JavaScript Grupparbete Studiegrupp 12, Frågor och Rapport

Innehållsförteckning

Frågor

Hur kan du i din webbläsare se ifall din Javascript-kod lyckas göra en request mot API:et?

Man kan se det utifall man får en statuskod tillbaka. Detta kan man kolla under “network” i inspektorn.

Om statuskoden är 2xx så menas det att det är en “lyckad” request (Success).

Förklara hur man kan debugga Javascript.

- Man kan debugga Javascript med att “console.logga” kod. Vilken menas att man skriver console.log(Xx) i sin kod där Xx kan vara precis vad som helst som man vill testa eller se om ens kod kommer till den positionen.(`console.log()`).
- Man kan sätta ut breakpoints i sin UI och kör koden. Så kommer den stanna vid alla breakpoints man satt ut.
- Man kan också genom VSC stega sig igenom koden rad för rad.
- Man kan använda sig utav ett nyckelord, “debugger”, i sin javaScript kod som gör att den kallar på debuggern och stannar koden på denna plats. (`debugger`).
- Automatiska tester.

Rapport

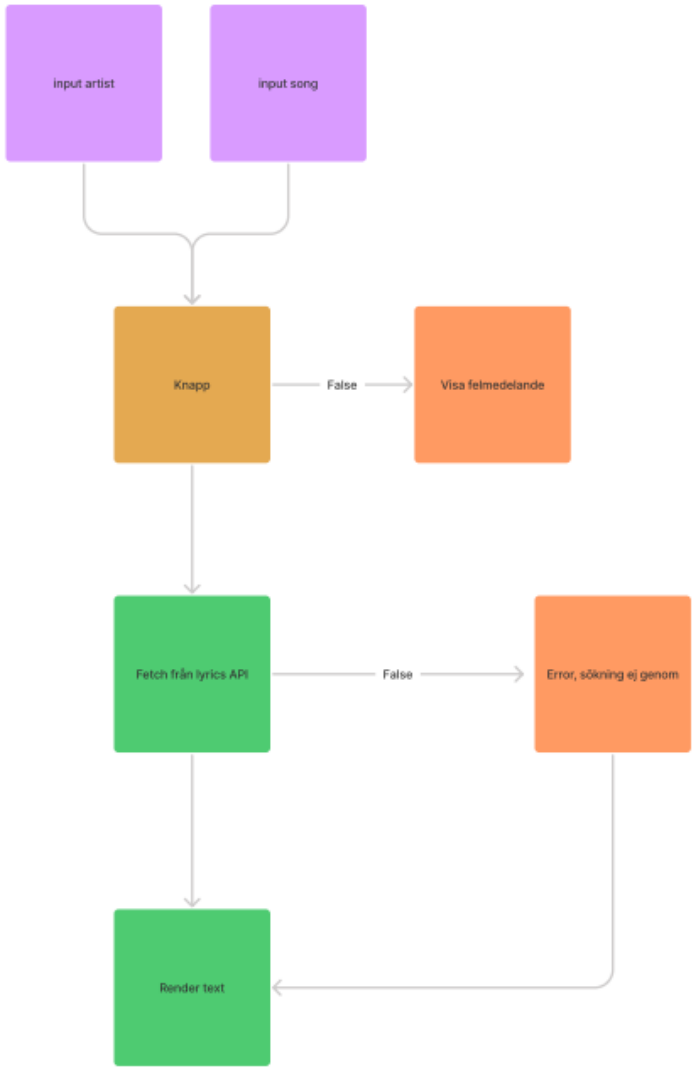
I denna rapport skall eleven beskriva hur den tänkt under de praktiska momenten.

Steg 1, Förarbete

Det första vi gjorde i vår arbetsprocess var att ha ett uppstartsmöte för projektet där vi gick igenom uppgiftsbeskrivningen och kravspecen för projektet. Därefter skapade vi ett gemensamt styrdokument där vi gjorde kravspecen tydligare och skrev upp alla delar som behövde göras och så skapade vi en en simpel to-do lista för projektet.

För att vi skulle få en tydligare bild av hur webbappen ska fungera så valde vi att skapa en enkel flowchart. Syftet med detta var att vi alla skulle vara på samma våglängd när det gäller hur all funktionalitet skall fungera.

Flowcharten:



Steg 2, Designskiss

För designen så valde vi först vilka färger och vilken font som vi skulle använda oss av. Därefter så skapade vi en simpel skiss/wireframe i Figma för att underlätta när vi ska koda upp all HTML och CSS. Vi valde att utgå från en desktop design då den mobila designen kommer att vara väldigt lik vilket gör att det räckte att endast ha en skiss för desktop. Genom att göra en enkel wireframe så blir strukturen av vår html mycket enklare att skapa då man visuellt ser vilka delar som är relaterade till varandra och vilka element som är förälder respektive barn.

Steg 3, Kodande

Efter att designskissen var färdig så började vi med att strukturera upp projektet. Strukturen för projektet blev ganska simpel då det endast skulle finnas en html sida. Vi valde att dela upp CSS, JavaScript och bilder i separata mappar för att få en tydligare struktur för projektet.

HTML och CSS

Efter att vår wireframe var klar så skapade vi all HTML och CSS för projektet. Vi skapade först alla html delar då det blir enklare att styla html:en när all html är på plats. Vi hade dock CSS i tankarna när vi skapade all html då CSS:en är beroende av rätt html struktur för att få flex att fungera som det är tänkt.

Efter att all HTML var på plats så började vi med css:en, vi tog in en normalize.css fil för att neutralisera så mycket av de olika webbläsares grundstyle som möjligt. Därefter använde vi oss av en CSS Boilerplate som Mattis hade skapat sen innan för att få in en hel del grund styling och en struktur på CSS filen. För färgerna så valde vi att använda oss av variabler då Boilerplaten hade variabler för färg redan implementerat.

CSS en är stylad med Mobile First, vi valde att göra på detta sättet då det är enklare att skala upp designen för större skärmar då man för den mestadels lägger till styling än att tar bort som man gör om man designar desktop first.

JavaScript

Efter att all HTML och CSS var på plats så var det dags att börja med funktionaliteten för webbappen.

Det första vi gjorde var att göra hamburgermenyn fungerande. Vi valde att lägga koden för hamburgaren i en mapp som vi namngav till components. Koden för hamburgermenyn är en simpel funktion som gömmer/ visar olika html element beroende på vilken knapp som användaren klickar på. Vi valde att använda oss av import och export för denna komponent. För att få imports att funka med vanilla js så är man tvungen att lägga till type=”module” i script taggen på html sidan och .js på javascript filen som man importerar. Därefter skapade vi en eventlyssnare på sök knappen där vi skapade en arrow function där vi länkade in input fälten. Vi valde att skapa en eventlyssnare för knappen direkt då vi valde att fokusera på användarens input först och ta hand om de delarna först.

Efter gick vi vidare och satte oss in lite i API:et som vi skulle använda oss av, vi läste igenom den dokumentation som fanns och testade sedan att köra get anrop till apiet genom Thunder Client tillägget i Vs Code. Redan här uppstod det några problem

med API:et och det var att apiet är väldigt långsamt vilket gör att det blir svårt att veta när ens kod är rätt eller felaktig då det tar väldigt lång tid att få svar från api:et.

Efter att vi hade testat att anropa API:et med Thunder Client gick vi vidare och skapade en funktion för att fetcha en sång utifrån det användaren har skrivit in i input fälten. Redan här insåg vi att vi behövde ta hand om användarens input då api:et inte gillade mellanrum. För att ersätta mellanrum så använde vi oss av JavaScripts inbyggda funktioner trim() och replace(). Trim använde vi oss av för ta bort mellanrum innan och efter. Replace använde vi oss av regex för att ersätta mellanrummen med %20. Därefter körde vi funktionen på båda inputfältens input.value.

För att säkerställa att användaren har skrivit in något i båda input fälten så skapade vi en if-sats där vi kollar om nån av inputfältens.value.length är lika med 0 så kommer det att visas en text som säger att båda fälten måste skrivas in. Om användaren har skrivit in mer än 0 tecken i båda inputfälten så kommer funktionen för fetch att köras.

Efter att vi säkerställt att värdena från inputfälten blir kompatibla med APIet och att det kommer fram ett felmeddelande om något av inputfälten är tomma så gick vi vidare med funktionen för att fetcha den låt som skrivits in och därefter skriva ut låttexten i vår simpla GUI.

Fetch funktionen är en async await där vi skapade en dynamisk URL som ändras beroende på användarens input. Därefter sker fetchen och för att få texten mer representabel så ersätter vi de JSON formaterade tecken så att låttexten blir mer läsbar. Därefter kör vi en funktion för att displaya texten under sökknappen.

Steg 4, Refaktorisering

Efter att vi skrivit kod som var fungerande så lät vi Kaj som är bäst av oss på JavaScript att refaktorisera våran kod så den blir bättre.