EJ-Box

En digital jukebox

Tommy Friberg
Examensarbete 20 p
.NET-Utveckling
VT 2019

Innehållsförteckning

Mål och syfte

Mål

Målet är att utveckla en applikation som spelar musikfiler av formatet mp3 och gör det i form av spellistor. Dessa spellistor skall även kunna redigeras såsom lägga till, ta bort, ändra de låtar som ingår i aktuell spellista (CRUD).

Applikationen skall bestå av tre delar, förutom ovan nämnda editerings-del så skall det även ingå själva spelaren som spelar upp spellistor med dess mp3-filer. Det skall även finnas en tabell med de tio mest frekvent spelade låtarna.

Vidare så kommer programmet att består av React som frontend och .Net som backend med en MVC-arkitektur där vyerna har bytts ut mot React-diton och där även en databas ingår för lagring av data gällande låtar och spellistor. Programmet kommer dessutom att vara en så kallad “single-page-app” i som nu nämnts några gånger tre olika delar vilka alla huserar på en och samma sida.

Programmet är tänkt att i huvudsak vara/användas lokalt men med ett webb-gränssnitt och API-funktionalitet (se nedan). Eftersom jag använder webbläsaren Google Chrome själv och den av många anses som den främsta idag och därmed har en stor användarbas så är programmet gjort att fungera endast för Google Chrome. Den är därmed inte testad eller utvärderad för någon annan webbläsare.

Den del som spelar upp spellistan och dess filer är av en ganska ordinär mp3-spelartyp med kontroller som gör att man kan spela aktuell låt, byta till kommande, föregående låt, pausa, loopa, volymreglage, flytta framåt/bakåt i låten etc. Till detta kommer val av låtlista och även kunna från denna aktuella låtlista välja enskilda låtar som skall spelas upp. Dessa listor skall vara dynamiskt genererade.

Det skall även hämtas några kompletterande egenskaper från en musikinriktad Databas API, som addering till de lokalt lagrade egenskaperna för en låt.

Slutligen kan jag även poängtera att eftersom programmet i huvudsak är lokalt fungerande och det därmed är lättare att kontrollera vem som använder den till vad än ett motsvarande program som existerar på nätet, så finner jag inte någon användning för att kunna skapa olika användare som sedan har access till olika delar av programmet (authentication och authorisation). Det krävs enkelt uttryckt inte en så hög säkerhetsnivå.

Syfte

Appens syfte är att kunna spela upp musik av formatet mp3 med spellistor som grund. Som extra funktioner så skall man även kunna skapa och ändra dessa spellistor enligt ovan beskrivning. Ytterligare en extra funktion i programmet är att det finns en lista över de tio

mest frekvent spelade låtarna samt data med upplysningar om från vilken spellista låten kommer, av vilken artist den är gjord, från vilket album osv.

I kort tycker jag därmed programmet kan liknas vid en modern jukebox men med viss utökad funktionalitet som CRUD och en topplista. Därav dess namn: Electronic Juke-Box eller EJ-Box.

Det har även varit min tanke att den här spelaren skall kunna vara som en slags enklare variant till Windows egen Media Player men med exempelvis ett webbgränssnitt. Likt många mp3-spelare och dylika program så läggs tonvikten på den lokal data men kombineras med diton som hämtas från nätet, i mitt fall några extra egenskaper för varje låt som hämtas från DisCogs API.

Det har också varit min tanke att producera ett program som kunde åtminstone i någon utsträckning agera som nämnda Media Player eller andra mp3-spelare för lokalt bruk men ändå under .NET-plattformen (eftersom det här är en .NET-utbildning). Lösningen blev det som jag beskrivit ovan, en mp3-liste-spelare med utökad funktionalitet men då som sagt med webbgränssnitt. Detta har även som senare kommer att nämnas att direkt göra med valet av språk/plattformar för utvecklingsarbetet.

Rent upplysningsvis eftersom DisCogs nämndes ovan; DisCogs är en amerikansk databas som främst inriktar sig mot skivsamlare och därmed innehåller en enorm mängd data om till exempel artister och deras alster. Väldigt bra köpguide både för nya och i ännu högre grad begagnade skivor.

DisCogs hemsida:
https://www.discogs.com/

Discogs API-dokumentation för utvecklare:
https://www.discogs.com/developers/

Slutligen tio-miljoner-kronors-frågan och den vita elefanten i rummet; Varför jag skapade en sådan här applikation? Vad var den springande grundtanken från allra första början?

Svar: Helt enkelt för att kultur (musik i det här fallet) och teknologi (programmering odyl i det här fallet) är mina två största intresseområden. Jag ville helt enkelt kombinera dessa och se om jag kunde få ihop en sådan här applikation. Vilket jag kunde.

Introduktion

.NET-utveckling

Nedan följer utläggningar om några av de mest essentiella punkterna beträffande .NET-utveckling och kan därmed tjäna som en god introduktion till ämnet.

Ramverk: NET Framework

.NET Framework (i fortsättningen kallat ramverk) är en integrerad komponent av operativsystemet Windows som inkluderar ett virtuell exekverings system kallat “Common Language Runtime” (CLR) och ett stort klassbibliotek. CLR är Microsofts egen version av CLI (Common Language Infrastructure), en internationell standard som utgör basen för att skapa lösningar och utvecklingsmiljöer där språk och bibliotek skapar samarbetande och flytande enheter.

Källkod skriven i C# är kompilerad till ett mellanliggande språk (intermediate language, IL) som rättar sig efter CLI specifikationen. IL-koden och dess resurser, såsom bitmaps och strings, är sparade på disk i en exekverbar fil som kallas en “assembly”, vanligtvis med en fil extension av typen exe eller dll. En assembly innehåller ett manifest gällande assemblys datatyper, version, kultur och säkerhetskrav.

När aktuellt program exekveras, så laddas assemblyn in i CLR, vilken kan utföra olika åtgärder beroende på tidigare nämnda manifests innehåll. Sedan om säkerhetskraven möts, utför CLR JIT (Just In Time) - kompilering för att konvertera IL-koden till lokala maskinkods-instruktioner. CLR utför även andra göromål så som automatisk garbage-collection, undantagshantering och resurshantering. Kod som är exekverad av CLR är ibland refererad till som “manage code”, i kontrast till “unmanaged code”, vilken är kompilerad till lokal maskinkod som har ett specifikt system som mål.

Interoperabilitet hos ramverket

Att kunna fritt jobba mellan olika .NET-språk är huvudfunktionerna hos .NET-ramverket. På grund av att IL-koden som produceras utgår från “The Common Type Specification” (CTS), så kan IL-kod genererad från säg C# interagera med kod från mer än 20 andra CTS-språk så som. Visual BAsic och Visual C++. En enda assembly kan innehålla multipla moduler som är skrivna i olika .NET-språk. Dess typer kan sedan vidare referera till varandra som om de alla var skrivna i samma språk.

Förutom de så kallade “run time”-tjänsterna, så innehåller .NET-ramverket ett stort klassbibliotek med över 4000 klasser organiserade som så kallade “namespaces”, som bidrager till en stor mängd varierande funktionalitet. Dessa mallar/förprogrammerad kod innehåller kod för en rad olika ändamål som tex kryptografi, databashantering, nätverk,

webbtjänster och algoritmer. Detta gör utvecklingsarbetet mycket lättare när något så att säga “att starta med”.

Den typiska .NET-programmet använder ramverkets klassbibliotek för att hantera många vanliga göromål.

Språk: C# (med flera)

Programmeringsspråket C# (C Sharp) är det mest frekvent använda av de språk som används inom ramverket. Ses även som ramverkets “standardspråk” för utveckling av mjukvara på ..NET-plattformen. Dess utveckling har löpt sida-vid-sida med .NET och gör så fortfarande. C# kan man säga är en vidareutveckling från föregångaren C++ med influenser från JAVA.

Det finns fler språk än C# för .NET-användare, som exempelvis F#, Visual Basic, J#, C++ med flera. Till dessa hör utvecklingsmiljöer och kompilatorer som kan kompilera dessa språk till CLI (Common Intermediate Language) som CLR sedan tolkar som maskinkod. Detta bidrar starkt till .NET:s plattformsoberoende.

Utvecklingsmiljö

För att kunna utveckla program för .NET behöver vi en så kallad utvecklingsmiljö. En sådan består i stora drag av följande delar:

- Texteditor
- Debugger
- Kompilator/Interpretator

Texteditorn är i grunden inte helt olikt en reguljär texteditor (som anteckningar i Windows) men med en rad olika hjälpmedel såsom “Syntax highlighting” och “Intellisense”. Syntax highlighting är kodarnas motsvarighet till färg överstrykningspenna, olika slags kod och/eller delar av kod markeras med olika färger för att göra dem mer överblickbara och lättare kan upptäcka fel.

En debugger är ett verktyg som hjälper oss att hitta fel i vår kod vid exekvering. Man kan exempelvis stegvis gå genom programmet och se vad som vid varje steg händer i programmet. Man kan också lägga in så kallade brytpunkter som avbryter ett programs körning vid den punkt där brytpunkten finns och där kan man sedan analysera programmet för att på så sätt hitta fel.

Metod

Planering av lösning

Lösningen har sin grund i en React-laboration(ett quiz-spel). Dess struktur/arkitektur är densamma med tre olika delar enligt vad som har beskrivits förut. Tanken var att omvandla spelet till en spelliste-spelare/digital jukebox.

De två stora språken/plattformarna i mitt program är React för frontend och MVC med Entity Framework som backend. React för att det har en så mycket mer omfattande funktionalitet för frontend-utvecklings-biten än vad vanliga vyer i MVC har. Detta främst med importeringen och implementeringen av komponenten som utgör själva spelar-delen av programmet i åtanke. Jag är ingen expert ännu men mig veterligen skulle det bli mycket svårt att göra något sådant med MVC-vyer. Möjligt kanske men React sågs som ett mer lämpligt val. MVC som backend med EF är också ett naturligt val med tanke på den smidighet som det utgör att kunna beskriva sin databas med databas-kontex, modellklasser osv och sedan via migrationsprocessen direkt kunna ha en synkad motsvarighet i själva databasen. .NET:s huvudspråk C# används eftersom det är det mest gångbara och det som jag behärskar. Övriga skäl att kunna välja C# nämns i avdelningen Introduktion ovan där språket behandlas.

Vidare så utgick planeringen av lösningen från att göra det som uppfattades som svårast och/eller mest omfattande först och sedan arbeta sig neråt mot mindre/enklare uppgifter. Det skulle ske i små segment vars funktionalitet frekvent kollades samt utvärderades om det hade blivit som man tänkt sig innan man gick vidare till nästa del. Frekventa backuper var också del av strategin.

Produktion av lösning

Först ut beträffande produktionen av programmet var importeringen och implementation av själva React Player-komponenten där den del av programmet som skötte själva spelet skulle bytas ut mot ett som skötte uppspelning av låtar från spellistor (en huvuddel byttes med andra ord mot en annan), ett moment som jag aldrig hade gjort tidigare. Den React-komponent som jag valde efter ha provat två andra som jag inte fick att fungera riktigt, hade en hel del exempelkod med sig och relativt god dokumentation vilket hjälpte till. Men att sedan anpassa komponenten till min applikation efter hur jag ville ha det visade sig vara en mycket mer omfattande och tuffare uppgift än jag trodde. Väldigt många funktioner, eventhanterare, props med mycket mera som dessutom skulle samverka och veta exempelvis vilket state som skulle gälla i olika situationer.

Någon gång under det inledande arbetet som beskrivs i stycket ovan så kom behovet att kunna testa den importerade komponenten eller helt enkelt kunna spela de konkreta fysiska mp3-filerna. Det betyder i och med att programmet använder sig av ett webbgränssnitt att

man behöver kunna få access från nätet (via lokal ip-adress) till ens egna lokala dator för att därifrån kunna komma åt att spela upp mp3-filer. Problemet med det gällande Google Chrome som det här programmet är utvecklat för, är att Chrome inte tillåter detta. Det är bra i sak för ingen skall ju från nätet kunna nå access på en lokal dator utan vidare. Men i vårt fall så ställde det till problem. Lösningen som jag valde på dessa två problem var Chromes egen webbserver, en liten smidig webbserver som gör att mp3-filerna blir nåbara. Det kräver dock att en sådan installeras (eller rättare sagt läggs till, den är ett s.k “tillägg” till Chrome) och konfigureras först innan man kan använda detta program. Samt kopierar de mp3-filer man vill ha till mappen “mp3” under projektmappen.

För att nå access till mp3-filerna så måste man först lägga till Chromes webbserver till ens Chrome webbläsare, vilket kan göras här:

https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb

Sedan kan man göra diverse inställningar och nedan är den lokala ip-adress samt port som anges i mitt program och används av Chrome web-server:

http://127.0.0.1:8887

Vald root-mapp är som sagt var “mp3” under projektmappen.

Slutligen gällande den del av programmet som spelar upp låtlistor/låtar så är här nedan en länk till den react-komponent som jag valde som spelare av mp3-filerna och som även agerar som den viktigaste delen/komponenten av denna lösning. Komponenten heter “react-player och har version: 1.11.0”.

https://www.npmjs.com/package/react-player#multiple-sources-and-tracks

Område nummer två var att göra databasen. Planera och implementera den inklusive dess kontext och modellklass. Köra migrationer för att synka mvc/backend-delen med databasen (enligt Entity Frameworks principer). Databasen begränsas till endast en tabell då exempelvis spellistor endast skulle innehålla en egenskap, spellistans namn (förutom egenskapen ID som alltid måste ingå), så jag ansåg inte det vara någon ide att göra sådana utbrytningar till flera tabeller, samt förhållanden mellan dessa. För att tex erhålla alla låtar som tillhör en viss spellista så filtreras dessa ut med kriteriet namnet på den spellista som efterfrågas. Fler utbrytningar kunde tänkas som mer data om varje artist eller varje album osv. Men jag ville hålla egenskaperna för varje post till ett minimum och nöjde mig därför med artist, album och låttitel samt övriga egenskaper som behövdes för att få det att fungera. Även tex genre valdes bort. Keep it simple (Dock införde jag senare med hjälp av hämtning från Discogs API ytterligare tre egenskaper för varje låt för att på så sätt implementera hämtning av extern data, se även nedan).

För att kunna få igång databasen lätt vid implementation av denna lösning på ett lätt sätt så anger jag här nedan dess anslutningssträng inklusive namn.

Anslutningssträngens namn: EJBCalldB

Total anslutningssträng:
Data Source=(localdb)\MSSQLLocalDB;**Initial Catalog=EJB**;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False

Nästa stora område var sedan att anpassa övrig befintlig kod som var avsett för ett spel till att sköta dess nya uppgifter i en musikalisk kontext, vilket visa sig vara ett monumentalt jobb och mycker mer omfattande än vad jag först trodde. Detta inkluderade allt från att byta namn till kod som fungerade annars korrekt i syn nya kontext till att byta ut, ersätta eller ta bort kod som inte längre fyller något syfte.

Det sista momentet gällande själva kodningen var att lägga till hämtning av data från DisCogs API. Där kompletterade jag de lokalt lagrade egenskaperna med sådant som inte ingick där som respektive låts genre, utgivningsland och utgivningsår. På grund av en redan trång layout fick dessa tre uppgifter samsas i samma textruta med med i alla fall kommatecken mellan sig.

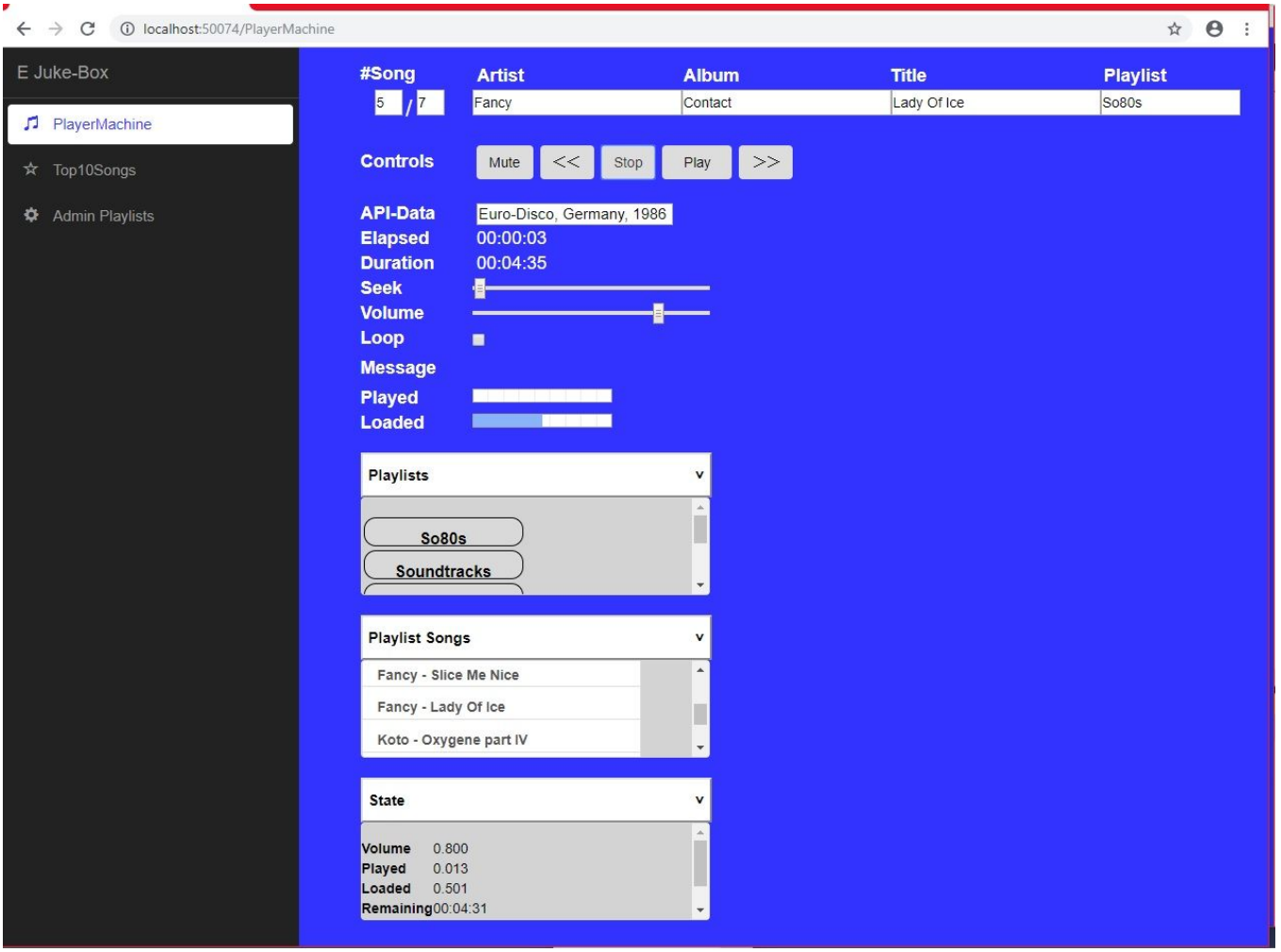
Slutligen så var det en stor mängd justeringar, testande av funktionaliteter så allt fungerande osv.

Val av metod

Metoden som valdes, vilket väl redan kan anas enligt ovan, var en form av agil metodik. Tyngdpunkt lades på att inte programmera för mycket i taget utan jobba med väl avgränsade uppgifter, sedan testa så att det man nyss gjort verkligen fungerande till belåtelse innan man gick vidare med nästa. Detta som sagt i kombination med frekventa backuper gjorde det lätt att även implementera iteration när så ansågs nödvändigt. Backuperna räddade även projektet ett fåtal gånger då misstag skedde då man i iterativ andra kunde ta bort den variant av projektet man jobbade på och gå tillbaka till senaste backupen och börja om.

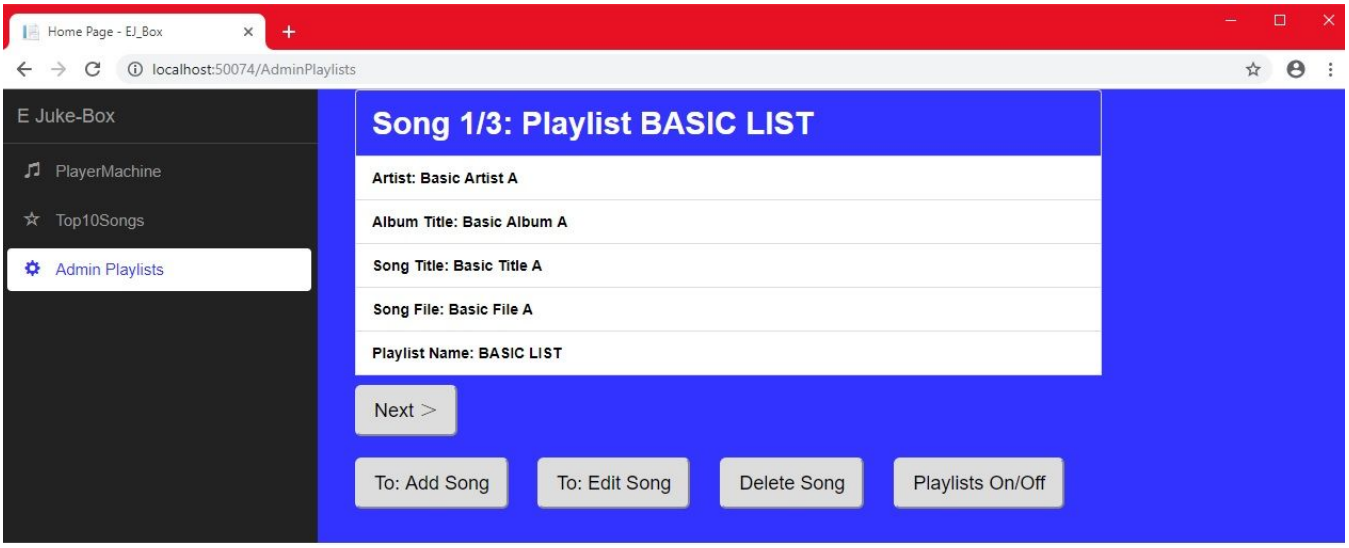
Det hade blivit mycket mer komplicerat att göra så om man hade programmerat stora delar av applikationen innan man testa den osv. Det hade även varit svårare i så fall att upptäcka fel, när man inte gör för mycket i taget blir sådant mycket lättare att detektera. Nu så blev det en personlig form av korta sprintar med uppföljning.

Resultat

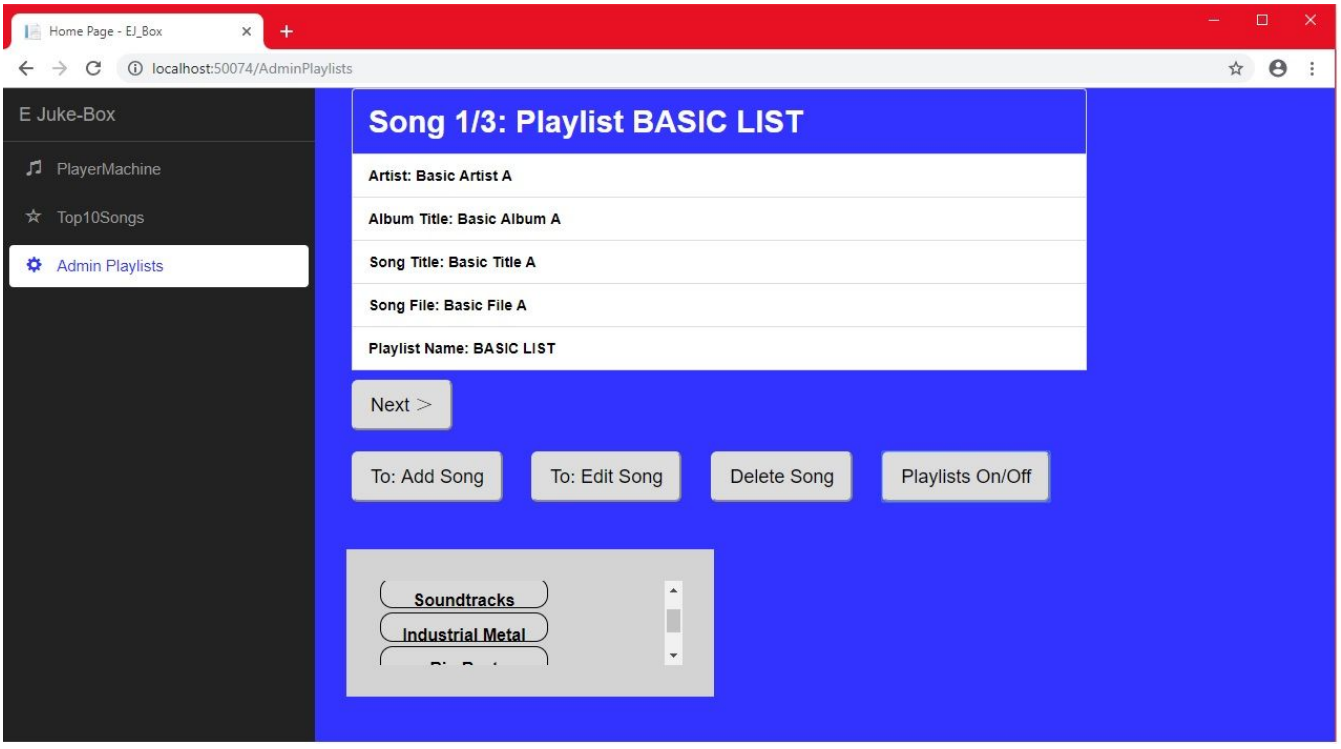


Huvuddelen av detta program som jag valt att kalla för “PlayerMachine” och den del som spelar upp låtlistan och dess låtar. Här finns också de vanliga funktionerna som man finner i de flesta mp3-spelare såsom play, stop, pause, byta till nästa/föregående låt osv. Man kan även välja om låtbyte skall ske med hjälp av dessa kontroller eller från “Player Songs”-listan dvs sköta både list- och låtval från dessa s.k kollaps-tabeller som för övrigt är dynamiskt genererade listor.

Det bör även noteras att innehållet i textrutan med rubriken “API-Data” hämtas från DisCogs API.

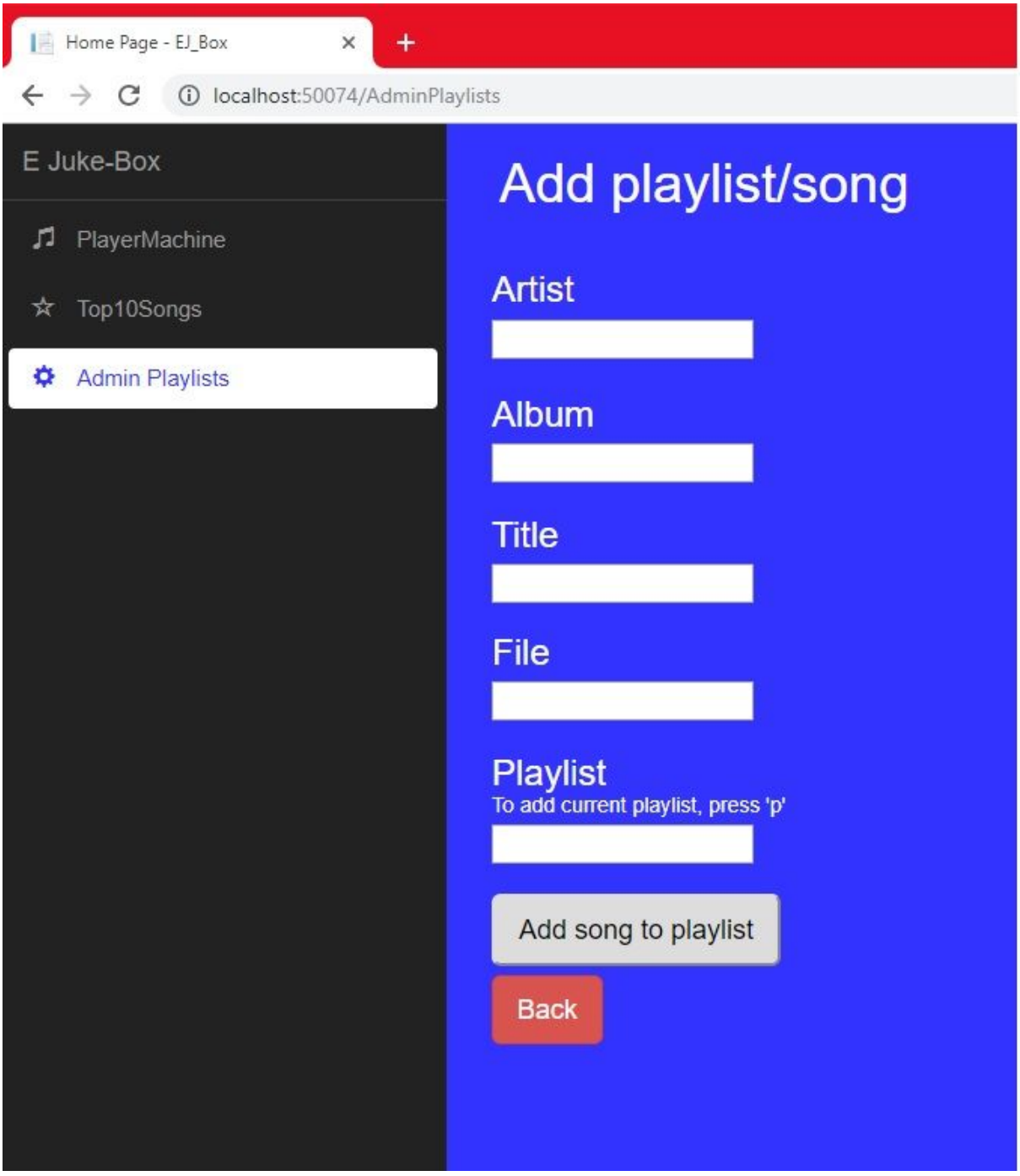


Näst viktigaste delen av denna applikation utgörs av vad jag kallar “Admin Playlists”. Det innebär helt enkelt editering och skötsel av låtar och låtlistor. Man kan visa låtlistan och dess låtar, man kan lägga till, ändra, ta bort osv. Ta bort -funktionaliteten är bara en knapp och har ingen egen area **(vad som annars brukar kallas vanligtvis för vy, kallas fortsättningsvis här alltså för area)**. Centrum för allt detta är denna area, som även agerar som Parent till areorna “Add Song” och “Edit Song” med tillhörande respektive knappar enligt bilden ovan. Nedan skall vi titta närmare på dessa areor. Allmänt och för övrigt kan man säga att alla förändringar som görs här också slår igenom i “Player Machine”-delen av programmet.



När man trycker på knappen “Playlists On/Off” så dyker denna scrollbara ruta upp. Den innehåller en dynamisk lista över spellistorna där man dubbelklickar på den spellista som

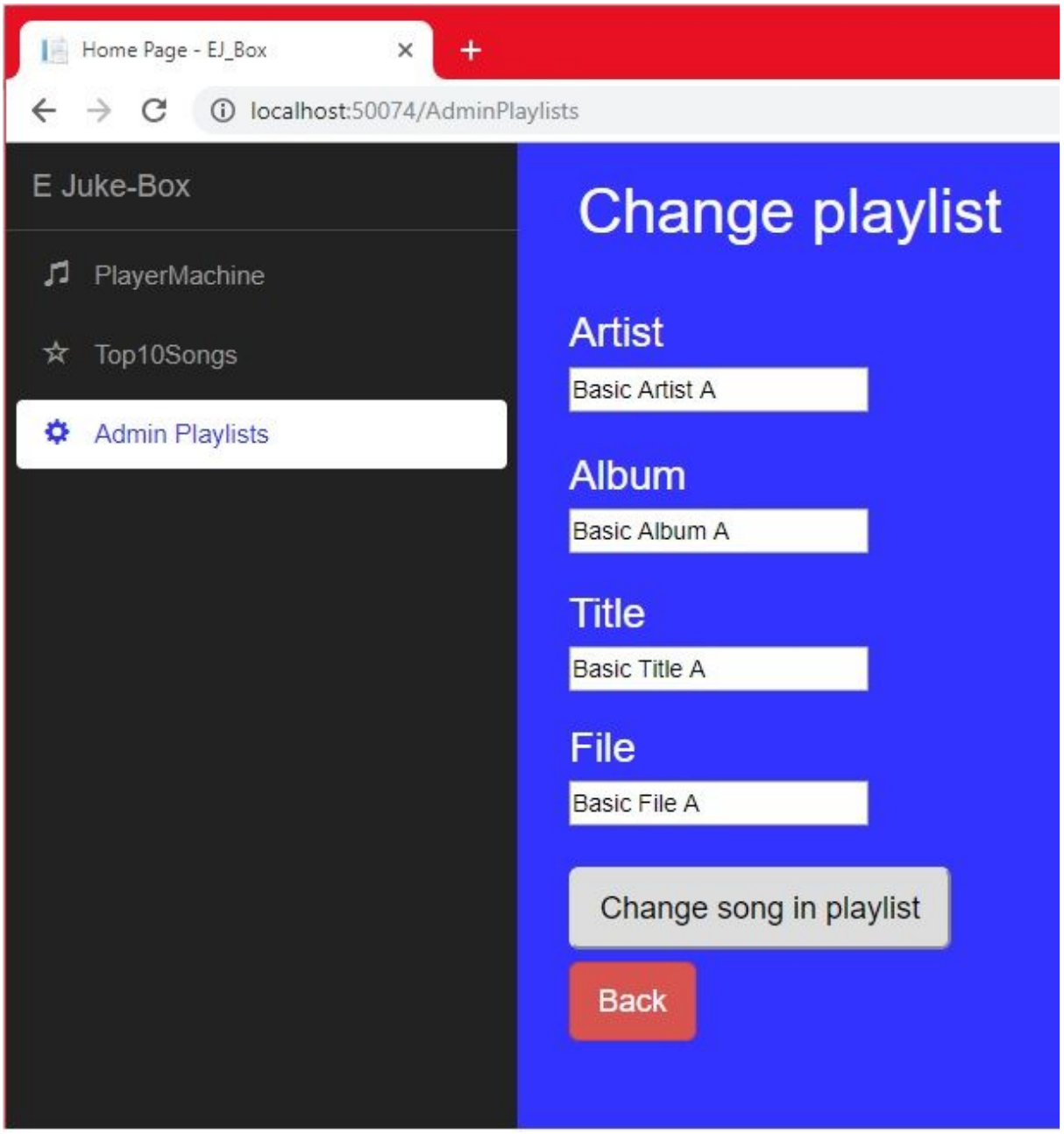
man vill jobba med. Motsvarande lista finns i “Player Machine” delen av programmet (se ovan).



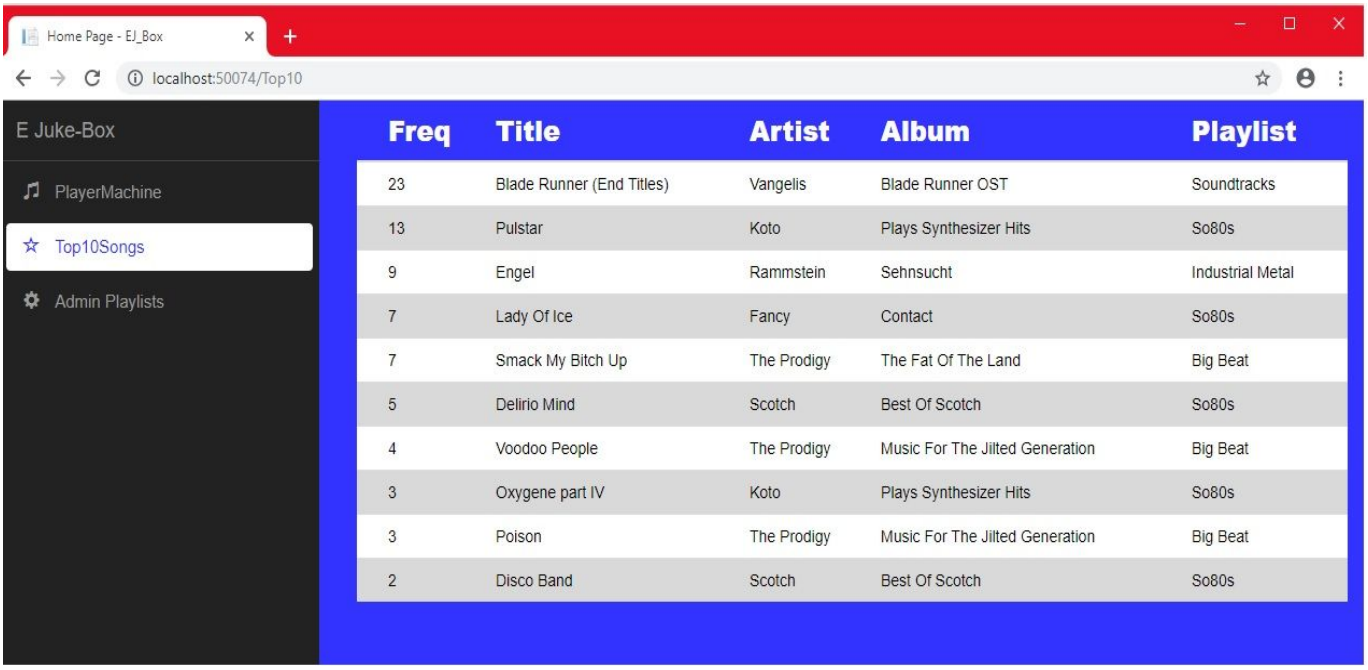
Detta är den area som dyker upp efter man har tryckt på knappen “To: Add Song”.Här kan man lägga till en sång till dels en befintlig lista (tryck ‘p’ när du befinner dig i textrutan ‘Playlist’ för att få fram aktuell spellista) eller skriv in namnet på en ny och den skapas. **Eftersom programmet endast hanterar dynamiska spellistor så finns en spellista kvar så länge någon post för en låt har det som värde i databasen för sin “Playlist”-egenskap.** File-textrutan är själva namnet på den fysiska filen (resten av sökvägen inklusive lokal url och “mp3”-extensionen fylls i automatiskt).

Känner här att det finns bitar av information på flera ställen gällande att lägga till en fysisk mp3-fil och dess tillhörande data så jag anger därför här hela processen enligt följande:

- Kopiera den fysiska mp3-filen från dess ursprungliga ställe till mappen “mp3” under projektmappen.
- Kopiera mp3-filens namn (genom att trycka först på F2 och sedan kopiera namnet)
- Gå in i “Admin Playlist” i React-programmet enligt ovan
- Tryck på knappen “To: Add Song” för att komma till rätt area
- Fyll i textrutorna också enligt ovan
- Tryck på knappen “Add song to playlist” och låten och dess spellista läggs till i databasen
- Sist så trycker du på knappen “Back” för att återgå till “Admin Playlists”- arean där man kan se den nyligen inlagda låten/spellistan.

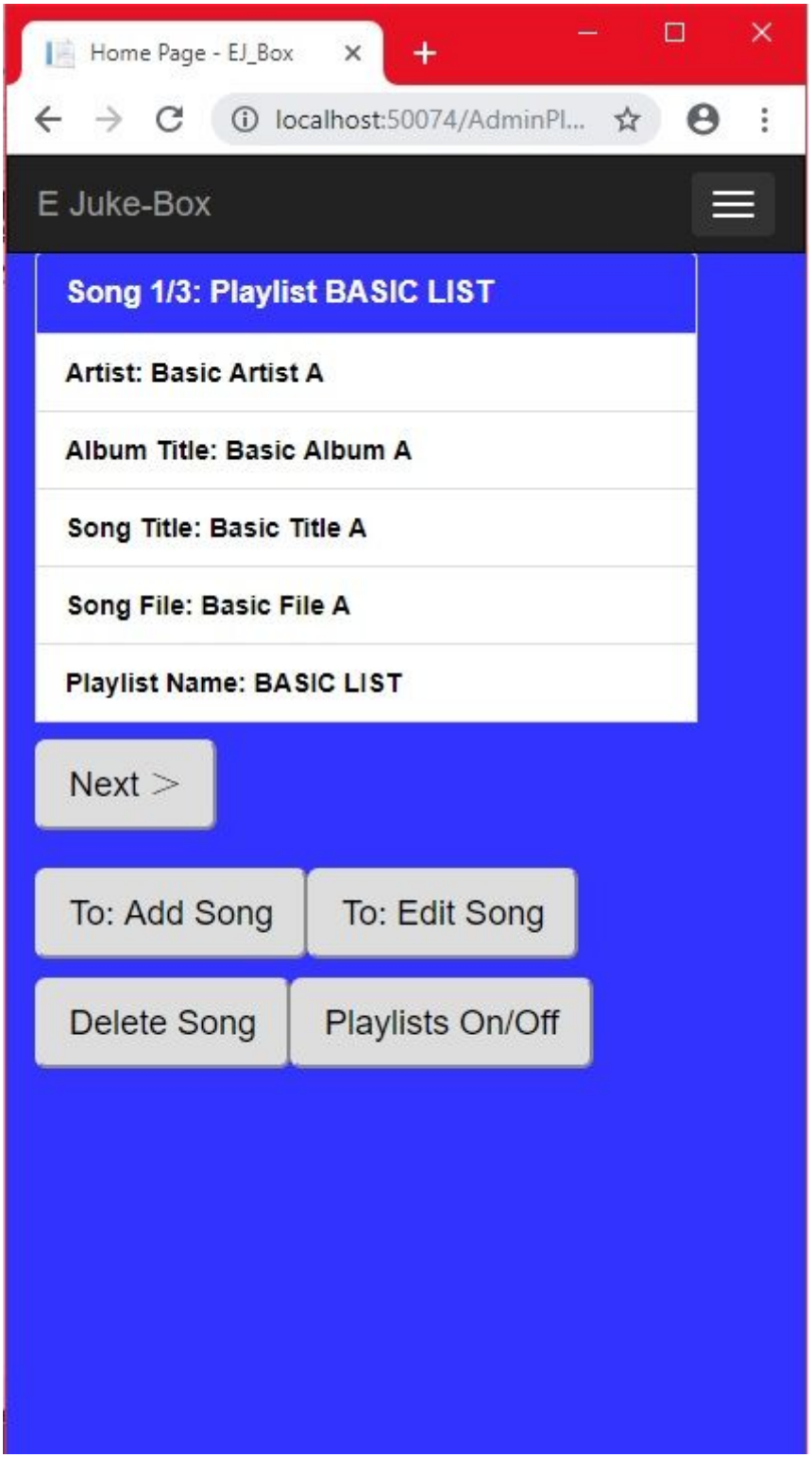


Detta är den area som dyker upp efter man har tryckt på knappen “To: Edit Song”. Denna area är mycket lik föregående förutom att här ändrar man istället för att lägga till. Det som dyker upp här är den aktuella låt som man valt. I textfälten ändrar man till vad man nu vill att de skall innehålla för värden. När man känner sig färdig trycker man på knappen “Change song in playlist” och låten ändras i databasen. Precis som föregående area så trycker man allra sist sedan på knappen “Back” för att återgå till “Admin Playlists”- arean där man kan se den nyligen ändrade låten med dess nya värden.



Den tredje och sista delen av programmet utgörs av en “tio-i-topp-lista” över de tio mest frekvent spelade låtarna och övrig data som artist, album samt från vilken spellista låten härrör. Listan sorteras först fallande efter hur ofta låten har spelats (frekvens) och som en andra sortering för de frekvenser som råkar vara lika så sorteras låten efter dess titel i stigande ordning.

PlayerMachine-komponenten i emulerat mobile-läge med hjälp av så kallad responsiv design (layoutens komponenter/delar ändras efter upplösningen).



AdminPlaylist-komponenten i emulerat mobile-läge med hjälp av så kallad responsiv design (layoutens komponenter/delar ändras efter upplösningen).

Freq	Title	Artist	Album	Playlist
28	Blade Runner (End Titles)	Vangelis	Blade Runner OST	Soundtracks
15	Engel	Rammstein	Sehnsucht	Industrial Metal
12	Smack My Bitch Up	The Prodigy	The Fat Of The Land	Big Beat
4	Poison	The Prodigy	Music For The Jilted Generation	Big Beat
1	Lady Of Ice	Fancy	Contact	So80s
1	Pulstar	Koto	Plays Synthesizer Hits	So80s
0	Basic Title A	Basic Artist A	Basic Album A	BASIC LIST
0	Basic Title B	Basic Artist B	Basic Album B	BASIC LIST
0	Basic Title C	Basic Artist C	Basic Album C	BASIC LIST
0	Delirio Mind	Scotch	Best Of Scotch	So80s

Top10-komponenten i emulerat mobile-läge med hjälp av så kallad responsiv design (layoutens komponenter/delar ändras efter upplösningen).

Kravspecifikation

- 10 olika komponenter (varav en importerad som inte ingår i React per default)
- 5 areor (vyer)
- Full CRUD-funktionalitet
- Lista med tio mest frekvent spelade låtarna
- Dynamiskt genererade spel- och låtlistor
- Möjlighet att välja låtar både från de vanliga kontrollerna för att välja nästa/föregående låt men även från en dynamiskt genererad låtlista tillhörande aktuell vald spellista
- Hämtning av kompletterande låt-data till lokala diton från DisCogs API

Diskussion

Det här programmet blev faktiskt i stort ganska mycket som jag ville ha det. Sedan finns det ju alltid förbättringar och vidare drömmar och önskningar att uppfylla. Varav vissa jag inte ens är säker på om de går att göra. Mitt huvudsakliga mål var dock att se om jag kunde omvandla den React-laboration vi hade gjort tidigare till ett motsvarande program som spelade spellistor med dess mp3-filer, hantering av dessa och så vidare. En annan fräck feature som jag ville se om jag kunde få till var de dynamiskt genererade så kallade “kollaps-listorna”. Egentligen bara helt enkelt se om jag över huvud taget kunde få ihop det. Det målet kan jag med råge här och nu säga att det blev uppfyllt!

Det här programmet i sin nuvarande form är rejält omfattande och en rejäl best att få ihop då många av dess saker man inte hade pysslat med alls eller inte så mycket förut. Så allt man hade tänkt sig hann jag inte få med.

Förbättringar för framtiden

Några förbättringar och önskningar/drömmar för framtiden:

- Kunna göra om applikationen till en cloud-dito, kanske någon form av cloud-tjänst där man lägger allt i en vm-maskin som innehåller webb-server, databas och blob-storage eller liknande så man kan spara de fysiska mp3-filerna som laddas upp.
- Drag-and-drop möjlighet så man kan addera filer på ett lättare sätt
- Kunna spela enskilda låt-filer som inte nödvändigtvis tillhör en låtlista utan de läggs i en temporärt skapad låtlista som agerar bara som en slags behållare medans låtarna spelas. Denna lista skall sedan kunna namnges och sparas och därmed förvandlas till en reguljär sådan.
- Kunna ta bort en hel låtlista med alla dess låtar. Nu måste man ta bort alla låtar som tillhör låtlistan för att den skall försvinna.
- Kunna inkludera sökvägen till filen vart som helst ifrån, istället för nu där alla filer ligger på ett och samma ställe. Vet inte hur användbart det är men är ändå en lockande tanke tillsammans med ovan drag-and-drop.
- Kunna hämta musik-relaterad data från diverse API och dylikt för att på dynamiskt sätt ständigt ha färsk data och kanske fler egenskaper/information för varje artist/album. Det är ju även lockande tillsammans med cloud-lösningen ovan. Det kräver dock att nuvarande layout för komponenten PlayerMachine görs om radikalt då det för närvarande inte finns någon mer plats för exempelvis textrutor eller om man tänker importera albums omslagsbilder etc. Ett helt nytt tänk gällande layouten med andra ord. Ett ytterligare steg framåt från idag också där det bara hämtas tre egenskaper från DisCogs API. Dessa hämtade värden från API:er skulle även kunna tänkas att sparas ner permanent tillsammans med de lokalt angivna i databasen.
- Man kanske även skulle kunna tänka sig att nollställa “tio-i-topp-listan” även om det är en funktion med frågetecken för sig i och med att det är mer tänkt som en lista att

vara i ständig förändring och därmed inte ha behov av nollställning. Men det är ändå en tanke.