Examensarbete, Högskolan på Åland, Utbildningsprogrammet för Informationsteknik

WEBBASERAD SYNTHESIZER

- ett digitalt piano i webbläsaren

Mathias Rosenback

2022:44

Datum för godkännande: 30.12.2022
Handledare: Björn-Erik Zetterman

EXAMENSARBETE
Högskolan på Åland

Utbildningsprogram:	Informationsteknik
Författare:	Mathias Rosenback
Arbetets namn:	Webbaserad synthesizer - ett digitalt piano i webbläsaren
Handledare:	Björn-Erik Zetterman

Abstrakt

Syftet med det här examensarbetet är att skapa en webbaserad synthesizer med inspelningsfunktionalitet. Jag har valt detta projekt då jag har ett stort intresse av musik men även webbaserad utveckling och design.

Hela projektet är gjort med JavaScript-biblioteket React för att skapa ett smidigt användargränssnitt. Pianot i webbläsaren är designmässigt skapat från grunden men när det kommer till ljuduppspelningen använder jag mig av Tone.js som är ett webbljudramverk.

Resultatet är ett interaktivt piano med möjlighet att spela in och spara melodier, som lagras i en lista där man kan lyssna på allt man spelat in.

Nyckelord (sökord)
Javascript, React, Tone.js

Högskolans serienummer:	**ISSN:**	**Språk:**	**Sidantal:**
2022:44	1458-1531	Svenska	23

Inlämningsdatum:	**Presentationsdatum:**	**Datum för godkännande:**
5.12.2022	16.12.2022	30.12.2022

DEGREE THESIS
Åland University of Applied Sciences

Degree Programme:	Bachelor of Information Technology
Author:	Mathias Rosenback
Title:	Web-Based Synthesizer - A Digital Piano In The Browser
Academic Supervisor:	Björn-Erik Zetterman

Abstract

The purpose of this thesis is to create a web-based synthesizer with recording functionality. I have chosen this project as I have a great interest in music but also web-based development and design.

The entire project is made with the JavaScript library React to create a smooth user interface. The piano is design-wise created from scratch, but when it comes to the audio playback, I use Tone.js which is a Web Audio framework.

The result is an interactive piano with the ability to record and save melodies, which are stored in a list where you can listen to everything you have recorded.

Keywords

Javascript, React, Tone.js

Serial number:	ISSN:	Language:	Number of pages:
2022:44	1458-1531	Swedish	23 pages

Handed in:	Date of presentation:	Approved:
5.12.2022	16.12.2022	30.12.2022

INNEHÅLLSFÖRTECKNING

1. INLEDNING

1.1 Bakgrund

Musik har alltid varit en stor del av mig. När jag var liten började jag spela piano självlärt, med andra ord såg jag på YouTube-videos på personer som spelade och försökte härma noterna de spelade för att få till låten. Än idag spelar jag piano men har numera gått över till digital musikproduktion där jag kan skapa låtar med kombinationer av flera olika instrument än bara piano. När jag började studera IT på Högskolan på Åland så hade jag inte direkt något område inom IT jag var extra intresserad av, det var kul att få lära sig lite om allt möjligt. Efter ett tag började jag få intresse för webbaserad utveckling, speciellt frontend. Dessa två områden, musik och webbaserad utveckling, slog jag ihop och fick idén för vad jag skulle göra som mitt examensarbete, en webbaserad synthesizer.

1.2 Syfte

Syftet med mitt examensarbete är att skapa en synthesizer vilket är ett digitalpiano som konverterar ljudsignaler till ljudvågor, till skillnad från en vanlig klaviatur piano som slår strängar för att producera ljud. Denna synthesizer ska vara webbaserad och kunna producera ljud med funktion att spela in och spara inspelningarna i en lista på en skild sida där man kan lyssna på dem. Detta projekt är inget jag planerat att släppa för allmänheten att använda, utan det är ett projekt jag skapat för min egen skull för att få mera erfarenhet inom webbaserad utveckling, men även för hur man kan skapa instrument med kod, i det här fallet en synthesizer.

1.3 Metod

Projektet har en kravspecifikation, som beskrivs i kapitel 3. För vadera kravs beskrivs i kapitel 4 hur det är implementerat och testat. Denna process kalasl Evolutionary prototyping, då jag som utvecklare jobbar inkrementellt. De olika kraven är prioriterade och utförs enligt högst prioritet(skall krav) först och sedan de medelhögt prioriterade(bör krav) och sisst de med lägst prioritet(!kan va bra att ha”).

Hela projektet kommer göras i utvecklingsverktyget Visual Studio Code och skrivs i språket JavaScript där jag använder mig av biblioteket React för att skapa alla visuella delar. Programspråket JavaScript har jag lärt mig i skolan så jag har en del grunder för hur jag ska skapa detta projekt. React har jag erfarenhet av då jag gjort projekt med det i tidigare kurser. Själva synthesizern kommer således vara skapat från grunden med Javascript och React. Javascript för att få till all funktionalitet och React för att skapa en visuell synthesizer som man kan interagera med. Ljuduppspelningen görs med Tone.js, ett webbljudsramverk som möjliggör skapandet av musik på webbläsaren.

1.4 Avgränsningar

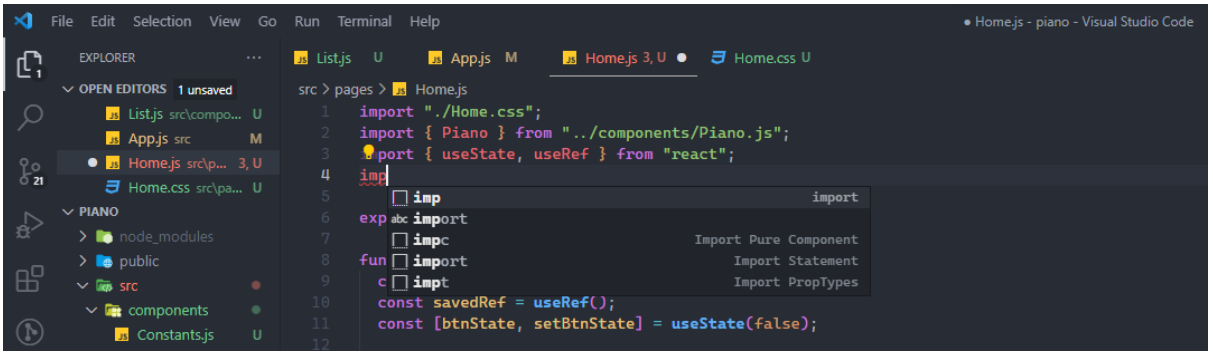
Webbapplikationen kommer inte att göras klart för allmänt bruk då det skulle krävas mera jobb för det. Istället kommer jag bygga upp en grund för de funktioner jag vill att applikationen ska ha och att de fungerar. Listan för inspelade melodier kommer även enbart sparas tillfälligt då applikationen är igång. När man startat om applikationen kommer inspelningarna att försvinna. Att implementera en databas där inspelningarna skulle sparas skulle således också kräva mera arbete än det jag kommer hinna med, därför har jag beslutat mig för att skala ned projektet så att jag hinner med de funktioner jag lagt upp i min kravspecifikation.

2. TEORI

Till att börja med kommer jag förklara de verktyg jag använt mig av, vilka funktioner de har och varför jag valt just det verktyget för att komma till mitt ändamål.

2.1 Utvecklingsmiljö

Mitt projekt har jag skrivit i Visual Studio Code, en programutvecklingsmiljö av Microsoft (*Documentation for Visual Studio Code*, n.d.). Med Visual Studio Code finns det ett installerbart tillägg direkt inne i programmet som heter Simple React Snippets som har underlättat för mig när jag skrivit mitt projekt med React. Tillägget gör så att jag kan skriva korta kommandon för att få en längre rad kod. Utöver det så har jag alltid gillat Visual Studio Code då det är enkelt och har allt som jag behöver, jämfört med andra miljöer jag testat. I Figur 1 ser man utvecklingsmiljön Visual Studio Code och användningen av tillägget *Simple React Snippets*.



Figur 1. Visual Studio Code med tillägget Simple React Snippets som ger färdig kod med kommandon, i detta fall "imp" för att få olika typer av "import".

2.2 JavaScript

JavaScript är ett programmeringsspråk som används i största del i webbutveckling, tillsammans med HTML och CSS. Programmeringsspråket skapades av Brendan Eich 1995 och används idag i upp till 98% av alla hemsidor på nätet (*JavaScript*, n.d.). Javascript används i *frontend* vilket är allt man ser på en hemsida, och *backend* vilket är det som sker bakom en hemsida, till exempel en databas som sparar information från sidan. Syntaxen, som

beskriver strukturen för ett programmeringsspråk (*What Is Syntax in Computer Programming?*, 2020), kan upplevas som enkel och lätt att lära sig jämfört med andra programmeringsspråk. I Figur 2 har vi ett exempel på kod i JavaScript som adderar två nummer och skriver ut det i konsolen.

```
function add(number1, number2) {
  return number1 + number2;
}

const number1 = 2;
const number2 = 3;

const result = add(number1, number2);

console.log(result);

```

Figur 2. Kod i JavaScript där vi först skapar två variabler med två siffror i sig och sedan anropar en funktion som adderar de två variablerna. Till sist skrivs resultatet ut i konsolen.

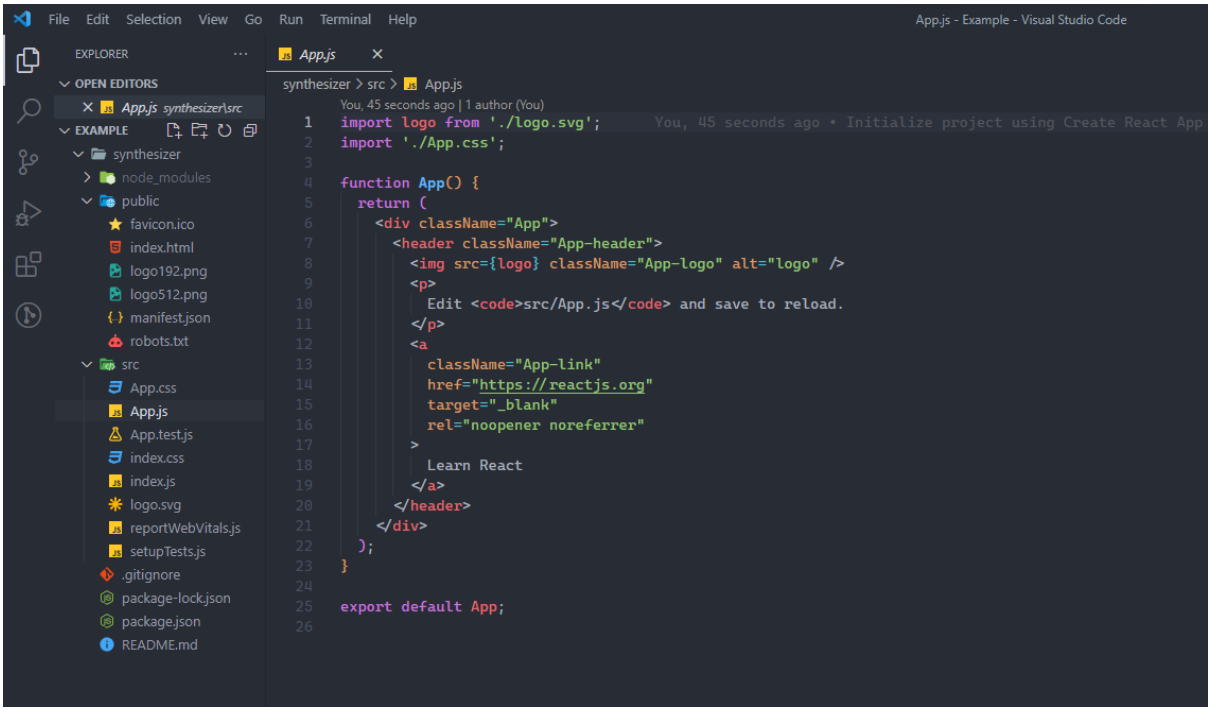
2.3 React

React är ett JavaScript-bibliotek för att bygga interaktiva användargränssnitt (*Getting Started – React*, n.d.). React är komponentbaserad, vilket betyder att man skapar komponenter som implementerar en *render()*-metod som returnerar det man vill att ska visas på skärmen. Komponenterna är skilda delar av användargränssnittet som man sedan kan kombinera till ett komplett användargränssnitt. T.ex. kan man skapa en komponent som är ett sökfält och en annan komponent som är en lista med resultat som man kombinerar för att få en fungerade sökning av produkter eller liknande i ett användargränssnitt. När man vill skapa ett projekt i React kör man kommandot “npx create-react-app synthesizer” i terminalen.

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\> npx create-react-app synthesizer
```

Figur 3. Kommandot för att skapa ett projekt i React.

Den första delen i kommandot vilket är *npx* som är ett verktyg för att exekvera Node packages (Marchán, 2017). *create-react-app* är själva kommandot för vad *npx* ska göra, som i det här fallet är att skapa ett projekt i React. *synthesizer* är namnet för vad projektet ska heta. I Figur 4 ser man hur ett projekt ser ut efter att man kört kommandot.



Figur 4. Projektet som skapas efter att man kört kommandot.

2.3.1 React Router

När man skapar ett projekt med React så följer det inte med något sätt att navigera till olika sidor, det är där biblioteket React Router kommer in (*Feature Overview V6.4.3 | React Router*, n.d.). Med React Router kan man lätt implementera sidor att navigera mellan. Till exempel i mitt projekt har jag taggen <*Routes*> i filen *App.js* som anropar *Home.js* och *Saved.js* som är två enskilda sidor jag skapat. Med <*Route*> taggen kan jag definiera en sökväg till de två filerna som fungerar på webbläsaren, där *Saved.js* får "/savedsongs" som sökväg och *Home.js* enbart får ett "/" eftersom det är startsidan. Till sist kan jag med infix notering länka en knapp i *Home.js* med en <*Link*> tagg som har sökvägen till *Saved.js* och tvärtom för att navigera. Koden för det finns i Figur 5 och ger en fungerande navigering mellan sidorna på webbläsaren.

```jsx
function App() {
  return (
    <div className="container">
      <div className="app">
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/savedsongs" element={<Saved />} />
        </Routes>
      </div>
    </div>
  );
}
```

```jsx
<Link to="/savedsongs" style={{ textDecoration: "none" }}>
  <button ref={savedRef} className="saved-btn">
    <h1 className="saved-text">Saved Songs</h1>
  </button>
</Link>
```

Figur 5. Implementation av React Routes och navigering mellan Home.js till Saved.js

2.4 Tone.js

Tone.js är ett webbljudramverk som möjliggör skapandet av musik på webbläsaren (*Tone.js*, n.d.). Med Tone.js kan man skapa olika typer av synthar med unika ljud och skicka ljudsignalerna till högtalaren. När man skapat en synth så kan man påverka allt från vilken not man ska spela, hur länge noten ska spelas och om den ska upprepas. Det finns även olika effekter man kan lägga till som påverkar ljudet, till exempel reverb och delay. I Figur 6 ser man exempel på skapandet av en enkel synth som sedan spelar c^1 (tonen ettsrukna c) / C på den fjärde oktaven i en sekund.

```jsx
const synth = new Tone.Synth().toDestination();

synth.triggerAttackRelease("C4", 1);
```

Figur 6. En synth som kopplas till högtalaren och spelar noten C4 i en sekund.

2.5 MediaRecorder API och dataformatet för musik

MediaRecorder API (Application Programming Interface) ger funktionen att spela in media som video och ljud (*MediaRecorder - Web APIs | MDN*, 2022). Genom att använda gränssnitet så har man kontroll över när inspelningen ska starta, sluta och även pausa den nuvarande inspelningen för att sedan fortsätta när man själv vill det. Man har även stor kontroll över formatet på inspelningen. För video kan man till exempel välja webm-, mp4-

eller avi-format. För ljud finns till exempel mp3-, ogg-, eller acc-format. I mitt fall har jag valt för ljudinspelning att använda mig av ogg-formatet, det vill säga Ogg Vorbis. Varför jag valt just det filformatet istället för mp3-format, som är det mest använda formatet för ljud, är för att jämfört med mp3 tar ogg-filer mindre utrymme och har oftast bättre ljudkvalité (*What Is Ogg Vorbis? | Definition From TechTarget*, n.d.).

3. KRAVSPECIFIKATION

I det här kapitlet kommer jag lägga upp de krav jag haft för projektet, som jag senare kommer hänvisa till i Kapitel 4 om implementationen. Kraven börjar med de viktigaste och slutar med de som inte är helt nödvändiga, men skulle va trevligt om jag lyckades implementera.

3.1 Krav på funktionalitet som skall uppfyllas

1. Ett visuellt/grafiskt synligt piano skapat för webbläsaren med 14 vita + 10 svarta tangenter.
2. Pianot visar visuellt vilka noter man trycker ned med hjälp av tangentbordet
3. Uppspelning av korrekt ljud(not/ton) beroende på vilket tangent man trycker ned

3.2 Krav på funktionalitet som bör uppfyllas

4. Möjlighet att spela in enstämmiga melodier, alltså följder av toner..
5. Lista med alla melodier man spelat in
6. Uppspelning av inspelad melodi.

3.3 Krav på funktionalitet som vore trevligt men icke nödvändigt

7. Olika effekter som påverkar ljudet t.ex reverb, distortion osv.

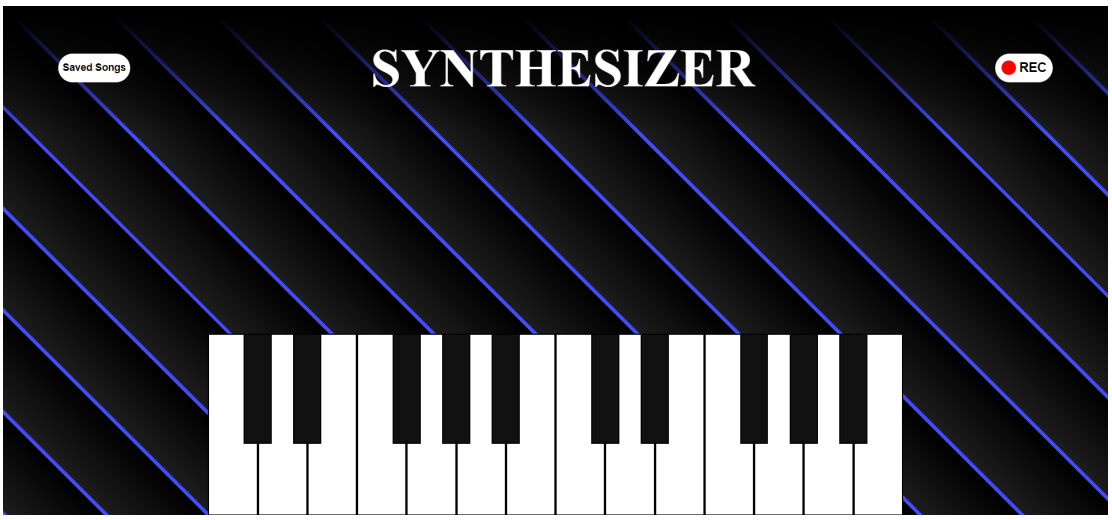
4. IMPLEMENTATION

Nu kommer jag gå in på hur jag skapat mitt projekt, det vill säga filer, funktioner och metoder. Alla rubriker i det här kapitlet är delar från mitt projekt som tillsammans blir den webbaserade synthesizer jag skapat.

4.1 Visuella Designen

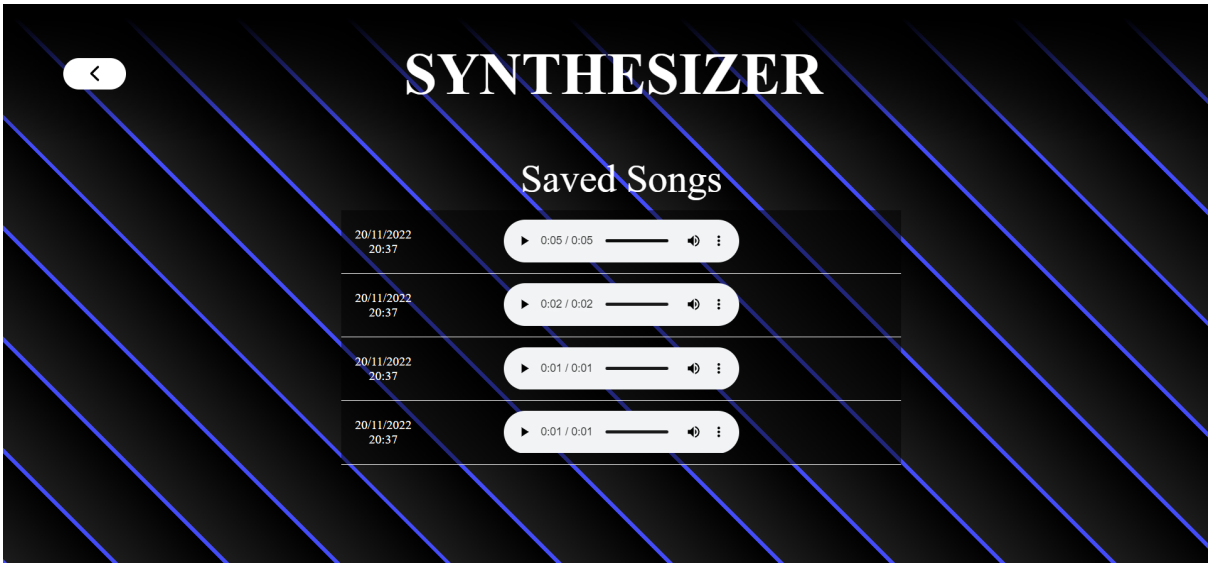
För min del var designen en viktig del av projektet, då det är något som jag tycker om, men med enbart en synthesizer som skulle i stort sett utgöra hela den första sidan så visste jag att det skulle bli en liten utmaning.

Jag bestämde mig för att skapa ett navigeringsfält högst upp på sidan med titeln och två knappar, en för att spela in det man spelar på synthesizern och en för att navigera till sidan med alla sparade inspelningar. Synthesizern blev placerad längst ner på sidan för att minska avståndet från tangentbordet för att underlätta när man försöker spela rätt not på tangentbordet och kunna se att det stämmer med det synthesizern speglar. Det stora problemet var då att det blev ett mellanrum från navigeringsfältet högst upp till den visuella synthesizern längst ner på sidan. För att då göra det lite mindre tomt så hittade jag olika abstrakta bakgrunder gjorda i CSS, som jag modifierade lite och fick en 3D-liknande bakgrund för att göra hela sidan lite mer intressant. I Figur 7 kan vi se hur resultatet blev av den första sidan med synthesizern.



Figur 7. Första sidan med synthesizern och två knappar, en för att navigera till sidan för inspelade melodier och en för att spela in.

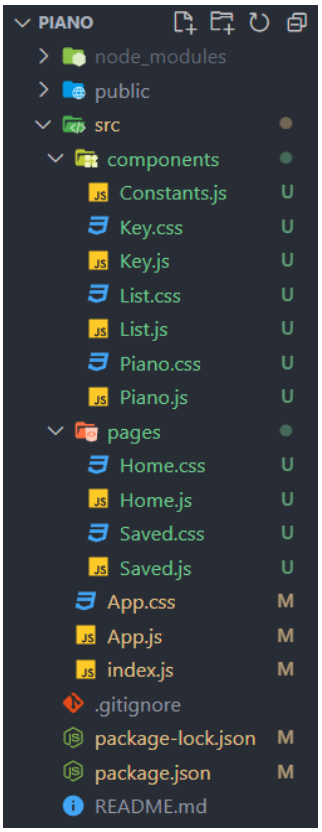
Den andra sidan för inspelningar fick låna samma design, ett navigeringsfält högst upp men i det här fallet enbart med en knapp för att navigera tillbaka och direkt under en lista för alla inspelningar. Detta uppfyller då ett krav som bör finnas från Kapitel 3.2, en lista med alla inspelade melodier. I Figur 8 kan vi se hur den sidan blev att se ut.



Figur 8. Andra sidan med en knapp för att navigera tillbaka till första sidan och en lista med alla sparade inspelningar.

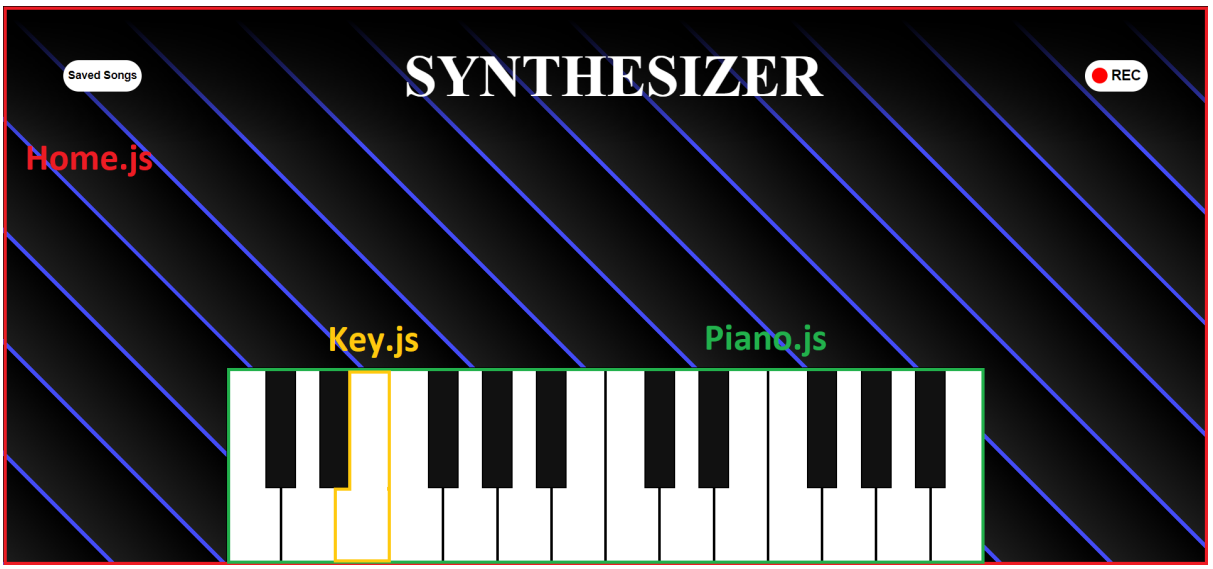
4.2 Filstruktur

Med React, som tidigare förklarades i kapitel 2.3, så kan man få en startkod med en färdig filstruktur. Det var där mitt projekt började och jag kunde lätt strukturera upp det med alla nya filer jag lade till. Jag skapade en mapp som heter *pages* där jag har de två sidorna för synthesizern och inspelning. Sedan har jag en skild mapp *components* som innehåller mina komponenter för synthesizern och för själva listan. I Figur 9 ser man hur strukturen ser ut för filerna i projektet.



Figur 9. Filstrukturen för projektet.

Filerna i mappen *components* används av filerna i mappen *pages* för att bygga upp de sidor som jag visade i Figur 7 och 8 i Kapitel 4.1. I Figur 10 ser man hur startsidan byggs upp med komponenterna *Key.js* och *Piano.js* där *Home.js* är själva sidan.



Figur 10. Uppbyggnaden av startsidan med komponenter.

4.3 Synthesizer

Hela synthesizern är byggd av flera olika komponenter. Om man går tillbaka till Figur 9 i Kapitel 4.2 så har jag tre stycken JavaScript-filer där i mappen *components*: *Constants.js*, *Key.js* och *Piano.js*.

Constant.js innehåller olika räckor där jag till att börja med lagrar alla tillåtna tangenter på skrivbordet och vilka tangenter som kommer höra till vita noter och svarta noter. Detta betyder att *Contant.js* egentligen inte är en komponent men jag valde att ha den i *components*-mappen eftersom det är en del av uppbyggnaden av synthesizern. Sedan lagrar jag de noter som ingår i synthesizern men även två räckor där jag sparar objekt för vilka tangenter på skrivbordet som ska höra till enskilda noter. I Figur 11 ser man hur en sådan räckor ser ut.

```
const KEY_TO_NOTE = {
  z: 'C3', s: 'C#3', x: 'D3', d: 'D#3', c: 'E3', v: 'F3', g: 'F#3', b: 'G3', h: 'G#3', n: 'A3', j: 'A#3', m: 'B3',
  q: 'C4', 2: 'C#4', w: 'D4', 3: 'D#4', e: 'E4', r: 'F4', 5: 'F#4', t: 'G4', 6: 'G#4', y: 'A4', 7: 'A#4', u: 'B4',
};
```

Figur 11. En räcka som innehåller objekt för tangenter på skrivbordet som hör till noter.

I filen *Key.js* börjar det hända lite mera saker. Den komponenten är i största del hjälp för att CSS:en ska bli rätt för alla enskilda noter på synthesizern, genom att kolla om den tangenten

på skrivbordet man trycker ner är en vit eller svart not och visa rätt CSS utifrån det men även för att visa visuellt när man trycker ner en not på skärmen. Jag har en *keyClassName*-variabel som till att börja med innehåller texten *"key"*. Sedan kör jag två funktioner som kollar först om det är en svart tangent och om det är sant så läggs texten " flat" till i *keyClassName* som då kommer få ett skilt utseende från CSS-filen. Andra funktionen kollar om noten och dess tillhörande tangent existerar, då läggs *" pressed"* till istället och får på samma sätt ett skilt utseende. Dessa funktioner ger resultatet till en av mina krav som ska uppfyllas i Kapitel 3.1, att synthesizern visuellt ska visa vilka noter som trycks ner. I Figur 12 ser man lite hur det går till när man kollar noterna.

```
noteIsFlat = (note) => {
  return note.length > 2;
}

keyIsPressed = (note, pressedKeys) => {
  return _.includes(pressedKeys, NOTE_TO_KEY[note]);
}

render() {
  let keyClassName = "key";
  const noteIsFlat = this.noteIsFlat(this.props.note);
  const keyIsPressed = this.keyIsPressed(this.props.note, this.props.pressedKeys);
  if (noteIsFlat) {
    keyClassName += " flat";
  }
  if (keyIsPressed) {
    keyClassName += " pressed";
  }
```

Figur 12. Funktioner för att kolla om noten är svart och om tangenten på skrivbordet och dess tillhörande not existerar.

Till sist har vi filen *Piano.js* som då bygger ihop synthesizern med alla noter från *Key.js*. Här sker även funktionerna för att spela upp ljud för noterna man trycker ner och inspelningen av melodier men det kommer jag prata om i senare kapitel. Här är det viktiga att komponenten kollar att noten visuellt på skärmen fortsätter vara nedtryck så länge man trycker ner en tangent på skrivbordet och att noten på skärmen uppdateras när man släpper tangenten. I Figur 13 visas funktionerna för att kolla det.

```
handleKeyDown = (event) => {
  if (event.repeat) {
    return;
  }
  const key = event.key;
  const updatedPressedKeys = [...this.state.pressedKeys];
  if (!updatedPressedKeys.includes(key) && VALID_KEYS.includes(key)) {
    updatedPressedKeys.push(key);
  }
  this.setState({
    pressedKeys: updatedPressedKeys,
  });

  console.log(KEY_TO_NOTE[key]);
  synth.triggerAttack(KEY_TO_NOTE[key]);
};

handleKeyUp = (event) => {
  const index = this.state.pressedKeys.indexOf(event.key);
  if (index > -1) {
    this.setState((state) => ({
      pressedKeys: state.pressedKeys.splice(index, 0),
    }));
  }
  synth.triggerRelease(KEY_TO_NOTE[event.key]);
};
```

Figur 13. Två funktioner som hanterar när tangenten är nedtryckt, samt när man släpper tangenten.

Alla dessa filer bygger då som sagt ihop en synthesizer som visuellt visar vilken not man trycker ner utifrån vilken tangent jag har valt att låta den tillhöra. Med det uppfyller jag ett annat krav som jag hade bestämt, ett visuellt piano med 14 vita och 10 svarta noter.

4.4 Ljuduppspelning

Min första idé för hur jag skulle spela upp ljud från synthesizern var med mp3-filer för varje not som jag kartlade genom att ha samma not som i arrayen från Constants.js som filnamn och jämföra det med noten på synthesizern, så att rätt mp3-fil spelas. Mp3-filerna skapade jag i ett musikprogram jag har där jag enkelt kunde spela alla noter på pianot och exportera det som skilda filer. Det funkade men problemet var att filen spelar samma längd på noten som när jag spelade in det, så oavsett hur länge man håller in en tangent på skrivbordet så är det alltid samma längd som spelas. Dessutom gillade jag inte hur kvalitén blev när jag spelade flera noter samtidigt. Så jag fick tänka om och komma på en annan lösning, det var då jag hittade *Tone.js*.

Tone.js är, som tidigare förklarat i Kapitel 2.5, ett webbljudramverk som gör det möjligt att skapa ljud direkt i webbläsaren. Efter att jag installerat och importerat *Tone.js* till mitt projekt så kunde jag skapa en synth, där jag först anropar *Tone*. Sedan väljer jag vilken typ av synth

jag vill ha som i mitt fall var en *PolySynth* som gör det möjligt att spela flera noter samtidigt. Till sist väljer jag att ljudet ska komma från primära ljudutgången, det vill säga den ljudutgång man har aktiv för tillfället. I Figur 14 nedan ser vi koden för detta.

```
const synth = new Tone.PolySynth().toDestination();
```

Figur 14. Skapandet av en synth med typen PolySynth och primär ljudutgång.

Nu när synthen är skapad och redo att använda så behöver jag bara anropa den först med *triggerAttack* i funktionen för att hantera när en tangent trycks ned och sedan med *triggerRelease* i funktionen för att hanter när man släpper en tangent. Båda anropen ser ni i Figur 12 i Kapitel 4.3. Nu spelas ljud från synthen så länge man håller ned en tangent och uppfyller då även mitt sista krav för mitt projekt, att kunna spela upp rätt ljud beroende på vilken tangent man trycker ned.

4.5 Ljudinspelning

Nu kommer vi till sista delen av mitt projekt, ljudinspelningen. Detta var den svåraste delen och tog mig längst tid att göra. Till att börja med så skapade jag en *isRecording*-variabel som ska få ett värde beroende på om man spelar in eller inte. Dessutom har jag en *btnState*-variabel som gör så att jag kan ha inspelningsknappen på navigeringsfältet i ett aktivt tillstånd, till skillnad från en vanlig knapp som är i ett aktivt tillstånd så länge musen är nedtryckt på den. Allt detta körs sen i en funktion som jag kallar *record()*, så om knappen för inspelning är aktiv så blir *isRecording*-variabeln *true*, om knappen blir avaktiverad så blir variabeln *false*. Eftersom jag har knappen på filen *Home.js* och inspelningsfunktionen på *Piano.js* så måste jag exportera *isRecording*-variabeln för att kunna använda den i Piano.js. I Figur 15 ser vi hur funktionen fungerar.

```
export var isRecording = false;

function Home() {
  const recordRef = useRef();
  const savedRef = useRef();
  const [btnState, setBtnState] = useState(false);

  function record() {
    setBtnState((btnState) => !btnState);
    if (!isRecording) {
      isRecording = true;
    } else {
      isRecording = false;
    }
  }

  let toggleClassCheck = btnState ? " active" : null;
```

```
<button
  ref={recordRef}
  className={`record-btn${toggleClassCheck}`}
  onClick={record}
>
```

Figur 15. Funktionen record() som sätter värdet på isRecording-variabeln och inspelningsknappen som anropar funktionen.

Nu när jag har en variabel som jag kan importera till *Piano.js*-filen och kolla med om inspelningsknappen är aktiv eller inte så kommer vi till funktionen för själva inspelningen. Jag började med att skapa ett objekt av *MediaRecorder()* som jag kallar *recorder*. Sedan skapade jag variabler som gör det möjligt att koppla ihop ljudet från synthen till *recorder* som lagras. När jag hade det klart var det bara att använda sig av i*sRecording*-variabeln för att bestämma när *recorder* ska spela in och när det ska stoppas. För att spela in behöver jag bara anropa *recorder.start()* och sedan *recorder.stop()* för att avsluta inspelningen. När jag spelat in en melodi lagras den i en så kallad *Blob*, vilket är ett objekt för rådata (*Blob - Web APIs | MDN*, 2022) som jag först gör till en OGG-fil som sedan refereras med en URL (Uniform Resource Locator) som gör det möjligt för en musikspelare att läsa in datan. I Figur 16 ser vi hur en *Blob* med sparad data ser ut.

```
▶ Blob {size: 32288, type: 'audio/ogg; codecs=opus'}    Piano.js:88
```

Figur 16. En Blob för inspelad data utskriven på konsolen i Google Chrome.

URL:en som jag konverterade till sparas i en variabel jag kallar *source* och på samma sätt som variabeln *isRecording* så exporterar jag den nu istället till *List.js* som lägger in den till en musikspelare som sedan går att spela upp från den. Detta uppfyller då mitt sista krav som bör uppfyllas att kunna spela in melodier och spara det i en lista. I Figur 17 syns en del av koden i *List.js* för listan där jag först har en funktion som lägger till ett nytt objekt i listan, det vill säga först sparar jag *input* lokalt, som då är data av inspelningen. Där skapar jag också datum och tid för när inspelningen gjordes. Till sist läser jag in allt längst ner i *function List()*.

```
export function addToList(input) {
  src = input;
  date =
    currentDate.getDate() +
    "/" +
    (currentDate.getMonth() + 1) +
    "/" +
    currentDate.getFullYear() +
    "\n" +
    currentDate.getHours() +
    ":" +
    currentDate.getMinutes();
  key += 1;
  const newSongs = songs.concat({ src, date, key });
  songs = newSongs;
}

function List() {
  return (
    <ul className="list">
      <div className="title">Saved Songs</div>
      {songs.reverse().map((song) => (
        <li key={song.key} className="content">
          <div className="date">{song.date}</div>
          <audio className="player" src={song.src} controls></audio>
        </li>
      ))}
    </ul>
  );
}
```

Figur 17. Kod från List.js som visar när jag lägger till ett nytt objekt i listan.

5. SLUTSATSER

Resultat av mitt projekt blev en visuell, interaktiv synthesizer med funktion att spela in melodier och lyssna på dem från en lista. Kraven jag satte som skulle uppfyllas och bör uppfyllas fick jag ihop i slutet av arbetet vilket jag är nöjd med.

Arbetet jag gjort har varit lärorikt och jag har fått lära mig allt från hur man kan skapa ett visuellt piano till att spela in och spela upp ljud från en webbläsare. Det har varit kul att lyckas få ihop ett projekt, av en hobby som jag tycker väldigt mycket om med ett ämne jag kommer jobba med i framtiden, som examensarbete.

Det finns aspekter i projektet som jag vill bygga vidare på i framtiden. Till exempel skulle jag vilja få alla inspelningar sparade på en server då man för tillfället enbart har inspelningarna tillfälligt sparade lokalt då webbsidan är igång. När man stänger ned den så försvinner alla inspelningar. En annan funktion jag skulle vilja få implementerat är olika effekter på synthen som var det krav jag hade som vore roligt men inte nödvändigt. Det skulle ge en lite mera möjlighet att påverka ljudet. Utöver det är jag nöjd med det arbetet jag gjort och kan numera börja säga att jag lärt mig spela musik genom kod, som ett instrument.

KÄLLFÖRTECKNING

Blob - Web APIs | MDN. (2022, September 15). MDN Web Docs. Retrieved December 1, 2022, from https://developer.mozilla.org/en-US/docs/Web/API/Blob

Documentation for Visual Studio Code. (n.d.). Visual Studio Code. Retrieved November 28, 2022, from https://code.visualstudio.com/docs

Feature Overview v6.4.3 | React Router. (n.d.). React Router. Retrieved November 28, 2022, from https://reactrouter.com/en/main/start/overview

Getting Started – React. (n.d.). React. Retrieved November 28, 2022, from https://reactjs.org/docs/getting-started.html

JavaScript. (n.d.). Wikipedia. Retrieved November 28, 2022, from https://en.wikipedia.org/wiki/JavaScript

Marchán, K. (2017, July 10). *Introducing npx: an npm package runner | by Kat Marchán*. Medium. Retrieved December 3, 2022, from https://medium.com/@maybekatz/introducing-npx-an-npm-package-runner-55f7d4bd282b

MediaRecorder - Web APIs | MDN. (2022, November 28). MDN Web Docs. Retrieved December 3, 2022, from https://developer.mozilla.org/en-US/docs/Web/API/MediaRecorder

Tone.js. (n.d.). Tone.js. Retrieved December 1, 2022, from https://tonejs.github.io/

What is Ogg Vorbis? | Definition from TechTarget. (n.d.). TechTarget. Retrieved December 3, 2022, from https://www.techtarget.com/whatis/definition/Ogg-Vorbis

What is Syntax in Computer Programming? (2020, June 1). Woz U. Retrieved December 3, 2022, from https://woz-u.com/blog/what-is-syntax-in-computer-programming/