EJ-Box

Kod-dokumentation

En digital jukebox

Tommy Friberg
Examensarbete 20 p
.NET-Utveckling
VT 2019

Innehållsförteckning

Förord

Denna kod-dokumentation är uppbyggd så att alla främst komponenter men även andra delar av detta program så som hela backend-delen med dess databas, kontext, modellklass med mera och övriga delar från MVC-arkitekturen så som modell och Controller, kommer alla att gås igenom på så sätt att först kommer ett eller flera stycken där först det aktuella kodavsnittet funktionalitet/syfte beskrivs översiktligt och därefter följer själva koden.

Denna inledande och beskrivande text för varje förekommande kod-avsnitt kommer variera i omfattning och sätt på vilket de beskrivs, på grund av att kortare-kodavsnitt kommer man kunna beskriva aningen mer ingående medans de som verkligen är längre kommer ha en mer översiktlig beskrivning. Det här dokumentet med all dess kod är ändå som det är väldigt långt. Finns helt enkelt inte plats för mer ingående beskrivningar av all förekommande kod.

Komponenter

AdminPlaylists.tsx (parent)

AdminPlaylist är parent till AddPlaylist och EditPlaylist och utgör den enda egentliga funktionella Parent-Child-kopplingen i min applikation. Resterande komponenter är fristående. Förutom kanske att komponenten PlayerMachine rent tekniskt talat är child till komponenten Home, men det är mer en konstruktion för att åstadkomma så programmets viktigaste del/komponent PlayerMachine hamnar under Home, än något mer övergripande funktionellt.

Adminplaylist utgör grunden för programmets CRUD-del där låtar och spellistor kan läsas, läggas till, tas bort, ändras. Man kommer först till AdminPlaylist komponenten och kan sedan via denna klicka på knappen för att öppna upp komponenterna (**eller i denna kontext kallar jag dem för areor**) för att lägga till låtar/spellistor eller ändra i dem. Ta bort (delete) som utgör den sista biten i en CRUD utgörs bara i AdminPlaylist av en enda knapp då den inte behöver någon egen area.

Förutom navigeringsknappar i Adminplaylist för att ta sig framåt och genom aktuell spellista så finns det även en knapp som man kan ta fram och ta bort (“toggla”) och välja den spellista som man vill titta på eller jobba med. Den låt som man sedan med navigeringsknapparna stegar sig fram till blir den som behandlas med add. edit eller delete.

```
import * as React from 'react';
import { RouteComponentProps, Redirect, Route } from 'react-router';
import 'isomorphic-fetch';
import { AddPlaylist } from './AddPlaylist';
import { EditPlaylist } from './EditPlaylist';



interface AdminPlaylistsProps {

}



interface AdminPlaylistsState {
    loading: boolean;
```

```
    songs: Song[];
    stepCounter: number;
    choosenSongId: number;
    replyText: string;
    isRadioBtnDisabled: boolean;
    replyClassName: string;
    isMainAreaVisible: boolean;
    //scoreState: number;
    isAddSongAreaVisible: boolean;
    isToAddSongButtonVisible: boolean;
    isEntryBtnHidden: boolean;
    isNextBtnHidden: boolean;
    isPrevBtnHidden: boolean;
    isEditSongAreaVisible: boolean;
    isToEditSongButtonVisible: boolean;
    choosenPlaylist: string;
    uniquePlaylistNames: string[];
    playlists: Song[];
    collapsePlaylists: boolean;


}




export interface Song {
    id: number;
    _artist: string;
    albumTitle: string;
    songTitle: string;
    songFile: string;
    playlistName: string;
    songFreq: number;
 }




export class AdminPlaylists extends React.Component<RouteComponentProps<{}>,
AdminPlaylistsState> {
    constructor() {
        super();
        this.state = {
```

```
    loading: false,
    songs: [],
    stepCounter: 0,
    choosenSongId: 0,
    replyText: '',
    isRadioBtnDisabled: false,
    replyClassName: '',
    isMainAreaVisible: true,
    //scoreState: 0,
    isAddSongAreaVisible: false,
    isToAddSongButtonVisible: true,
    isEntryBtnHidden: true,
    isNextBtnHidden: true,
    isPrevBtnHidden: false,
    isEditSongAreaVisible: false,
    isToEditSongButtonVisible: true,
    choosenPlaylist: 'BASIC LIST',
    uniquePlaylistNames: [''],
    playlists: [],
    collapsePlaylists: true


};


this.nextSong = this.nextSong.bind(this);
this.addSongArea = this.addSongArea.bind(this);
this.returnToMainArea = this.returnToMainArea.bind(this);
this.prevSong = this.prevSong.bind(this);
this.EditSongArea = this.EditSongArea.bind(this);
this.deletePlaylist = this.deletePlaylist.bind(this);

this.checkIfLastSongOfList = this.checkIfLastSongOfList.bind(this);
this.checkIfFirstSongOfList = this.checkIfFirstSongOfList.bind(this);


this.loadSpecificPlaylist = this.loadSpecificPlaylist.bind(this);

this.loadUniquePlaylists = this.loadUniquePlaylists.bind(this);

this.loadChoosenPlaylist = this.loadChoosenPlaylist.bind(this);

this.generateUniqePlaylist = this.generateUniqePlaylist.bind(this);
```

```
        //laddar in unik lista av befintliga spellistor
        this.loadUniquePlaylists();


        //laddar in default spellista
        this.loadSpecificPlaylist();




    } //konstruktorn slutar här




    public render() {


        let stepCounter = this.state.stepCounter;    //stegräknare samt anger index på aktuell
sång

        let songStepCount = this.state.stepCounter;  //steg-räknare som anger position för sång
i aktuell sånglista


        //renderar unik spellista med html-tags
        let ulplaylists: JSX.Element[];
        ulplaylists = [];

        for (let i = 0; i <= this.state.uniquePlaylistNames.length - 1; i++) {
            ulplaylists[i] = <button key={i} value={this.state.uniquePlaylistNames[i]}
onClick={this.loadChoosenPlaylist}
                 className="list-group-item list-group-item-action"  >
{this.state.uniquePlaylistNames[i]}</button >;

        }



        let contents = this.state.loading
```

```
        ? this.renderPlaylistTable(this.state.songs, stepCounter, songStepCount, ulplaylists)
        : <p><em>Loading...</em></p>;


      return <div> {contents}


      </div>;


  }




  public renderPlaylistTable(songs: Song[], stepCounter1: number, questionStepCount1:
number, ulplaylists: JSX.Element[]) {


      return <div>


        <ul className="list-group" hidden={!this.state.isMainAreaVisible}>



          <div className="list-group-item" id="songheadercoloradmin">Song
{questionStepCount1 + 1}/{this.state.songs.length}: Playlist
{songs[stepCounter1].playlistName}
          </div>



           <label className="list-group-item">
             Artist: {songs[stepCounter1]._artist}
          </label>


          <label className="list-group-item">
            Album Title:  {songs[stepCounter1].albumTitle}
          </label>

          <label className="list-group-item">
```

```
            Song Title: {songs[stepCounter1].songTitle}
          </label>

          <label className="list-group-item">
            Song File: {songs[stepCounter1].songFile}
          </label>


          <label className="list-group-item">
            Playlist Name: {songs[stepCounter1].playlistName}
          </label>




          <button hidden={!this.state.isPrevBtnHidden} className="btn-pill btn-danger
btn-lg"
            type="button" onClick={this.prevSong} id="prevbutton">&#65308;
Previous</button>



          <button hidden={!this.state.isNextBtnHidden} className="btn-pill btn-sucess
btn-lg"
            type="button" onClick={this.nextSong} id="nextbutton" >Next
&#65310;</button>



          <h3 className='messageText'>{this.state.replyText}</h3>



          <button hidden={!this.state.isToAddSongButtonVisible}
            className="btn-pill btn-sucess btn-lg" type="button"
            onClick={this.addSongArea} id="toaddsongbutton">To: Add Song</button>

          <button hidden={!this.state.isToEditSongButtonVisible}
            className="btn-pill btn-sucess btn-lg" type="button"
            onClick={this.EditSongArea} id="toeditsongbutton">To: Edit Song</button>


          <button hidden={!this.state.isToEditSongButtonVisible}
            className="btn-pill btn-sucess btn-lg" type="button"
            onClick={this.deletePlaylist} id="deletesongbutton">Delete Song</button>
```

```
{/*  unik lista över spellistor   */}


            <button id="buttonCollapsePlaylistsAdmin" className="btn-pill btn-sucess btn-lg"
type="button"
                onClick={this.onCollapsePlaylists}>Playlists On/Off</button>

          <div  id="adminlistofplaylists"  hidden={this.state.collapsePlaylists}>



              <div className="row">

                <div className="list-group list-group-tf" id="playlistadminprops">
                  {ulplaylists}
                </div>
              </div>

          </div>



          </ul>




          <div hidden={!this.state.isAddSongAreaVisible}>


             <AddPlaylist

                loadSpecificPlaylist={this.loadSpecificPlaylist.bind(this)}
                choosenPlaylist={this.state.choosenPlaylist}
             />



             <button className="btn btn-danger btn-lg"
                onClick={this.returnToMainArea} id="backToMainAreaButton"> Back </button>
```

```
    </div>



    <div hidden={!this.state.isEditSongAreaVisible}>


        <EditPlaylist

            id={this.state.songs[this.state.stepCounter].id}
            _artist={this.state.songs[this.state.stepCounter]._artist}
            albumTitle={this.state.songs[this.state.stepCounter].albumTitle}
            songTitle={this.state.songs[this.state.stepCounter].songTitle}
            songFile={this.state.songs[this.state.stepCounter].songFile}
            playlistName={this.state.songs[this.state.stepCounter].playlistName}
            songFreq={this.state.songs[this.state.stepCounter].songFreq}

            loadSpecificPlaylist={this.loadSpecificPlaylist.bind(this)}

            choosenPlaylist={this.state.choosenPlaylist}

         />



        <button className="btn btn-danger btn-lg"
            onClick={this.returnToMainArea} id="backToMainAreaButton2"
            target="_self">Back
        </button>


    </div>









</div>;   //return slutar här
```

```
  }   //klass slutar här




  checkIfLastSongOfList() {

    if (this.state.stepCounter + 1 == this.state.songs.length - 1) {  //spellistans längd
      this.setState({ replyText: "Last song in playlist" })

      this.setState({ isNextBtnHidden: false });
      this.setState({ isPrevBtnHidden: true });


    }
  }




  checkIfFirstSongOfList() {

    if (this.state.stepCounter - 1 == 0) {  //början av spellistan
      this.setState({ replyText: "First song in playlist" })

      this.setState({ isNextBtnHidden: true });
      this.setState({ isPrevBtnHidden: false });


    }
  }







  nextSong(event: any) {


    if (this.state.stepCounter == this.state.songs.length - 1) {  //så man inte hamnar utanför
arrayen/spellistan
```

```
        this.setState({ replyText: "Ooops, outside the playlist, back up a bit :)" })
    }
    else {


        let count = this.state.stepCounter + 1;
        this.setState({ stepCounter: count });


        this.setState({ choosenSongId: 0 });
        this.setState({ replyText: '' });
        this.setState({ isNextBtnHidden: true });
        this.setState({ isPrevBtnHidden: true });
        this.setState({ isEntryBtnHidden: true });

        this.setState({ isRadioBtnDisabled: false });

        this.checkIfLastSongOfList();

    }



}




 prevSong(event: any) {
    let count = this.state.stepCounter - 1;
    this.setState({ stepCounter: count });


    this.setState({ choosenSongId: 0 });
    this.setState({ replyText: '' });
    this.setState({ isNextBtnHidden: true });
    this.setState({ isPrevBtnHidden: true });
    this.setState({ isEntryBtnHidden: true });

    this.setState({ isRadioBtnDisabled: false });

    this.checkIfFirstSongOfList();

 }
```

```
//används för att ta fram area där man skall lägga till en låt
addSongArea(event: any) {
    this.setState({ isToAddSongButtonVisible: false });
    this.setState({ isToEditSongButtonVisible: false });
    this.setState({ isNextBtnHidden: false });
    this.setState({ isPrevBtnHidden: false });
    this.setState({ isEntryBtnHidden: false });
    this.setState({ isRadioBtnDisabled: true });
    this.setState({ isMainAreaVisible: false });
    this.setState({ isAddSongAreaVisible: true })
    this.setState({ isEditSongAreaVisible: false })
}



// samma som ovan fast för att ändra en låt
EditSongArea(event: any) {
    this.setState({ isToAddSongButtonVisible: false });
    this.setState({ isToEditSongButtonVisible: true });
    this.setState({ isNextBtnHidden: false });
    this.setState({ isPrevBtnHidden: false });
    this.setState({ isEntryBtnHidden: false });
    this.setState({ isRadioBtnDisabled: true });
    this.setState({ isMainAreaVisible: false });
    this.setState({ isAddSongAreaVisible: false })
    this.setState({ isEditSongAreaVisible: true })


    this.state.songs.length + 1;

}



//används för att komma tillbaka till arean med låtarna/spellistorna
returnToMainArea(event: any) {
    let count = this.state.stepCounter;

    let questionUpdate = this.state.songs;
    this.setState({ songs: questionUpdate });

    this.setState({ isAddSongAreaVisible: false });
```

```
        this.setState({ isEditSongAreaVisible: false });
        this.setState({ isMainAreaVisible: true });

        this.setState({ replyText: '' });
        this.setState({ isRadioBtnDisabled: false });
        this.setState({ isEntryBtnHidden: true });
        this.setState({ isNextBtnHidden: true });
        this.setState({ isPrevBtnHidden: true });
        this.setState({ isToAddSongButtonVisible: true });
        this.setState({ isToEditSongButtonVisible: true });

        console.log('Return to main area');
    }




    public deletePlaylist() {

        if (this.state.stepCounter == this.state.songs.length - 1) {  //så man inte hamnar utanför
arrayen/spellistan
            this.setState({ replyText: "OBS, You deleted the LAST song in the playlist!" })
        }


        try {
            fetch('api/Playlists/' + this.state.songs[this.state.stepCounter].id, { method: 'delete' })
                .then(data => {
                    this.setState({
                        songs: this.state.songs.filter((q) => {
                            return (q.id != this.state.songs[this.state.stepCounter].id);
                        })
                    });

                });
        } catch (e) {
            console.log("Error", e)
        }


    }
```

```
  loadSpecificPlaylist() {

    fetch('api/Playlists/GetPlaylists?playlistName=' + this.state.choosenPlaylist)
        .then(response => response.json() as Promise<Song[]>)
        .then(data => {
            console.log("refreshed database from parent")
            this.setState({ songs: data, loading: true });

        })

        .catch(message => {
            console.log('Error' + message);

        });

}




loadUniquePlaylists() {

    fetch('api/Playlists/Unique')
        .then(response => response.json() as Promise<Song[]>)
        .then(data => {
            this.setState({
                playlists: data, loading: true
            });

            this.generateUniqePlaylist();

            console.log("datadump from unique playlists: ", data)
        })

        .catch(message => {
            console.log('Error' + message);

        });

}
```

```
//sorterar ut unika spelliste-namn för array med dessa
generateUniqePlaylist() {
    var Plist = [''];
    var filteredArray = [''];

    // 1. sorterar ut alla spelliste-namn
    for (var i = 0; i < this.state.playlists.length; i++) {

        Plist[i] = this.state.playlists[i].playlistName;

    }


    // 2. filtrerar fram unika spellistenamn
    filteredArray = Plist.filter(function (item, pos) {
        return Plist.indexOf(item) == pos;
    });
    this.setState({ uniquePlaylistNames: filteredArray })

    console.log("uniquePlaylistNames-lista: ", this.state.uniquePlaylistNames)

}




//laddar den valda spellistan från listan över spellistor
loadChoosenPlaylist = (event: any) => {
    this.setState({ choosenPlaylist: event.target.value })
    this.loadSpecificPlaylist();
    this.setState({ stepCounter: 0 })

    this.setState({ isNextBtnHidden: true, isPrevBtnHidden: false, replyText: ''})
 }



//togglar kollaps(collapse)-tabellerna
onCollapsePlaylists = () => {

    this.setState({ collapsePlaylists: !this.state.collapsePlaylists })

    this.loadUniquePlaylists();
```

```
        this.setState({ stepCounter: 0 })
        this.setState({ isNextBtnHidden: true, isPrevBtnHidden: false, replyText: "  })
    }




}  //klass slutar här
```

AddPlaylist.tsx (child 1)

AddPlaylist är den första av två child-komponenter till AdminPlaylist som kunde beskådas ovan. Som namnet antyder så är det en area för att lägga till en låt/spellista. I och med att alla spellistor skapas dynamiskt och existerar endast som en egenskap i databasen över låtarna så börjar en låtlista först att existera när man lägger till den första låten som skall ha denna låtlista. För att sedan separera fram låtarna så filtreras dessa fram. Se mer om detta under Controllers.

Så arean utgörs av en mängd textfält där man fyller egenskaper så som artist, album, låt, fysiskt filnamn (endast själva namnet, url och filextension läggs till automatiskt), och som då har nämnts namnet på antingen en ny låtlista eller en befintlig (tryck ‘p’ för detta). Därefter är det bara att trycka på knappen lägga till och låtens data adderas till databasen.

Värt att nämna är att den fysiska mp3-ljudfilen får kopieras in till undermappen ‘mp3’ till projektmappen och dess namn sedan kopieras in till fältet för fysisk fil som nämndes ovan. Drag and drop och dylikt är framtida features.

```
import * as React from 'react';
import { render } from 'react-dom';
import { Component } from 'react';
import { RouteComponentProps, Redirect } from 'react-router-dom';
import { Song } from './AdminPlaylists';



interface IAddPlaylistSongProps {

    loadSpecificPlaylist: Function;
    choosenPlaylist: string;
    }




interface IAddPlaylistSongState {
    id: Number;
    _artist: string;
    albumTitle: string;
    songTitle: string;
    songFile: string;
    playlistName: string;
    hasFetchedData: boolean;
```

```
    loading: boolean;
    isPropUsed: boolean;
}




export class AddPlaylist extends React.Component<IAddPlaylistSongProps,
IAddPlaylistSongState> {
    public constructor(props: IAddPlaylistSongProps) {
        super(props);
        this.state = {
            id: 0,
            _artist:'',
            albumTitle: '',
            songTitle: '',
            songFile:'',
            playlistName: '',
            hasFetchedData: false,
            loading: false,
            isPropUsed: false




        };
        this.changeArtist = this.changeArtist.bind(this);
        this.changeAlbumTitle = this.changeAlbumTitle.bind(this);
        this.changeSongTitle = this.changeSongTitle.bind(this);
        this.changeSongFile = this.changeSongFile.bind(this);
        this.changePlaylistName = this.changePlaylistName.bind(this);


        this.addPlaylist = this.addPlaylist.bind(this);

        this.runAsyncMethods = this.runAsyncMethods.bind(this);

        this.runOtherMethod = this.runOtherMethod.bind(this);

    }



    runOtherMethod() {
```

```
        this.runAsyncMethods(Function);
    }


    //köra funktionera i rätt ordning så att allt uppdateras som det skall
    runAsyncMethods(callback: any) {

        setTimeout(() => callback(this.addPlaylist()), 0);

        setTimeout(() => callback(this.props.loadSpecificPlaylist()), 500);

    }







    public render() {
        return (
            <div className="list-group">
                <h1 className="artistLabel"><span className="songheadercolor">Add
playlist/song</span></h1>

                <div className="AddSong">
                    <label id="artistlabel"><h3><span
className="songitemcolor">Artist</span></h3></label>

                    <input
                        id="artisttextbox"
                        className="textBox"
                        type="text"
                        placeholder=""
                        value={this.state._artist}
                        onChange={this.changeArtist} />
                </div>

                <div className="Song">
                    <label id="albumlabel"><h3><span
className="songitemcolor">Album</span></h3></label>

                    <input
                        id="albumtextbox"
                        className="textBox"
```

```
                type="text"
                placeholder=""
                value={this.state.albumTitle}
                onChange={this.changeAlbumTitle} />
            </div>

            <div className="Song">
              <label id="titlelabel"><h3><span
className="songitemcolor">Title</span></h3></label>

              <input
                id="titletextbox"
                className="textBox"
                type="text"
                placeholder=""
                value={this.state.songTitle}
                onChange={this.changeSongTitle} />
            </div>


            <div className="Song">
              <label id="filelabel"><h3><span
className="songitemcolor">File</span></h3></label>

              <input
                id="filetextbox"
                className="textBox"
                type="text"
                placeholder=""
                value={this.state.songFile}
                onChange={this.changeSongFile} />
            </div>




            <div className="Song">
              <label id="playlistlabel"><h3><span
className="songitemcolor">Playlist</span></h3></label>
              <label id="playlistlabel2"><h5><span className="songitemcolor">To add
current playlist, press 'p'</span></h5></label>

              <input
                id="playlisttextbox"
                className="textBox"
```

```
                type="text"
                placeholder=""
                value={this.state.playlistName}
                onChange={this.changePlaylistName} />

            </div>




            <button id="btnAddSong" className="btm btn-sucess btn-lg"
onClick={this.runOtherMethod}>Add song to playlist</button>


        </div>
    )
}









//redundant kod ja, men kom inte på ett sätt att kunna använda properties (_question et.al)
som metod-argument

changeArtist(event: any) {

    const regexOnlyAlphanumveric = /[^a-öA-Ö0-9?.()-[]_ ]/g;
    if (event.target.value == "" || regexOnlyAlphanumveric.test(event.target.value)) {
        this.setState({  _artist: '' })
    }
    else if (!regexOnlyAlphanumveric.test(event.target.value)) {
        this.setState({ _artist: event.target.value })
    }

}



changeAlbumTitle(event: any) {
```

```
    const regexOnlyAlphanumveric = /[^a-öA-Ö0-9?.()-[]_ ]/g;
    if (event.target.value == "" || regexOnlyAlphanumveric.test(event.target.value)) {
      this.setState({ albumTitle: '' })
    }
    else if (!regexOnlyAlphanumveric.test(event.target.value)) {
      this.setState({ albumTitle: event.target.value })

    }

}




changeSongTitle(event: any) {
    const regexOnlyAlphanumveric = /[^a-öA-Ö0-9?.()-[]_ ]/g;
    if (event.target.value == "" || regexOnlyAlphanumveric.test(event.target.value)) {
      this.setState({ songTitle: '' })
    }
    else if (!regexOnlyAlphanumveric.test(event.target.value)) {
      this.setState({ songTitle: event.target.value })
    }

}



changeSongFile(event: any) {
    const regexOnlyAlphanumveric = /[^a-öA-Ö0-9?.()-[]_ ]/g;
    if (event.target.value == "" || regexOnlyAlphanumveric.test(event.target.value)) {
      this.setState({ songFile: '' })
    }
    else if (!regexOnlyAlphanumveric.test(event.target.value)) {
      this.setState({ songFile: event.target.value })
    }

}




changePlaylistName(event: any) {
    const regexOnlyAlphanumveric = /[^a-öA-Ö0-9?.()-[]_ ]/g;
    if (event.target.value == 'p') {  //skriv 'p' för att få fram vald spellista
```

```
            this.setState({ playlistName: this.props.choosenPlaylist })

        }
        else if (event.target.value == regexOnlyAlphanumveric.test(event.target.value)) {
            this.setState({ playlistName: '' })
        }
        else if (!regexOnlyAlphanumveric.test(event.target.value)) {
            this.setState({ playlistName: event.target.value })

        }

    }




    addPlaylist() {


        fetch('api/Playlists/CreatePlaylist?_artist=' + this.state._artist + '&albumTitle=' +
this.state.albumTitle + '&songTitle=' + this.state.songTitle + '&songFile=' + this.state.songFile
+ '&playlistName=' + this.state.playlistName)
            .then(response => response.json() as Promise<Song[]>)


            .then(data => {
                console.log("add song fired: ", data);
                this.setState({
                    hasFetchedData: false,
                    _artist: "",
                    albumTitle: "",
                    songTitle: "",
                    songFile: "",
                    playlistName: ""
                })
            })

            .catch(message => { console.log('Error' + message); });


    }
```

```
} //class ends here
```

EditPlaylist.tsx (child 2)

EditPaylist är mycket snarlik AddPlaylist förutom att här så ändrar man innehållet i den aktuella spellistans valda låt. Upplägget är precis samma som för AddPlaylist, det är ett gäng textboxar där de befintliga värdena står och man ändrar sedan dem efter behag. När man är klar så trycker man på en knapp och innehållet ändras i databasen, därefter trycker man på en annan knapp för att komma tillbaka till arean AdminPlaylist där man kan se ändringarna direkt.

Värt att rent programmeringstekniskt notera är väl att de egenskaper skall ändras skickas som props från AdminPlaylist till EditPlaylists där de sedan kopierar över sina states (nuvarande värden) till motsvarande lokala states (states i betydelsen lokala props) eftersom props inte kan ändra sina värden. Detta sker med hjälp av componentWillReceiveProps som är en så kallas livscykel-metod som äger rum på ett mycket tidigt stadium i react-programmet innan något annat görs med nya props, vilket motverkar att ofrivilliga förändringar kan äga rum.

```
import * as React from 'react';
import { render } from 'react-dom';
import { Component, EventHandler } from 'react';
import { RouteComponentProps, Redirect } from 'react-router-dom';
import { Song } from './AdminPlaylists'



interface EditPlaylistSongProps {

    id: number;
    _artist: string;
    albumTitle: string;
    songTitle: string;
    songFile: string;
    playlistName: string;
    songFreq: number;
    loadSpecificPlaylist: Function;
    choosenPlaylist: string;

}




interface EditPlaylistSongState {
```

```
    id: number;
    _artist: string;
    albumTitle: string;
    songTitle: string;
    songFile: string;
    playlistName: string;
    songFreq: number;
    hasFetchedData: boolean;
    loading: boolean

}




export class EditPlaylist extends React.Component<EditPlaylistSongProps,
EditPlaylistSongState> {
    public constructor(props: EditPlaylistSongProps) {
        super(props);
        this.state = {
            id: 0,
            _artist: "",
            albumTitle: "",
            songTitle: "",
            songFile: "",
            playlistName: "",
            songFreq: 0,
            hasFetchedData: false,
            loading: false
        };



        this.changeArtist = this.changeArtist.bind(this);
        this.changeAlbumTitle = this.changeAlbumTitle.bind(this);
        this.changeSongTitle = this.changeSongTitle.bind(this);
        this.changeSongFile = this.changeSongFile.bind(this);

        this.UpdatePlaylist = this.UpdatePlaylist.bind(this);

        this.runAsyncMethods = this.runAsyncMethods.bind(this);
        this.runOtherMethod = this.runOtherMethod.bind(this);

    }
```

```
runOtherMethod() {

    this.runAsyncMethods(Function);
}

//köra funktionera i rätt ordning så att allt uppdateras som det skall
runAsyncMethods(callback: any) {

    setTimeout(() => callback(this.UpdatePlaylist()), 0);

    setTimeout(() => callback(this.props.loadSpecificPlaylist()), 500);

}



 componentWillReceiveProps() {


    this.setState({ id: this.props.id });

    this.setState({ _artist: this.props._artist });

    this.setState({ albumTitle: this.props.albumTitle });

    this.setState({ songTitle: this.props.songTitle });

    this.setState({ songFile: this.props.songFile });

    this.setState({ playlistName: this.props.playlistName });

    this.setState({ songFreq: this.props.songFreq });


}
```

```
    public render() {
        return (
            <div className="list-group">
                <h1 className="songLabel"><span className="songheadercolor">Change
playlist
                    </span></h1>


                <div className="AddSong">
                    <label id="artistlabel"><h3><span className="songitemcolor">Artist</span>
                    </h3></label>

                    <input
                        id="artisttextbox"
                        className="textBox"
                        type="text"
                        placeholder=""
                        value={this.state._artist}
                        onChange={this.changeArtist} />
                </div>

                <div className="Song">
                    <label id="albumlabel"><h3><span
className="songitemcolor">Album</span></h3>
                    </label>

                    <input
                        id="albumtextbox"
                        className="textBox"
                        type="text"
                        placeholder=""
                        value={this.state.albumTitle}
                        onChange={this.changeAlbumTitle} />
                </div>

                <div className="Song">
                    <label id="titlelabel"><h3><span
className="songitemcolor">Title</span></h3>
                    </label>

                    <input
                        id="titletextbox"
                        className="textBox"
```

```
                type="text"
                placeholder=""
                value={this.state.songTitle}
                onChange={this.changeSongTitle} />
            </div>


            <div className="Song">
               <label id="filelabel"><h3><span className="songitemcolor">File</span></h3>
               </label>

               <input
                  id="filetextbox"
                  className="textBox"
                  type="text"
                  placeholder=""
                  value={this.state.songFile}
                  onChange={this.changeSongFile} />
            </div>




            <button id="btnEditSong" className="btm btn-sucess btn-lg"
onClick={this.runOtherMethod}>Change song in playlist</button>



         </div>


      )




   }    //render slutar här
```

```
    //redundant kod ja, men kom inte på ett sätt att kunna använda properties som
metod-argument

    changeArtist(event: any) {

        const regexOnlyAlphanumveric = /[^a-öA-Ö0-9?.()-[]_ ]/g;
        if (event.target.value == "" || regexOnlyAlphanumveric.test(event.target.value)) {
            this.setState({ _artist: '' })
        }
        else if (!regexOnlyAlphanumveric.test(event.target.value)) {
            this.setState({ _artist: event.target.value })
        }

    }



    changeAlbumTitle(event: any) {

        const regexOnlyAlphanumveric = /[^a-öA-Ö0-9?.()-[]_ ]/g;
        if (event.target.value == "" || regexOnlyAlphanumveric.test(event.target.value)) {
            this.setState({ albumTitle: '' })
        }
        else if (!regexOnlyAlphanumveric.test(event.target.value)) {
            this.setState({ albumTitle: event.target.value })

        }

    }




    changeSongTitle(event: any) {
        const regexOnlyAlphanumveric = /[^a-öA-Ö0-9?.()-[]_ ]/g;
        if (event.target.value == "" || regexOnlyAlphanumveric.test(event.target.value)) {
            this.setState({ songTitle: '' })
        }
        else if (!regexOnlyAlphanumveric.test(event.target.value)) {
            this.setState({ songTitle: event.target.value })
        }

    }
```

```
  changeSongFile(event: any) {
    const regexOnlyAlphanumveric = /[^a-öA-Ö0-9?.()-[]_ ]/g;
    if (event.target.value == "" || regexOnlyAlphanumveric.test(event.target.value)) {
      this.setState({ songFile: '' })
    }
    else if (!regexOnlyAlphanumveric.test(event.target.value)) {
      this.setState({ songFile: event.target.value })
    }

  }




  UpdatePlaylist() {

    fetch('api/Playlists/UpdatePlaylist?id=' + this.state.id + '&_artist=' + this.state._artist +
'&albumTitle=' + this.state.albumTitle + '&songTitle=' + this.state.songTitle + '&songFile=' +
this.state.songFile + '&playlistName=' + this.props.choosenPlaylist + '&songFreq=' +
this.state.songFreq)

      .then(Response => Response.json() as Promise<Song>)

      .then(data => {
        console.log("UpdatePlaylist fired: ", data);

        this.setState({
          hasFetchedData: false,
          _artist: "",
          albumTitle: "",
          songTitle: "",
          songFile: "",
          songFreq: 0

        })
      })


      .catch(message => { console.log('Error' + message); });


  }
```

```
}   //klass slutar här
```

Home.tsx

Home-komponentens innehåll ersätts här av MachinePlayer-komponentens. Detta i och med att player-komponenten är den viktigaste i programmet då den utför själva grundsyftet att spela upp musik. Det är ju annars en roll som home, index och liknande brukar ha i program, ett ställe i programmet man återvänder till och är centralt. Detta återgår även i tex komponenten routes.tsx där komponenten Home mappas till just sökvägen till PlayerMachine-komponenten (när denna placeras i Layout-komponenten i det som sedan blir programmets navigations-meny) .

```
import * as React from 'react';
import { RouteComponentProps } from 'react-router';
import { PlayerMachine } from './PlayerMachine';


export class Home extends React.Component<RouteComponentProps<{}>, {}> {




    public render() {

        return <div>
            { /*
            Home komponenten utskrift
            */}



            <PlayerMachine

            />
        </div>;
    }
}
```

Layout.tsx

Sköter om programmets layout, bland annat i form av att importera och placera React-sidans meny-komponent (NavMenu).

```
import * as React from 'react';
import { NavMenu } from './NavMenu';

export interface LayoutProps {
    children?: React.ReactNode;
}

export class Layout extends React.Component<Layout Props, {}> {
    public render() {
        return <div class Name='container-fluid'>
            <div className='row'>
                <div className='col-sm-3'>
                    <NavMenu />
                </div>
                <div className='col-sm-9'>
                    { this.props.children }
                </div>
            </div>
        </div>;
    }
}
```

Nav Menu.tsx

Kopplar ihop länkarna i navigeringsmenyn med de komponenter som skall kunnas nås från denna. I detta program är det följande: Player Machine, Admin Playlist (Add Playlist och Edit Playlist är childs till denna och nås därför inte direkt) samt Top 10.

Enklare uttryckt kan man säga att det här är innehållet i den navigeringsmeny som förekommer på programmets förstasida (och enda sida)..

```
import * as React from 'react';
import { Link, NavLink } from 'react-router-dom';

export class NavMenu extends React.Component<{}, {}> {
    public render() {
        return <div class Name='main-nav'>
                <div className='navbar navbar-inverse'>
                <div className='navbar-header'>
                    <button type='button' className='navbar-toggle' data-toggle='collapse'
data-target='.navbar-collapse'>
                        <span class Name='sr-only'>Toggle navigation</span>
                        <span class Name='icon-bar'></span>
                        <span class Name='icon-bar'></span>
                        <span class Name='icon-bar'></span>
                    </button>
                    <Link className='navbar-brand' to={'/'}>E Juke-Box</Link>
                </div>
                <div className='clearfix'></div>
                <div className='navbar-collapse collapse'>
                    <ul class Name='nav nav bar-nav'>
                        <li>
                            <Nav Link to={ '/Player Machine' } exact active ClassName='active'>
                                <span class Name='glyphicon glyphicon-music'></spain> Player
Machine
                            </NavLink>
                        </li>
                        <li>
                            <Nav Link to={ '/Top10' } active ClassName='active'>
                                <span class Name='glyphicon glyphicon-star-empty'></span> Top 10
Songs
                            </NavLink>
                        </li>
                        <li>
                            <Nav Link to={ '/Admin Playlists' } active ClassName='active'>
                                <span className='glyphicon glyphicon-cog'></span> Admin Playlists
```

```
                    </NavLink>
                  </li>
                </ul>
              </div>
            </div>
          </div>;
    }
}
```

PlayerMachine.tsx

Programmets huvudkomponent (funktionellt viktigaste komponent/del) där spellistor och dess låtar spelas upp. Förutom sedvanliga kontrollers som finns på alla mp3-spelare som play, pause, stop etc så kan man även navigera med hjälp av dynamiska listor. Dessa är tre, den första väljer man spellista med, den andra är den spellistans låtar som man välde i den första dynamiska menyn och den tredje slutligen är ett antal props värden som annars i programmet visas i grafisk form som där istället visas i numeriskt.

Den dynamiska listan som visar aktuell vald spellistas låtar kan man även navigera genom att helt enkelt klicka på den låt man vill skall spelas. Detta synkas med de “vanliga” kontrollerna som då följer med.

Värt är även att nämna den textruta vars rubrik lyder “API-Data”, här hämtas tre egenskaper från DisCogs API (https://www.discogs.com/developers/) gällande den aktuella låten. Dessa är: genre, utgivningsland samt utgivningsår. Eftersom den lokala datan har prioritet och textrutor och andra kontroller för dessa skapades först och layouten redan är trångbodd så fick dessa tre egenskaper husera inom samma textruta med hjälp av kommatecken-separation. Funktionellt ok och påvisar att jag hämtar data från extern data (API) men kanske inte så estetiskt tilltalande, men jag är ju en backend-utvecklare när allt kommer omkring.

```
import * as React from 'react';
import ReactPlayer from 'react-player';
import { Song } from './AdminPlaylists';



interface playerMachineProps {

}


interface playerMachineState {
    url: string[];
    urlIP: string;
    fileExtension: string;
    playing: boolean;
    volume: number;
    muted: boolean;
    played: number;
    loaded: number;
    duration: number;
```

```
    playbackRate: number;
    loop: boolean;
    seeking: boolean;
    playListIndex: number;
    messageText: string;
    maxLengthSongNo: number;
    collapseStateArea: boolean;
    collapsePlaylists: boolean;
    collapsePlaylistSongs: boolean;
    songs: Song[];
    playlists: Song[];
    loading: boolean;
    isNextButtonHidden: boolean;
    isPrevButtonHidden: boolean;
    maxLengthSongTextbox: number;
    playedSongFreq: number;
    hasFetchedData: boolean;
    choosenPlaylist: string;
    uniquePlaylistNames: string[];
    collapsTable1: string;
    collapsTable2: string;
    collapsTable3: string;
    discogsApiResult: string;



};









export class PlayerMachine extends React.Component<playerMachineProps,
playerMachineState> {

    player: any
    instance: any;



    constructor(props: playerMachineProps) {
        super(props);
        this.state = {
```

```
    url: [],
    urlIP: 'http://127.0.0.1:8887/',
    fileExtension: '.mp3',
    playing: false,
    volume: 0.8,
    muted: false,
    played: 0,
    loaded: 0,
    duration: 0,
    playbackRate: 1.0,
    loop: false,
    seeking: false,
    playListIndex: 0,
    messageText: '',
    maxLengthSongNo: 3,
    collapseStateArea: true,
    collapsePlaylists: true,
    collapsePlaylistSongs: true,
    songs: [],
    playlists: [],
    loading: false,
    isNextButtonHidden: false,
    isPrevButtonHidden: true,
    maxLengthSongTextbox: 50,
    playedSongFreq: 0,
    hasFetchedData: false,
    choosenPlaylist: 'BASIC LIST',
    uniquePlaylistNames: [''],
    collapsTable1: 'collapsTable1',
    collapsTable2: 'collapsTable3',
    collapsTable3: 'collapsTable5',
    discogsApiResult: ''


}


this.msToTime = this.msToTime.bind(this);
this.addPlayedSongFreqToDb = this.addPlayedSongFreqToDb.bind(this);

this.loadSpecificPlaylist = this.loadSpecificPlaylist.bind(this);
this.loadUniquePlaylists = this.loadUniquePlaylists.bind(this);

this.loadChoosenPlaylist = this.loadChoosenPlaylist.bind(this);

this.generateUniqePlaylist = this.generateUniqePlaylist.bind(this);
```

```
        //metod som anropar discogs-API
        this.getGenreFromDiscogsAPI = this.getGenreFromDiscogsAPI.bind(this);


        //laddar in unik lista av befintliga spellistor
        this.loadUniquePlaylists();


        //laddar in default spellista
        this.loadSpecificPlaylist();


    }  //konstruktor slutar här




    public render() {

        const { url, urlIP, fileExtension, playing, volume, muted, played, loaded, duration,
playbackRate, loop, playListIndex, messageText, maxLengthSongNo, songs, playlists,
collapseStateArea, isNextButtonHidden, isPrevButtonHidden, maxLengthSongTextbox,
discogsApiResult } = this.state


        let playlist = [''];
        let count = 0;
        let count2 = 0;
        let count3 = 0;


        let tuneNumber = [1];
        let tuneArtist = [''];
        let tuneAlbum = [''];
        let tuneTitle = [''];
        let tunePlaylist = [''];
        let listOfPlaylists = [''];

        let playlistSongs: JSX.Element[];
        playlistSongs = [];

        let result = discogsApiResult.toString();
```

```
        //discogsApiResult;



    //renderar unik spellista med html-tags
    let ulplaylists: JSX.Element[];
    ulplaylists = [];

    for (count2 = 0; count2 <= this.state.uniquePlaylistNames.length - 1; count2++) {
        ulplaylists[count2] = <button key={count2}
value={this.state.uniquePlaylistNames[count2]} onClick={this.loadChoosenPlaylist}
            className="list-group-item list-group-item-action"  >
{this.state.uniquePlaylistNames[count2]}</button >;

    }




    //vald spellistas låtar i en egen lista
    for (count = 0; count <= songs.length - 1; count++) {
        playlistSongs[count] = < button key={count} value={count}  onClick = {
this.loadChoosenPlaylistSong }  className="list-group-item list-group-item-action" >
{songs[count]._artist} - {songs[count].songTitle} </button >;
    }




    //upplägg av spellistorna
    let contents = this.state.loading
        ? this.renderPlaylistTable(ulplaylists, playlistSongs)
        : <p><em>Loading...</em></p>;



    //hämtad data laddas in
    for (count3 = 0; count3 <= songs.length - 1; count3++) {
        playlist[count3] = urlIP + songs[count3].songFile + fileExtension;

        tuneNumber.push(count3 + 2);
        tuneArtist[count3] = songs[count3]._artist;
        tuneAlbum[count3] = songs[count3].albumTitle;
        tuneTitle[count3] = songs[count3].songTitle;
        tunePlaylist[count3] = songs[count3].playlistName;
```

```
}


this.playlistProp = playlist;


//rendering av bla child-komponenter!!
return <div>


<ReactPlayer
        ref={this.ref}
        playing={playing}
        url={playlist[playListIndex]}
        onPlay={this.onPlay}
        onPause={this.onPause}
        volume={volume}
        // controls={true}   inhemska mp3-kontroller för test å jämförelse
        onBuffer={() => console.log('onBuffer')}
        onDuration={this.onDuration}
        onProgress={this.onProgress}
        muted={muted}
        className='react-player'  //styling
        width='100%'
        height='100%'
        onEnded={this.onEnded}
        onStart={() => console.log('onStart')}
        onReady={() => console.log('onReady')}
        loop={loop}

    />




    <table>

        {/* .....Textrutor .....    */}


        <tbody id="psGrp1">
```

```
                <tr>
                    <th>
                        <label id="playlistNmbSonglabel" className="headercolor_textboxes"
htmlFor='playlistNmbSong'>#Song</label>
                    </th>
                    <th>
                        <label id="playlistArtistlabel" className="headercolor_textboxes"
htmlFor='playlistArtist'>Artist</label>
                    </th>
                    <th>
                        <label id="playlistAlbumlabel" className="headercolor_textboxes"
htmlFor='playlistAlbum'>Album</label>
                    </th>
                    <th>
                        <label id="playlistSongTitlelabel" className="headercolor_textboxes"
htmlFor='playlistSongTitle'>Title</label>
                    </th>
                    <th>
                        <label id="playlistNamelabel" className="headercolor_textboxes"
htmlFor='playlistName'>Playlist</label>
                    </th>
                </tr>


                <tr>
                    <td>
                        <input id="playlistNmbSong" type="text"
value={tuneNumber[playListIndex]} maxLength={maxLengthSongNo} readOnly />

                        <span id="fatBackSlash" className="headercolor_textboxes">  /
</span>

                        <input id="playlistNmbSongLength" type="text"
value={tuneNumber.length -1 } maxLength={maxLengthSongNo} readOnly />

                    </td>
                    <td>
                        <input id="playlistArtist" className="playlistTextboxes" type="text"
value={tuneArtist[playListIndex]} maxLength={maxLengthSongTextbox} readOnly />
                    </td>
                    <td>
                        <input id="playlistAlbum" className="playlistTextboxes" type="text"
value={tuneAlbum[playListIndex]} maxLength={maxLengthSongTextbox} readOnly />
                    </td>
                    <td>
```

```
                    <input id="playlistSongTitle" type="text" value={tuneTitle[playListIndex]}
maxLength={maxLengthSongTextbox} readOnly />
                </td>
                <td>
                    <input id="playlistName" className="playlistTextboxes" type="text"
value={tunePlaylist[playListIndex]} maxLength={maxLengthSongTextbox} readOnly />
                </td>
            </tr>


        </tbody>







        {/* ..... Kontroll-knappar .....    */}



        <tbody id="psGrp2">
            <tr>
                <th className="headercolor_textboxes">Controls</th>
                <td>

                    <button id="mutebutton" className="btn-pill btn-sucess btn"
type="button"  onClick={this.toggleMuted}> Mute </button>
                    <button id="backbutton" className="btn-pill btn-sucess btn"
type="button" hidden={isPrevButtonHidden} onClick={this.onBackward}> &#65308;&#65308;
</button>
                    <button id="stopbutton" className="btn-pill btn-sucess btn"
type="button" onClick={this.stop}> Stop </button>
                    <button id="playpausebutton" className="btn-pill btn-sucess btn"
type="button" onClick={this.playPause}>{playing ? ' Pause ' : ' Play '}</button>
                    <button id="nexxtbutton" className="btn-pill btn-sucess btn"
type="button" hidden={isNextButtonHidden} onClick={this.onForward}>  &#65310;&#65310;
</button>

                </td>

            </tr>
        </tbody>
```

```
            <tbody id="psGrp3">

                <tr>

                   <th className="headercolor_textboxes">API-Data</th>

                <td className="headercolor_textboxes ctrlpos2">
                   <input id="apiData" type="text" value={result}
maxLength={maxLengthSongTextbox} readOnly />
                   </td>

                </tr>


                <tr>
                   <th className="headercolor_textboxes">Elapsed</th>

                  <td className="headercolor_textboxes ctrlpos2">

                      {/*  visar hur länge låten har spelats */}

                      {this.msToTime((duration * played * 1000).toFixed(0))}

                   </td>
                </tr>



                <tr>
                   <th className="headercolor_textboxes">Duration</th>
                <td className="headercolor_textboxes ctrlpos2">

                      {/*  visar låtens längd */}

                      {this.msToTime((duration * 1000).toFixed(0))}

                   </td>

                </tr>
```

```
                <tr>
                <th className="headercolor_textboxes">Seek</th>
                <td>
                     <input id="seek" className="ctrlpos"
                         type='range' min={0} max={1} step='any'
                         value={played}
                         onMouseDown={this.onSeekMouseDown}
                         onChange={this.onSeekChange}
                         onMouseUp={this.onSeekMouseUp}
                      />
                   </td>
                </tr>




                <tr>
                <th className="headercolor_textboxes">Volume</th>
                <td>
                   <input id="volume" className="ctrlpos" type='range' min={0} max={1}
step='any' value={volume} onChange={this.setVolume} />
                   </td>
                </tr>



                <tr>
                   <th className="headercolor_textboxes">
                      <label htmlFor='loop'>Loop</label>
                   </th>
                   <td>
                   <input id='loop' className="ctrlpos" type='checkbox' checked={loop}
onChange={this.toggleLoop} />
                   </td>
                </tr>




                <tr>
                   <th className="headercolor_textboxes">
```

```
                    <label htmlFor='messageText'>Message</label>
                  </th>
                  <td>

                  <h4 className='messageText ctrlpos3'>{messageText}</h4>
                  </td>
                </tr>



                <tr>
                  <th className="headercolor_textboxes">Played</th>
                  <td className="ctrlpos"><progress max={1} value={played} /></td>
                </tr>



                <tr>
                  <th className="headercolor_textboxes">Loaded</th>
                  <td className="ctrlpos"><progress max={1} value={loaded} /></td>
                </tr>

            </tbody>



        </table >




        {/*  placering av spellistorna     */}
        {contents}



    </div>;

}
```

```
//spellistor, dess låtar, states, allt i dragspelsstuk :-P
public renderPlaylistTable(ulplaylists: JSX.Element[], playlistSongs: JSX.Element[]) {

 return <div>

   <table>

      <tbody id="psGrp4">

         <tr>

            {/*   Dragspel. del 1: spelliste-listan   */}

            <td>
              <button className={this.state.collapsTable1}
onClick={this.onCollapsePlaylists}><b>Playlists</b></button>

            <div hidden={this.state.collapsePlaylists}>

              <div className="panel panel-default">

                 <div className="panel-body">

                   <div className="row">

                     <div className="list-group list-group-tf">
                       {ulplaylists}
                     </div>
                   </div>
                 </div>

            </div>
```

```
                    </div>
                </td>

            </tr>



        <tr>
                {/*   Dragspel. del 2:  den ovan valda spellistans låtar   */}

            <td>

                <button className={this.state.collapsTable2}
onClick={this.onCollapsePlaylistSongs}><b>Playlist Songs</b></button>


                <div hidden={this.state.collapsePlaylistSongs}>


                <div className="panel panel-default">



                    <div className="panel-body">

                        <div className="row">

                            <div className="list-group list-group-tf2">
                                {playlistSongs}
                            </div>
                        </div>
                    </div>




                </div>
            </div>

        </td>

    </tr>



     <tr>
```

```
                {/*   Dragspel. del 3: xtra fakta om vissa av elementens data aka states */}


                <td>
                    <button className={this.state.collapsTable3}
onClick={this.onCollapseStateArea}><b>State</b></button>


                    <div hidden={this.state.collapseStateArea}>


                    <div className="panel panel-default">



                      <div className="panel-body">

                        <div className="row">

                          <div className="list-group list-group-tf">

                            <table><tbody>


                              <tr>
                                <th>Volume</th>
                                <td>{this.state.volume.toFixed(3)}</td>
                              </tr>


                              <tr>
                                <th>Played</th>
                                <td>{this.state.played.toFixed(3)}</td>
                              </tr>


                              <tr>
                                <th>Loaded</th>
                                <td>{this.state.loaded.toFixed(3)}</td>
                              </tr>

                              <tr>
                                <th>Remaining</th>
```

```
                                        <td> {this.msToTime((this.state.duration * ((1 -
this.state.played) * 1000)).toFixed(0))} </td>

                                    </tr>


                                </tbody></table>


                            </div>
                        </div>
                    </div>




                </div>
            </div>


                </td>

            </tr>


        </tbody>


    </table>




    </div>;  //renderPlaylistTable - return slutar här

}



  //eventhandlers(properties) och metoder
```

```
load = (url: any) => {
    this.setState({
        url,
        played: 0,
        loaded: 0,

    })
}


onSeekChange = (e: any) => {
    this.setState({ played: parseFloat(e.target.value) })
}


onSeekMouseDown = (e: any) => {
    this.setState({ seeking: true })
}


onSeekMouseUp = (e: any) => {
    this.setState({ seeking: false })
    this.player.seekTo(parseFloat(e.target.value))
}


onProgress = (state: any) => {
    console.log('onProgress', state)

    if (!this.state.seeking) {
        this.setState(state)
    }
}




onDuration = (duration: any) => {
    console.log('onDuration', duration)
    this.setState({ duration })
}

setVolume = (e: any) => {
    console.log('setVolume')
```

```
    this.setState({ volume: parseFloat(e.target.value) })
    console.log(parseFloat(e.target.value))
  }

  onPlay = () => {
    console.log('onPlay')
    this.setState({ playing: true, playedSongFreq:
this.state.songs[this.state.playListIndex].songFreq + 1 });


    //anropar metods som skall hämta aktuell låts genfre från discogs-API:et
    this.getGenreFromDiscogsAPI(this.state.songs[this.state.playListIndex]._artist + " " +
this.state.songs[this.state.playListIndex].songTitle);


    //ökar aktuell låts antal spelade gånger med ett
    this.addPlayedSongFreqToDb(this.state.playedSongFreq);

    console.log('PlayState:', this.state.playing);
  }


  stop = () => {
    console.log('stop')
    this.setState({ url: [], playing: false })
    console.log('PlayState:', this.state.playing)

    this.setState({ messageText: '' })
  }

  playPause = () => {
    console.log('playPause')
    this.setState({ playing: !this.state.playing })
  }


  onPause = () => {
    console.log('onPause')
    this.setState({ playing: false })
    console.log('PlayState:', this.state.playing)
  }


  toggleMuted = () => {
    this.setState({ muted: !this.state.muted })
  }
```

```
ref = (player: any) => {
    this.player = player

}


    msToTime(milliseconds: any) {

    //Erhåll timmar från ms
    let hours = milliseconds / (1000 * 60 * 60);
    let absoluteHours = Math.floor(hours);
    let h = absoluteHours > 9 ? absoluteHours : '0' + absoluteHours;

    //Erhåll resten från timmar och konvertera till min
    let minutes = (hours - absoluteHours) * 60;
    let absoluteMinutes = Math.floor(minutes);
    let m = absoluteMinutes > 9 ? absoluteMinutes : '0' + absoluteMinutes;

    //Erhåll resten från minuter och konvertera till sek
    let seconds = (minutes - absoluteMinutes) * 60;
    let absoluteSeconds = Math.floor(seconds);
    let s = absoluteSeconds > 9 ? absoluteSeconds : '0' + absoluteSeconds;


    return h + ':' + m + ':' + s;

}



onEnded = () => {

    (this.state.playListIndex == this.playlistProp.length - 1 ? this.setState({
        playing: false, isNextButtonHidden: true, isPrevButtonHidden: false
    }) :
        [
            this.setState({
                playListIndex: this.state.playListIndex + 1, playing: this.state.loop,
                isNextButtonHidden: false, isPrevButtonHidden: false }),
            this.setState({ playing: true })
        ]
    );
    console.log('onEnded: ', this.state.playListIndex)
}
```

```
  toggleLoop = () => {
    this.setState({ loop: !this.state.loop })
  }



  onForward = () => {

    (this.state.playListIndex == this.playlistProp.length - 1 ? [console.log('WARNING: THE
END OF THE PLAYLIST!!'),
    this.setState({ messageText: 'END OF PLAYLIST!!', isNextButtonHidden: true,
isPrevButtonHidden: false })]
        :
        [this.setState({ playListIndex: this.state.playListIndex + 1 }),
        this.setState({ messageText: '', isNextButtonHidden: false, isPrevButtonHidden: false
}),
        console.log('+1 song')
        ]
    );
  }



  onBackward = () => {

    (this.state.playListIndex == 0 ? [console.log('WARNING: THE BEGINNING OF THE
PLAYLIST!!'),
    this.setState({ messageText: 'BEGINNING OF PLAYLIST!!', isNextButtonHidden: false,
isPrevButtonHidden: true })]
        :
        [this.setState({ playListIndex: this.state.playListIndex - 1 }),
        this.setState({ messageText: '', isNextButtonHidden: false, isPrevButtonHidden: false
}),
        console.log('-1 song')

        ]
    );
```

```
}

//props för playlists
playlistProp = [''];



onCollapseStateArea = () => {

    this.setState({ collapseStateArea: !this.state.collapseStateArea })


    //fixar togglingen mellan tecknen uppåtpil å nedåtpil
    if (this.state.collapsTable3 == 'collapsTable5') {
        this.setState({
            collapsTable3: 'collapsTable6'
        });
    } else {
        this.setState({
            collapsTable3: 'collapsTable5'
        });
    }



    console.log("toggled States")


}


onCollapsePlaylists = () => {

    this.setState({ collapsePlaylists: !this.state.collapsePlaylists })

    this.loadUniquePlaylists();


    //fixar togglingen mellan tecknen uppåtpil å nedåtpil
    if (this.state.collapsTable1 == 'collapsTable1') {
        this.setState({
            collapsTable1: 'collapsTable2'
        });
    } else {
        this.setState({
```

```
            collapsTable1: 'collapsTable1'
        });
    }



    console.log("toggled playlist")
}



onCollapsePlaylistSongs = () => {

    this.setState({ collapsePlaylistSongs: !this.state.collapsePlaylistSongs })


    //fixar togglingen mellan tecknen uppåtpil å nedåtpil
    if (this.state.collapsTable2 == 'collapsTable3') {
        this.setState({
            collapsTable2: 'collapsTable4'
        });
    } else {
        this.setState({
            collapsTable2: 'collapsTable3'
        });
    }


    console.log("toggled playlistsongs")

}









addPlayedSongFreqToDb(songFreq: number) {


    fetch('api/Playlists/UpdatePlaylist?id=' + this.state.songs[this.state.playListIndex].id +
'&_artist=' + this.state.songs[this.state.playListIndex]._artist + '&albumTitle=' +
```

```
this.state.songs[this.state.playListIndex].albumTitle + '&songTitle=' +
this.state.songs[this.state.playListIndex].songTitle + '&songFile=' +
this.state.songs[this.state.playListIndex].songFile + '&playlistName=' +
this.state.songs[this.state.playListIndex].playlistName + '&songFreq=' + songFreq)

        .then(Response => Response.json() as Promise<Song>)

        .then(data => {
            console.log("addPlayedSongFreqToDb has fired: ", data);
            console.log("songFreq: ", songFreq )

            this.setState({
                hasFetchedData: false,
                playedSongFreq: 0

            })


            this.loadSpecificPlaylist();

        })


        .catch(message => { console.log('Error' + message); });
    }









  loadSpecificPlaylist() {


    fetch('api/Playlists/GetPlaylists?playlistName=' + this.state.choosenPlaylist)
        .then(response => response.json() as Promise<Song[]>)
        .then(data => {
            this.setState({ songs: data, loading: true });
            console.log("load specified: ", data)

        })

        .catch(message => {
            console.log('Error' + message);
```

```
        });

    }



    loadUniquePlaylists() {

        fetch('api/Playlists/Unique')
            .then(response => response.json() as Promise<Song[]>)
            .then(data => {
                this.setState({ playlists: data, loading: true });

                this.generateUniqePlaylist();

                console.log("datadump from unique playlists: ", data)
            })

            .catch(message => {
                console.log('Error' + message);

            });

    }







    getGenreFromDiscogsAPI(artistandsong: string) {

        fetch('https://api.discogs.com/database/search?q=' + artistandsong +
'&page=1&per_page=5&token=bfocOKYafOHcCoPEalDRKoJiOEeZoPwtZwqpCxsI')
            .then(response => response.json() as Promise<any>)
            .then(data => {
                let genre = data.results[0].style[0];

                let country = data.results[0].country;

                let year = data.results[0].year;

               let apidata = genre + ", " + country + ", " + year;
```

```
            this.setState({ discogsApiResult: apidata,  loading: true });

            console.log("datadump from discogs API: ", this.state.discogsApiResult)

        })

        .catch(message => {
            console.log('Error' + message);

        });

}




//sorterar ut unika spelliste-namnn för array med dessa
generateUniqePlaylist() {
    var Plist = [''];
    var filteredArray = [''];

    // 1. sorterar ut alla spelliste-namn
    for (var i = 0; i < this.state.playlists.length; i++) {

        Plist[i] = this.state.playlists[i].playlistName;

    }


    // 2. filtrerar fram unika spellistenamn
     filteredArray = Plist.filter(function (item, pos) {
        return Plist.indexOf(item) == pos;
    });
    this.setState({ uniquePlaylistNames: filteredArray })

    console.log("uniquePlaylistNames-lista: ", this.state.uniquePlaylistNames)

}
```

```
    //laddar den valda spellistan från listan över spellistor
    loadChoosenPlaylist = (event: any) => {
        this.setState({ choosenPlaylist: event.target.value })
        this.loadSpecificPlaylist();
        this.setState({ playListIndex: 0 });
        this.setState({ isNextButtonHidden: false, isPrevButtonHidden: true })

    }




    //laddar vald sång i aktuell spellista
    loadChoosenPlaylistSong = (event: any) => {

        this.setState({ playListIndex: parseInt(event.target.value) });

        this.onPlay();


    }




}   //klass slutar här
```

Top10.tsx

En tio-i-topp-lista över de tio mest spelade låtarna i de förekommande spellistorna. Tabellen sorteras först enligt hur många gånger låtarna är spelade i fallande ordning och sedan som en andrasortering i de fall det förekommer låtar med samma frekvens så sorteras de stigande i bokstavsordning. Mer data förekommer också i tabellen som exempelvis artist, album och spellistenamn..

```
import * as React from 'react';
import { RouteComponentProps, Redirect } from 'react-router';
import 'isomorphic-fetch';
import { Song } from './AdminPlaylists'


interface Top10Props {
}

interface Top10State {
    loading: boolean;
    songs: Song[];

}




export class Top10 extends React.Component<RouteComponentProps<{}>,  Top10State> {
    constructor() {
        super();
        this.state = {
            loading: false,
            songs: [],

        };


        fetch('api/Playlists/GetSongsFreq')
            .then(response => response.json() as Promise<Song[]>)
            .then(data => {
                this.setState({ songs: data, loading: true });
            });

    }
```

```
public render() {


    let contents = this.state.loading
        ? this.renderSongFreqTable(this.state.songs)
        : <p><em>Loading...</em></p>;


    return <div> {contents} </div>;
}

public renderSongFreqTable(songs: Song[]) {

    return <table className="table table-striped">
        <thead>
            <tr>
                <th></th>
                <th className="top10headers" scope="col">Freq</th>
                <th className="top10headers" scope="col">Title</th>
                <th className="top10headers" scope="col">Artist</th>
                <th className="top10headers" scope="col">Album</th>
                <th className="top10headers" scope="col">Playlist</th>

            </tr>
        </thead>

        <tbody>
            {songs.map(item =>
                <tr key={item.id}>
                    <td></td>
                    <td>{item.songFreq}</td>
                    <td>{item.songTitle}</td>
                    <td>{item._artist}</td>
                    <td>{item.albumTitle}</td>
                    <td>{item.playlistName}</td>
                </tr>
            )}
        </tbody>
    </table>
}
```

```
}
```

boot.tsx

De fetmarkerade kommentarerna nedan förklarar egentligen bra vad denna komponent gör men skall ändå kortfattat för sakens skull försöka mig på en egen förklaring här:

Den här komponenten startar React-programmet och är också var AppCointainer skapas och router-komponenten placeras. Filen innehåller även anropet för att rendera React-programmet i rätt HTML-element. Renderar även innehållet i routes.tsx-komponenten. Programmet laddas sedan in (injiceras) i ett DOM-element.

“Allow Hot Module Replacement” nedan innebär ungefär följande; React Hot Loader är en plugin som tillåter komponenter att bli så kallat “live loaded” det vill säga omladdade/reboot-ade utan att de förlorar state och därmed kan man fortsätta jobba med dem med rätt värden. Att förlora state är annars ett av de stora gisslen med React.

```
import './css/site.css';
import 'bootstrap';
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { AppContainer } from 'react-hot-loader';
import { BrowserRouter } from 'react-router-dom';
import * as RoutesModule from './routes';
let routes = RoutesModule.routes;

function renderApp() {
    // This code starts up the React app when it runs in a browser. It sets up the routing
    // configuration and injects the app into a DOM element.
    const baseUrl = document.getElementsByTagName('base')[0].getAttribute('href')!;
    ReactDOM.render(
        <AppContainer>
            <BrowserRouter children={ routes } basename={ baseUrl } />
        </AppContainer>,
        document.getElementById('react-app')
    );
}

renderApp();

// Allow Hot Module Replacement
if (module.hot) {
    module.hot.accept('./routes', () => {
        routes = require<typeof RoutesModule>('./routes').routes;
        renderApp();
    });
}
```

routes.tsx

Routing-komponent importerar komponenter från där de förekommer i projektmappen.

Den mappar även de komponenter som man kan direkt komma åt från webbläsaren med motsvarande komponent från de importerade komponenterna.Gör så genom att (längst ner på denna sida) lägga in dem i Layout-komponenten, som sedan i sin tur figurerar i den meny som finns på huvudsidan.

```
import * as React from 'react';
import { Route } from 'react-router-dom';
import { Layout } from './components/Layout';
import { Home } from './components/Home';
import { FetchData } from './components/FetchData';
import { Counter } from './components/Counter';

import { PlayerMachine } from './components/PlayerMachine';
import { AdminPlaylists } from './components/AdminPlaylists';
import { AddPlaylist } from './components/AddPlaylist';
import { EditPlaylist } from './components/EditPlaylist';
import { Top10 } from './components/Top10';


export const routes = <Layout>
    <Route exact path='/PlayerMachine' component={Home} />
    <Route path='/Top10' component={ Top10 } />
    <Route path='/AdminPlaylists' component={ AdminPlaylists } />
</Layout>;
```

.NET Backend

controllers

PlaylistsController.cs

Själva React-programmets kontroller som agerar som en länk mellan React som frontend och .NET backend (som utgörs av bland annat modellklass, databas-kontext m.m) och databasen.

Det är här CRUD-operationerna utförs rent konkret mot databaserna, där exempelvis olika slags dynamiska spellistor sorteras ut (eller början till sortering som sedan slutförs i aktuell komponent). Detta sker under read-avdelningen. Sedan finns det även övrig CRUD-funktionalitet som Create (Add), Update och Delete. Alla som sagt med tillhörande databas-kommandon och övrig mottagning/sändning av data.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using EJBox.Data;
using EJBox.Data.Models;
using Microsoft.EntityFrameworkCore.Internal;


namespace EJBox.Controllers
{
    [Produces("application/json")]
    [Route("api/Playlists")]
    public class PlaylistsController : Controller
    {

        private readonly EJBoxContext _context;

        public PlaylistsController(EJBoxContext context)
        {
            _context = context;
        }
```

```
// Läs (Read) allt
// GET: api/Playlists
[HttpGet]
public IEnumerable<Song> GetSongs()
{
    return _context.Songs;
}




//Läs (Read) data från specific spellista
//GET: /GetPlaylist/playlistName
[Route("GetPlaylists")]
[HttpGet("{playlistName}")]
public IEnumerable<Song> GetPlaylists(string playlistName)
{

    var a =_context.Songs.Where(p => p.PlaylistName == playlistName);

    return a;
}




//Läs (Read)  UNIK playlistnames
//GET: /GetPlaylistsUnique
[Route("Unique")]
[HttpGet]
public IEnumerable<Song> Unique()
{

    return _context.Songs;

}




// Läs (Read) allt + filtrera efter låtens spel-frekvens (hur många ggr den har spelats)
// GET: /GetSongsFreq
```

```
        [Route("GetSongsFreq")]
        [HttpGet]
        public IEnumerable<Song> GetSongsFreq()
        {
            return _context.Songs.OrderBy( s => s.SongTitle).OrderByDescending(s =>
s.SongFreq).Take(10);

        }




        //Resterande CRUD:


        //SKapa ny (Add)
        // GET: /CreatePlaylist
        [Route("CreatePlaylist")]
        [HttpGet]
        public string CreatePlaylist(string _artist, string albumTitle, string songTitle, string
songFile, string playlistName)
        {

          Song CreatePlaylist = new Song
          {
              _artist = _artist,
              AlbumTitle = albumTitle,
              SongTitle = songTitle,
              SongFile = songFile,
              PlaylistName = playlistName

           };

        _context.Songs.Add(CreatePlaylist);
        _context.SaveChanges();


          return "Song added";
       }




        //Uppdatera (Update)
```

```
        [HttpGet("UpdatePlaylist")]
        public async Task<IActionResult> UpdatePlaylist(int id, string _artist, string albumTitle,
string songTitle, string songFile, string playlistName, int songFreq)
        {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);


            var song = await _context.Songs.SingleOrDefaultAsync(p => p.Id == id);
            if (song == null)
                return BadRequest();

            song.Id = id;
            song._artist = _artist;
            song.AlbumTitle = albumTitle;
            song.SongTitle = songTitle;
            song.SongFile = songFile;
            song.PlaylistName = playlistName;
            song.SongFreq = songFreq;

            _context.Songs.Update(song);
            await _context.SaveChangesAsync();

            return Ok(song);
        }




        //Ta bort (Delete)
        // Delete: api/Playlists/5
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeletePlaylist([FromRoute] int id)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var song = await _context.Songs.SingleOrDefaultAsync(s => s.Id == id);
            if (song == null)
            {
                return NotFound();
            }

            _context.Songs.Remove(song);
```

```
            await _context.SaveChangesAsync();

            return Ok(song);
        }




        //Tom spellista??
        private bool PlaylistExists(int id)
        {
            return _context.Songs.Any(s => s.Id == id);
        }




    }

}
```

Modeller

Song.cs (modellklass)

Detta är det som brukar omnämnas som modellklassen. Alltså en klass med tillhörande egenskaper och annoteringar som visar hur motsvarande tabell i databasen skall se ut. Överföringen till databasen sker genom något som kallas för migration. Denna modellklass och dess motsvarande tabell i databasen måste exakt överensstämma annars får man felmeddelanden när man försöker migrera.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace EJBox.Data.Models
{
    public class Song
    {
        [Key]
        public int Id { get; set; }

        [Required]
        public string _artist { get; set; }

        public string AlbumTitle { get; set; }

        [Required]
        public string SongTitle { get; set; }

        [Required]
        public string SongFile { get; set; }

        [Required]
        public string PlaylistName { get; set; }

        [DefaultValue(0)]
        public int SongFreq { get; set; }
    }
}
```

Databas

appsettings.json (connection-string för databas)

Inte så mycket att säga om gällande det här filen. Det är helt enkelt här databas-strängen för anslutning mot databasen huserar.

```
{
  "ConnectionStrings": {
    "EJBCalldB": "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=EJB;Integrated
Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiS
ubnetFailover=False"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  }
}
```

Databas-kontext

EJBoxContext.cs

Databas-kontexten koordinerar Entity Framework-funktionaliteten för en given modellklass. Man skapar denna klass genom att derivera från System.Data.Entity.DbContext.

Databas-kontexten är även vidare där som man anger vilka modellklasser (se ovan) som skall motsvaras av vilka tabeller i databasen och ser till att dessa kopplingar byggs upp. Kontexten möjliggör även hantering av databas så man kan exempelvis ändra och lägga till i denna (CRUD-operationer).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using EJBox.Data.Models;

namespace EJBox.Data
{
   public class EJBoxContext: DbContext
   {
      public EJBoxContext(DbContextOptions<EJBoxContext> options) : base(options) { }

      public DbSet<Song> Songs { get; set; }

      protected override void OnModelCreating(ModelBuilder modelBuilder)
      {
         var questionsModel = modelBuilder.Entity<Song>();

         questionsModel.ToTable("Songs");

         base.OnModelCreating(modelBuilder);


      }
  }

}
```

Seed-fil

SeedSongs.cs

Seed-filen innehåller helt enkelt den defaultdata som man vill skall finnas med när man för första gången kör programmet och läggs då till i databasen och ger den dess initiala/första värden. Väldigt praktiskt vid utveckling där man gärna vill att samma data skall kunna återkomma i oskadd form gång efter gång vid diverse utprövanden.

Funktionellt så ser den först om det finns någon tidigare data, om så är fallet avslutas klassen redan här, om inte så fyller den databasen med den data som har angivits i seed-filen. För mitt program så är det tre stycken låtar med tillhörande data såsom låtlista som i sin tur pekar på tre fejk-mp3-filer.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using EJBox.Data;
using EJBox.Data.Models;

namespace EJ_Box
{
   public class SeedSongs
   {

      public static void Initialize(IServiceProvider serviceProvider)
      {
         using (var context = new EJBoxContext(
            serviceProvider.GetRequiredService<DbContextOptions<EJBoxContext>>()))
         {
            // Leta efter vilken låt som helst (any song).
            if (context.Songs.Any())
            {
               return;   // DB har seed:ats
            }

            context.Songs.AddRange(
               new Song
               {
                  _artist = "Basic Artist A",
```

```
                AlbumTitle = "Basic Album A",
                SongTitle = "Basic Title A",
                SongFile = "Basic File A",
                PlaylistName = "BASIC LIST",
                SongFreq = 0
            },
            new Song
            {
                _artist = "Basic Artist B",
                AlbumTitle = "Basic Album B",
                SongTitle = "Basic Title B",
                SongFile = "Basic File B",
                PlaylistName = "BASIC LIST",
                SongFreq = 0
            },
            new Song
            {
                _artist = "Basic Artist C",
                AlbumTitle = "Basic Album C",
                SongTitle = "Basic Title C",
                SongFile = "Basic File C",
                PlaylistName = "BASIC LIST",
                SongFreq = 0
            }



        );
            context.SaveChanges();



        }
    }



} //klass slutar här
}
```

Program.cs

Ett ASP.NET Core program (MVC i vårt fall) bygger en värd, “host”, vid uppstarten av programmet. Värden är ett objekt som inkapslar alla programresurser, såsom:

- En HTTP server implementation
- Middleware komponenter
- Loggning
- DI (Direct Injection)
- Konfiguration
- Tjänster

Den huvudsakliga anledningen för att inkluder programmets resurser, som är beroende av varandra, i ett objekt är så kallad “lifetime management”-kontroll över programmets uppstart och att det avslutas på ett lämpligt sätt.

Koden för att skapa en värd (“host”) följer det så kallade “builder”-mönstret (“pattern”). Vidare så är metoder anropade för att konfigurera varje resurs som utgör en del av värden ifråga. En builder-metod anropas för att föra allt samman och initiera värd-objektet.

ASP.NET Core 2x använder Web Host (WebHost klassen) för webb-appar. Ramverket förser CreateDefaultBuilder-metoden för att en värd med vanligt använda egenskaper, såsom:

- Ladda in konfigurering från *appsettings.json*, *appsettings.{Environment Name}.json*, miljövariabler, kommandotolks-argument i konsol, och andra konfigurations källor
- Sända output från loggning till konsolen och och “debug providers”.

Slutligen och i vårt fall så är den resurs som skapas enligt ovan en tjänst för att använda vår seed-klass som en tjänst.

(kod på nästa sida)

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using EJBox.Data.Models;

namespace EJ_Box
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = BuildWebHost(args);

            using (var scope = host.Services.CreateScope())
            {
                var services = scope.ServiceProvider;

                try
                {
                    //Måste använda Namespace.Models;
                    SeedSongs.Initialize(services);
                }
                catch (Exception ex)
                {
                    var logger = services.GetRequiredService<ILogger<Program>>();
                    logger.LogError(ex, "An error occurred seeding the DB.");
                }
            }



            host.Run();

            //BuildWebHost(args).Run();
        }

        public static IWebHost BuildWebHost(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
```

```
            .UseStartup<Startup>()
            .Build();
    }
}
```

Layout

site.css

(Samt Bootstrap som är lokalt installerat i form av medföljande npm-paket. Se även förteckning över dessa under “Övrigt”)

Beträffande css-filen så hanterar den tillsammans med Bootstrap all layout, grafisk formatering, positionering, responsiv design osv.

```
/*
.textBox{

}
*/




body {
    background-color: #3333ff;
    color:black;
    overflow-x: hidden;
    overflow-y: hidden;

}



.main-nav li .glyphicon {
    margin-right: 10px;
}

/* Highlighting rules for nav menu items */
.main-nav li a.active,
.main-nav li a.active:hover,
.main-nav li a.active:focus {
    /*background-color: #4189C7;  */
    background-color: white;
    color: #3333ff;
}

/* Keep the nav menu independent of scrolling and on top of other items */
```

```
.main-nav {
    position: fixed;
    top: 0;
    left: 0;
    right: 0;
    z-index: 1;
}



/* ..... */
/* ..... */



/* Eget css */



/*  Generellt - för olika komponenter   */



/*  generella  header-färger för olika komponenter   */

.songheadercolor {
    /*background-color: white;*/
    /*
    background-color: #d9d9d9;
    color: #3333ff;
    */
    color: white;
}



/* ..... */
/* ..... */
```

```
/*   PlayerMachine - komponenten   */



/*  Positionsgrupper för att underlätta positionering av olika grupper av objekt   */


#psGrp1 {
    position: absolute;
    left: 3em;
    top: 1em;
}



#psGrp2 {
    position: absolute;
    left: 3em;
    top: 7em;
}


#psGrp3 {
    position: absolute;
    left: 3em;
    top: 11em;
}



#psGrp4 {
    position: absolute;
    left: 3em;
    top: 29em;
}




.messageText { /*  även för admin-komponent*/
    color: red;
    /* font-size: 10pt; */
}

.headercolor_textboxes {
```

```
    color: white;
    font-size: large;

}



/*  labels till texboxar nedan    */

#playlistNmbSonglabel {
    position: relative;
    left: 0em;

}


#playlistArtistlabel {
    position: absolute;
    left: 6.5em;
    top: 0.1em;

}


#playlistAlbumlabel {
    position: absolute;
    left: 18em;
    top: 0.1em;
}


#playlistSongTitlelabel {
    position: absolute;
    left: 29.5em;
    top: 0.1em;
}




#playlistNamelabel {
    position: absolute;
    left: 41.4em;
    top: 0.1em;
}
```

```
/*  texboxar */

#playlistNmbSong {
    position: absolute;
    width: 2em;
    left: 1em;
    top: 2em;

}



#playlistNmbSongLength {
    position: absolute;
    width: 2em;
    left: 4em;
    top: 2em;

}


#playlistArtist {
    width: 15em;
    position: absolute;
    left: 8em;
    top: 2em;
}


#playlistAlbum {
    width: 15em;
    position: absolute;
    left: 23em;
    top: 2em;
}



#playlistSongTitle {
    width: 15em;
    position: absolute;
```

```
    left: 38em;
    top: 2em;
}



#playlistName {
    width: 10em;
    position: absolute;
    left: 53em;
    top: 2em;
}



#apiData{
    height:1.4em;
    color:black;
    font-size: 0.85em;
}



#fatBackSlash {
    font-size: 1.7em;
    color: white;
    position: relative;
    left: 1.9em;
}


.playlistTextboxes {
    width: 10em;
}




/*  kontroll-knappar (play, stop osv    */



/*
.controlbuttons {
    width: 3.6em;
```

```
}
   */



#mutebutton {
   position: relative;
   left: 3em;

}



#backbutton {
   position: relative;
   left: 3.5em;

}



#stopbutton {
   position: relative;
   left: 4em;

}




#playpausebutton {
   position: relative;
   left: 4.5em;
   width: 5em;
}



#nexxtbutton {
   position: relative;
   left: 5em;

}
```

```
#seek {
    width: 17em;
}

#volume{
    width: 17em;
}



/* positionering av kontrollreglage odyl fr.o.m elapsed till loaded  */
.ctrlpos {
    position: relative;
    left: 2.5em;
}


/*  för elapsed och duration  */
.ctrlpos2 {
    position: relative;
    left: 2.2em;
}


/*  för message  */
.ctrlpos3 {
    position: relative;
    left: 2em;
}



/*  dynamiskt genererade list-group element i playlist-listan */
.list-group-tf .list-group-item {
    background-color: #d9d9d9;
    color: #000000;
    border: 1px solid #000000;
    width: 10em;
    height: 1.8em;
    font-weight: bold;
    border-radius: 12px;
    text-align: center;
    vertical-align: auto;
    text-decoration: none;
```

```
    font-size: 16px;
    margin: 4px 2px;
}



    .list-group-tf .list-group-item:hover {
        background-color: #999999;
        color: red;
        border: 1px solid #000000;
        width: 10em;
        height: 1.8em;
        font-weight: bold;
        border-radius: 12px;

        text-align: center;
        vertical-align: auto;
        text-decoration: none;

        font-size: 16px;
        margin: 4px 2px;
    }




.list-group-tf2 .list-group-item {
    width: 20em;
    height: 2.4em;
    font-weight: bold;
    background-color: white;
}




/*    dragspelet      */


.panel {
    width: 25em;
    background-color: lightgray;
    transition: max-height 0.2s ease-out;
    max-height: 7em;
    overflow-y:scroll;
```

```
    overflow-x:hidden;

}



.panel-heading {
    background-color: white;
    color: black;
 }



.panel-title::after {
    font-weight: bold;
    float: right;
    margin-left: 5px;
    color: black;
    content: '\02C4';
}



/*  dragspelets Kollapstabeller  */

.collapsTable1, .collapsTable2, .collapsTable3, .collapsTable4, .collapsTable5,
.collapsTable6 {
    background-color: white;
    width: 23.5em;
    height: 3em;
    color: black;
    text-align: left;
    font-size: 15px;
    font-weight: normal;
    transition: 0.8s;
}


   /* Kollapstabllernas uppåtpilar  */

   .collapsTable1::after, .collapsTable3::after, .collapsTable5::after {
        font-weight: bold;
        font-size: 15px;
        float: right;
        margin-left: 5px;
        color: black;
```

```
        content: '\02C4';
    }




    /* Kollapstabllernas nedåtpilar  */

    .collapsTable2::after, .collapsTable4::after, .collapsTable6::after {
        font-weight: bold;
        font-size: 15px;
        float: right;
        margin-left: 5px;
        color: black;
        content: '\02C5';
    }




    /* ..................................................... */
    /* ..................................................... */



    /*  Top10 (10 mest frekvent spelade låtar) komponenten */


    /* tabellen  */

     /* invertera striping:en på tabellen   */
.table-striped > tbody > tr:nth-child(1n+1) > td, .table-striped > tbody > tr:nth-child(1n+1) > th
{
    background-color: #d9d9d9;
}

.table-striped > tbody > tr:nth-child(2n+1) > td, .table-striped > tbody > tr:nth-child(2n+1) > th
{
    background-color: white;
}


/* headers   */

.top10headers{
```

```
   color:white;
   font-weight:900;
   font-size:x-large;
}



/* ..... */
/* ..... */



/*  AdminPlaylists komponenten  */



/* header  */

#songheadercoloradmin {
    background-color: #3333ff;
    color: white;
    font-weight: bold;
    font-size: 22pt;
}


/* tabell med spellistor där även knappar m.m ingår*/

.list-group {
    width: 50em;
}




/* knappar */
#nextbutton {
    position: relative;
    top: 10px;
    left: 0px;
}


#prevbutton {
    position: relative;
```

```
    top: 10px;
    left: 0px;
}


#toaddsongbutton {
    position: relative;
    top: 10px;
    left: 0px;
}


#toeditsongbutton {
    position: relative;
    top: 10px;
    left: 27px;
}


#deletesongbutton {
    position: relative;
    top: 10px;
    left: 55px;
}


#buttonCollapsePlaylistsAdmin {
    position: relative;
    top: 10px;
    left: 83px;
}


#adminlistofplaylists {
    position: relative;
    top: 55px;
    left: 10px;
}




/* Add Playlist Area:n */
#backToMainAreaButton {
    position: absolute;
    top: 565px;
```

```
    left: 10px;
}



/* Edit Playlist Area:n */


#backToMainAreaButton2 {
    position: absolute;
    top: 475px;
    left: 10px;
}



/* Egenskaper för playlist-listan i admin-komponenten  */


#playlistadminprops {
    background-color: lightgray;
    width: 25em;
    height: 10em;
    overflow-y: scroll;
    overflow-x: hidden;
    border: 30px solid lightgray;
}




/* ..... */
/* ..... */



/* AddPlaylist komponenten */
/* bakgrundsfärger för textboxar och labels  */
.songitemcolor {
    color: white;
}



/* positionering: labels  */
```

```
#artistlabel {
    position: absolute;
    top: 77px;
    left: 10px;
}



#albumlabel {
    position: absolute;
    top: 160px;
    left: 10px;
}

#titlelabel {
    position: absolute;
    top: 240px;
    left: 10px;
}

#filelabel {
    position: absolute;
    top: 318px;
    left: 10px;
}


#playlistlabel {
    position: absolute;
    top: 398px;
    left: 10px;
}

#playlistlabel2 {
    position: absolute;
    top: 435px;
    left: 10px;
}



/*  positionering:  textboxar   */
#artisttextbox {
    position: absolute;
```

```
    top: 130px;
    left: 10px;
}

#albumtextbox {
    position: absolute;
    top: 212px;
    left: 10px;
}


#titletextbox {
    position: absolute;
    top: 292px;
    left: 10px;
}


#filetextbox {
    position: absolute;
    top: 370px;
    left: 10px;
}


#playlisttextbox {
    position: absolute;
    top: 465px;
    left: 10px;
}




/*  add song sänd-knapp */

#btnAddSong {
    position: absolute;
    top: 511px;
    left: 10px;
}


/*  edit song sänd-knapp */

#btnEditSong {
```

```
    position: absolute;
    top: 420px;
    left: 10px;
}



/* ..................................................... */
/* ..................................................... */



/* media queries justeringar */



/*    AdminPlaylists komponenten  */


@media (max-width: 767px)
{
    #toaddsongbutton {
        position: relative;
        top: 10px;
        left: 0px;
    }

    #toeditsongbutton {
        position: relative;
        top: 10px;
        left: 0px;
    }


    #deletesongbutton {
        position: relative;
        top: 20px;
        left: 0px;
    }


    #buttonCollapsePlaylistsAdmin {
```

```
        position: relative;
        top: 20px;
        left: 0px;
    }


    #adminlistofplaylists {
        position: relative;
        top: 40px;
        left: 20px;
    }


    /* header  */

    #songheadercoloradmin {
        background-color: #3333ff;
        color: white;
        font-weight: bold;
        font-size: 12pt;
    }


    /* tabell med spellistor där även knappar m.m ingår*/

    .list-group {
        width: 25em;
    }


}


/* ..... */
/* ..... */


/*  Top10 (10 mest frekvent spelade låtar) komponenten; Mediajusteringar */


@media (max-width: 767px) {

    /* headers   */
```

```
    .top10headers {
        color: white;
        font-weight: 900;
        font-size: 12pt;
    }

}



/* ..................................................... */
/* ..................................................... */



/*   PlayerMachine - komponenten   */


@media (max-width: 767px) {

    /* Positionsgrupper för att underlätta positionering av olika grupper av objekt */


    #psGrp2 {
        position: absolute;
        left: 3em;
        top:  20em;
    }


    #psGrp3 {
        position: absolute;
        left: 3em;
        top: 27em;
    }



    #psGrp4 {
        position: absolute;
        left: 3em;
        top: 45em;
    }
```

```
/* Flytta yttersta kontroll-knappen (nextbutton) till ny rad  */

#nexxtbutton {
    position: relative;
    left: -14em;
    top: 3em;

}


/*  Flytta textrutor och labels till vertikala positioner  */



/* Labels till textboxar  */


#playlistAlbumlabel {
    position: absolute;
    left: 6.5em;
    top: 3.5em;
}


#playlistSongTitlelabel {
    position: absolute;
    left: 6.5em;
    top: 7em;
}




#playlistNamelabel {
    position: absolute;
    left: 6.5em;
    top: 10.5em;
}




/*  textboxar */

#playlistAlbum {
    width: 15em;
```

```
        position: absolute;
        left: 8em;
        top: 6.3em;

    }



    #playlistSongTitle {
        width: 15em;
        position: absolute;
        left: 8em;
        top: 11em;
    }



    #playlistName {
        width: 10em;
        position: absolute;
        left: 8em;
        top: 15.7em;
    }



}









    /*  slut på eget css  */
```

```
    /* ..... */
    /* ..... */




    @media (max-width: 767px) {
        /* On small screens, the nav menu spans the full width of the screen. Leave a space for
it. */
        body {
            padding-top: 50px;
            background-color: #3333ff;
        }
    }



    @media (min-width: 768px) {
        /* On small screens, convert the nav menu to a vertical sidebar */
        .main-nav {
            height: 100%;
            width: calc(25% - 20px);
        }

            .main-nav .navbar {
                border-radius: 0px;
                border-width: 0px;
                height: 100%;
            }

            .main-nav .navbar-header {
                float: none;
            }

            .main-nav .navbar-collapse {
                border-top: 1px solid #444;
                padding: 0px;
            }

            .main-nav .navbar ul {
                float: none;
            }

            .main-nav .navbar li {
```

```
            float: none;
            font-size: 15px;
            margin: 6px;
        }

            .main-nav .navbar li a {
                padding: 10px 16px;
                border-radius: 4px;
            }

        .main-nav .navbar a {
            /* If a menu item's text is too long, truncate it */
            width: 100%;
            white-space: nowrap;
            overflow: hidden;
            text-overflow: ellipsis;
        }
}
```

Övrigt

package.json

De npm-paket som finns för närvarande installerade. Här bör väl den react-player-komponent som importerades noteras, då den inte per default ingår i React.

```
{
  "name": "EJ_Box",
  "private": true,
  "version": "0.0.0",
  "devDependencies": {
    "@types/history": "4.6.0",
    "@types/react": "15.0.35",
    "@types/react-dom": "15.5.1",
    "@types/react-hot-loader": "3.0.3",
    "@types/react-router": "4.0.12",
    "@types/react-router-dom": "4.0.5",
    "@types/webpack-env": "1.13.0",
    "aspnet-webpack": "^2.0.1",
    "aspnet-webpack-react": "^3.0.0",
    "awesome-typescript-loader": "3.2.1",
    "bootstrap": "3.3.7",
    "css-loader": "0.28.4",
    "event-source-polyfill": "0.0.9",
    "extract-text-webpack-plugin": "2.1.2",
    "file-loader": "0.11.2",
    "isomorphic-fetch": "2.2.1",
    "jquery": "3.2.1",
    "json-loader": "0.5.4",
    "react": "15.6.1",
    "react-dom": "15.6.1",
    "react-hot-loader": "3.0.0-beta.7",
    "react-router-dom": "4.1.1",
    "style-loader": "0.18.2",
    "typescript": "2.4.1",
    "url-loader": "0.5.9",
    "webpack": "2.5.1",
    "webpack-hot-middleware": "2.18.2"
  },
  "dependencies": {
    "react-player": "^1.8.0"
  }
```

```
}
```

Startup.cs

Startup-klassen har en “Configure Services”-metod för att konfigurera programmets services. En service är en återanvändbart komponent som förser programmet med funktionalitet. Services konfigureras i metoden “ConfigureServices” och konsumeras i hela programmet med hjälp av så kallad “dependency injection (DI)” (Går i fallet nedan till så att interfacet IConfiguration läser in konfigurering).

I startup-klassen ingår även en “Configure”-metod för att skapa programmets “request processing pipeline” (ingen aning om vad det svenska begreppet är?). Denna “pipeline” hanterar MVC:s livscykel och beskrivs i nästa stycke och de punkter som ingår i denna livscykel beskrivs i punktform nedan.

Denna konfigurerings-metod används därmed alltså för att specificera hur programmet hanterar HTTP-förfrågningar. Pipeline:n är konfigureras genom att addera så kallad “middleware” komponenter till en instans av interfacet IApplicationBuilder. Interfacet är sedan i sin tur accessbar för “Config”-metoden (men inte registrerad i service-containern). Programmet skapar en IApplicationBuilder som används som parameter direkt till konfigurerings-metoden “Configure”.

- Routing
- Controller initiering
- Action exekvering
- Result exekvering
- View initiering och rendering

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.SpaServices.Webpack;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;

using EJBox.Data;
using EJBox.Data.Models;
```

```
namespace EJ_Box
{
   public class Startup
   {
      public Startup(IConfiguration configuration)
      {
         Configuration = configuration;
      }

      public IConfiguration Configuration { get; }

      // This method gets called by the runtime. Use this method to add services to the
container.
      public void ConfigureServices(IServiceCollection services)
     {
         services.AddDbContext<EJBoxContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("EJBCalldB")));

         services.AddMvc();
      }

      // This method gets called by the runtime. Use this method to configure the HTTP
request pipeline.
      public void Configure(IApplicationBuilder app, IHostingEnvironment env)
      {
         if (env.IsDevelopment())
         {
            app.UseDeveloperExceptionPage();
            app.UseWebpackDevMiddleware(new WebpackDevMiddlewareOptions
            {
               HotModuleReplacement = true,
               ReactHotModuleReplacement = true
            });
         }
         else
         {
            app.UseExceptionHandler("/Home/Error");
         }

         app.UseStaticFiles();

         app.UseMvc(routes =>
         {
            routes.MapRoute(
               name: "default",
               template: "{controller=Home}/{action=Index}/{id?}");
```

```
            routes.MapSpaFallbackRoute(
                name: "spa-fallback",
                defaults: new { controller = "Home", action = "Index" });
        });
    }
  }
}
```