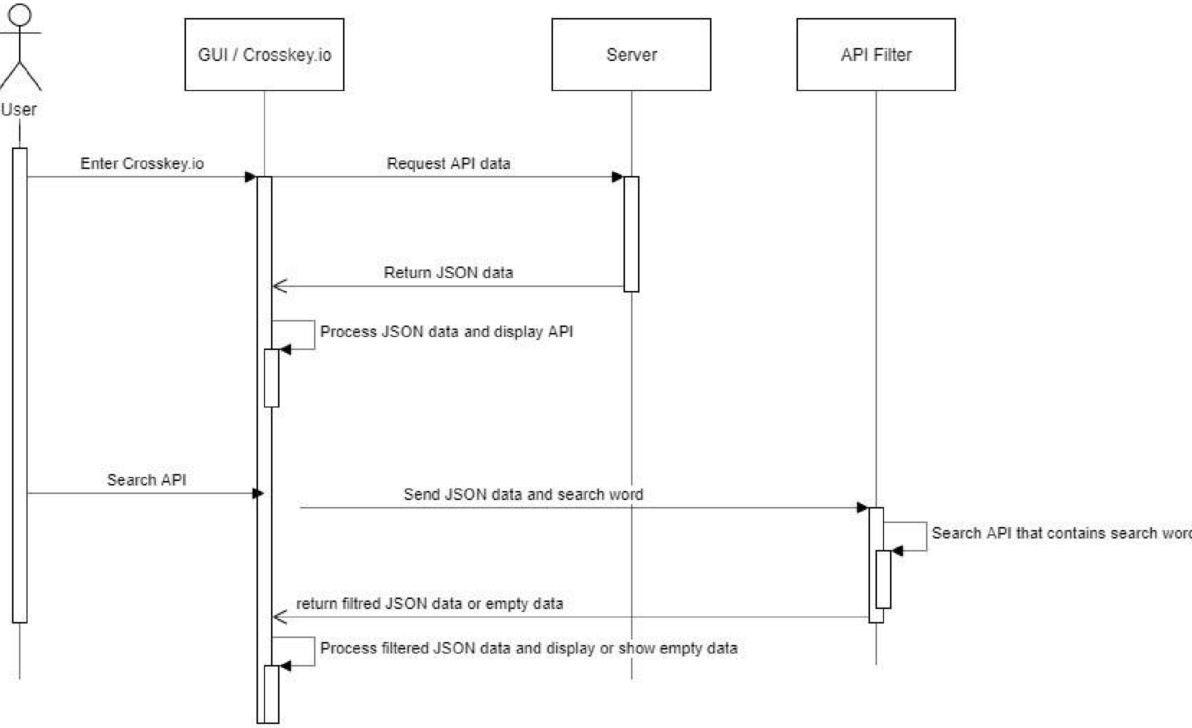
I Figur 30 visar hur det ser ut ifall användaren klickar på list-knappen. Då representeras API:er i listvy





Figur 30, API i listvy

I figur 31 ses hur det ser ut om man ritar upp en sekvensdiagram hur kapitel 4.1.1 och kapitel 4.1.2 anropas mellan server och API filter och vad som returneras.



Figur 31, sekvensdiagram

5. SLUTSATS

Detta uppdrag har varit att utveckla användargränssnitt till crosskey.io webbportal i Angular. Jag har lärt mig mycket om Angular hos Crosskey. Jag har tidigare kunskaper i webbutveckling inom *HTML*, *CSS*, *JavaScript* och *JQuery*, vilket gjorde mitt arbete lite lättare men intressantaste utmaningen för mig var att lära *TypeScript* och att bygga upp hela användargränssnitt i mindre komponenter.

Min förhoppning är att fortsätta och lära mig mera om Angular och även testa på flera intressanta programmeringsspråk och verktyg.

5.1 Framtida utveckling

Framtida utveckling för både webbportal crosskey.io och mitt eget examensarbete kunde vara att förbättra och anpassa så att användargränssnittet uppfyller EU:s tillgänglighetsdirektiv som trädde i kraft den 8:de januari 2021.

Tillgänglighetsdirektiv innebär att användare som har funktionsnedsättning eller är synskadade, skall kunna navigera och använda tjänsten utan större svårigheter med använda tangentbordet och/eller använda talsyntes.

KÄLL- OCH LITTERATURFÖRTECKNING

Angular. (2021). Angular - transforming data using pipes. Retrieved from https://angular.io/guide/pipes#marking-a-class-as-a-pipe

AngularJS. (2020,). AngularJS: Developer guide: Conceptual overview. Retrieved from https://code.angularjs.org/snapshot/docs/guide/concepts

BBVA. (2019, -10-16T03:20:58+00:00). Everything you need to know about PSD2 | BBVA. Retrieved from https://www.bbva.com/en/everything-need-know-psd2/

Brodsky, L., & Oakes, L. (2017). Data sharing and open banking. Retrieved from https://www.mckinsey.it/sites/default/files/data-sharing-and-open-banking.pdf

Crosskey. (2021,). Open banking - PSD2 as a service. Retrieved from https://www.crosskey.fi/openbanking/

Dubey, B. K. (2018, -06-14T18:39:39+00:00). Difference between TypeScript and JavaScript. Retrieved from https://www.geeksforgeeks.org/difference-between-typescript-and-javascript/

Felipe Pezoa, Domagoj Vrgoč, Martín Ugarte, Juan L. Reutter, & Fernando Suarez. (2016). Foundations of JSON schema., 263-273. Retrieved from https://martinugarte.com/media/pdfs/p263.pdf

Global Leading Conferences. (2017, -09-12T07:35:50+00:00). PSD2 - A game changing landscape for retail banking is coming. Retrieved from https://glceurope.com/psd2-a-game-changing-landscape-for-retail-banking-is-coming/

Hathaway, S. (2020). *What's the hype about open banking and how could it affect my business?* (). Retrieved from

https://www.sandfield.co.nz/news/opinion/526/whats-the-hype-about-open-banking-and-how-could-it-affect-my-business

Michels, B. (2018). Why you should use TypeScript on the backend too. Retrieved from https://medium.com/ae-studio/why-you-should-use-typescript-on-the-backend-too-22fa94efb768

node.js. (2021). About. Retrieved from https://nodejs.org/en/about/

Pabliq. (2021a). Börkrav. Retrieved from https://www.pabliq.se/ordlista/borkrav

Pabliq. (2021b). Skakrav. Retrieved from https://www.pabliq.se/ordlista/skakrav

Sowden, G. (2019, -10-29T18:11:46+00:00). Open banking generates cybersecurity risks. Retrieved from https://www.fintechnews.org/stronger-security-needed-before-open-banking-arrives/

Trend Micro Incorporated. (2019). The risks of open banking: Are banks and their customers ready for PSD2? Retrieved from https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/the-risks-of-open-banking-are-banks-and-their-customers-ready-for-psd2

TW, J. B. (2021). What is open banking? Retrieved from https://www.openbanking.org.uk/customers/what-is-open-banking/

Typescript. (2021). Documentation - TypeScript for JavaScript programmers. Retrieved from https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

Wikipedia. (2021a). *Angular (web framework)* Retrieved from https://en.wikipedia.org/w/index.php?title=Angular_(web_framework)&oldid=1004981399

Wikipedia. (2021b). *AngularJS* Retrieved from https://en.wikipedia.org/w/index.php?title=AngularJS&oldid=1003192961

Wikipedia. (2021c). *Node.js* Retrieved from

https://en.wikipedia.org/w/index.php?title=Node.js&oldid=1021215941