IP2 Viktor Hultman Redogörelse

CSS-ramverk

CSS-ramverk förenklar lösningar med att hålla en konstant design och uppbyggnad genom att bidrar med en grundlig struktur. Ramverkens grundliga struktur bidrar med att utvecklare slipper bygga upp en hemsida från "ruta noll".
Några av de mest populära CSS-ramverken är för tillfället:
Bootstrap som används för att skapa responsiva applikationer som även är mobilvänliga "Mobile-first".
Materialize som är likt Bootstrap i att dem använder en liknande struktur uppbyggnad och att responsivitet är i fokus. Den stora skillnaden är att Materialize är en implementering av Material Design och följer dem strikt (Designprinciper skapade av Google).
Bulma som fokuserar på att vara en lättförståelig och väldokumenterat ramverk som fungerar som de flesta andra ramverk. Bulmas klassnamn är enkla att förstå och dokumentationen är utförlig vilket hjälper nya användare.

Fördelarna med ramverken är bidragandet av tidsbesparande och struktur. En anledning till att utvecklare använder sig av CSS-ramverk kan vara ifall en konstant design över flera projekt är en viktig aspekt. Ramverken bidrar till att utvecklare kan snabbt börja arbeta med funktionaliteten av ett projekt utan att fastna på att bygga en grund och en design.

Nackdelar med CSS-ramverk är inlärning samt originalitet. Det ena problemet är att man måste man först lära sig ramverkets syntax. Syntaxen är även olika bland ramverken på grund att de har olika skapare, vilket blir problematisk tidsmässigt ifall man vill byta ramverk.
Det andra problemet är ifall man vill att sin hemsida inte ska likna andra, de flesta ramverken kommer med en "fast" design för knappar, formulär element osv.

Nackdelarna med ramverk bidrog till att jag inte använde mig av något i detta projekt. Jag ville spendera mer tid åt att bygga en egen portfolio från grunden istället för att ta tid åt att lära mig ett ramverk som sedan skulle begränsa mig med designen. Jag kan dock se fördelarna med att lära mig några ramverk i framtiden för att förenkla mitt kodande.

SASS och LESS

SASS och LESS är CSS för-processorer, vilket betyder att de är skriptspråk som tillåter utvecklare att skriva kod i ett språk och sen sammanställa det till CSS.

Målet med för-processorer är att göra kodningsprocessen enklare, snabbare samt mer effektiv. De bidrar även med nya funktioner som CSS inte har vilket gör att man har mer möjligheter.

SASS och LESS har flera liknelser, båda tillåter användningen av variabler samt mixins (bidrar med återanvändning av kod). Båda kan även använda sig av loopar för att t.ex. ändra designen på en hemsida ifall specifika villkor gäller.
En skillnad är att SASS behöver en Compiler som använder "C++", ifall man använder sig av Node.js så måste man installera "Node compiler". LESS är dock skrivet i bara JavaScript vilket betyder att installera LESS är lika enkelt som att länka ett JS-bibliotek till sitt html document. Det är dock många som anser att SASS är den "bättre" för-processorn på grund av jämförelsen av funktionaliteten mellan SASS och LESS.

Anledningen till att jag undvek att använda mig av SASS eller LESS i detta projekt var för det första, inlärningskurvan. Att spendera massa tid åt att lära mig/ försöka förstå syntaxen när jag samtidigt lär mig om React kändes inte som en bra idé. För det andra, kände jag mig inte helt felfri med att använda vanlig CSS än. Jag gör fortfarande missar och känner mig lite osäker på CSS fortfarande. Jag lär mig fortfarande och vill känna mig säker på grunderna innan jag lär mig genvägarna och ytterligare funktionalitet.

Grundläggande JavaScript Syntax

```
const capitalize = (s) => {
    if (!s) return '';
    return s.charAt(0).toUpperCase() + s.slice(1);
}
```

Ovan är en kort JavaScript kodexempel från mitt projekt. Jag skapade en funktion som ska ta en sträng som argument, sen kör en "if statement" på argumentet för att se ifall det är en sträng. Ifall argumentet inte är en sträng, (if statement blev falskt) så får jag ut en tom sträng som resultat. Ifall det är en sträng, då används 3 sträng metoder som först isolerar den första bokstaven i strängen, omvandlar den till stor bokstav, och sen applicerar tillbaka den först på strängen. Resultatet blir då att den sträng som körs som argument i funktionen får en versal som första bokstav. T.ex.
`capitalize("stockholm")` kommer att resultera = Stockholm.
Jag använde denna funktion för att ändra en sträng från ett api som jag manuellt inte kunde ändra på.

JavaScriptversioner och dess Kompatibilitet

ECMAScript (European Computer Manufacturers Association) är standarden för skriptspråk som JavaScript. Ända sedan den sjätte versionen av ECMAScript (ES6) publicerades 2015 så har det publicerats en ny version varje år. De tre senaste är:

ES9, vilket introducerade bland annat “The spread operator” som tillåter kopiering av objekt egenskaper.
ES10, vilket introducerade bland annat “flat()” metoden som skapar en ny array där alla underliggande arrays är ihopsatta med resten av arrayen till det specifika “djupet”. T.ex. (const arr = [0, 1, 2, [[3, 4]]]; arr.flat() === [0, 1, 2, [3, 4]];
arr.flat(2) === [0, 1, 2, 3, 4];)
ES11, vilket introducerade bland annat en ny operatör (??). Vid användandet av ?? kommer värdet på höger sida av operatören resulterade ifall det vänstra värdet antingen är *undefined* eller *null*. Annars kommer det vänstra värdet resulterade varje gång. T.ex. (*undefined* ?? "string" === "string")

Dessa versioner är även fullt Kompatibla på de senaste versionerna av webbläsarna som; Chrome, Firefox, Microsoft Edge och Opera.
Safari har några saker som det inte stöder, bland annat; delar av ES11:s *BigInt* som är en “primitive type” samt även en del av ES9:s tillägg till *RegExp*.
Microsoft Edge version 18 som introducerades i en uppdatering av windows 10 (2018) är näst intill helt inkompatibel med ES9, 10 och 11. Edge 18 är dock nästan helt kompatibel med äldre versioner av ES. Och sist och även minst använd är Internet Explorer som är nästan helt inkompatibel med de senaste 5 versionerna av ES.

Med tanke på att majoriteten av internetanvändare använder sig av Chrome samt Safari och att jag inte använt mig utav *BigInt* eller *RegExp* i min portfolio, betyder det att nästintill ingen kommer ha några problem med att granska på min portfolio.

Hur JavaScript körs i webbläsaren

När en webbläsare kör javascript så körs en sak i taget. Det betyder att när en funktion körs så hamnar den först i “kön” av det som ska hända. Funktionen ligger också kvar i kön ända tills vi har fått ett slutresultat eller returnering från funktionen. Detta betyder att kön kommer växa ifall en funktion leder till en annan o.s.v ända tills den första returneringen infaller, då kommer den funktion som är “först” (senaste tillagd) i kön att försvinna. På grund av det så kan en så kallat “Blowing the stack” hända. Genom att en funktion t.ex. leder till sig själv och aldrig ett resultat. Vilket leder till att kön bara växer och växer tills webbläsaren fått nog och avslutar det som körs och visar varningar när funktionerna når max-gränsen av kön. Jag har sett till att undvika skapa funktions flöden som aldrig resulterar i ett resultat (return).

JavaScript-Bibliotek

Javascript-bibliotek är förskriven Javascript kod som tillåter förenkling av utvecklingen i applikationer. Dem liknar CSS-ramverk i med att dem är skapade av människor som vill bidra med genvägar samt lösningar med komplex kodning, för att skapa bättre förutsättningar för framtida utvecklare.

Det finns JS bibliotek för det mesta man kan tänka sig. Animationer, GUI relaterade, applicering av kartor, bildhantering o.s.v.
Några av de mest använda biblioteken är:
Chart.js som fokuserar kring visualisering av data genom diagram och tabeller.
Anime.js som fokuserar kring skapandet av livliga animationer.
jQuery som är designat för att förenkla HTML DOM manipulation.

Ett bra sätt att veta ifall ett JS bibliotek är värt att använda är att ta en titt på dokumentationen. Är den omfattande och tydlig? Då kan det vara värt att lära sig mer om den och kanske applicera den i sina projekt där den passar.

Jag använde inte mig utav något JavaScript-Bibliotek på grund av att jag inte börjat lära mig något än. Jag var intresserad av att lära mig Anime.js för att använda i min portfolio, men kände att jag behövde fokusera mer tid åt att skriva kod i React för att bli klar med portfolion. Jag kommer dock i framtiden uppdatera portfolion samt tidigare projekt när jag börjar lära mig Anime.js.

Många debatterar ifall React är ett JS bibliotek eller ramverk, skaparna kallar det för ett bibliotek, men det används som ett ramverk. Vissa anser att på grund av användandet av JSX som språk i React så räknas det som ett ramverk. Och andra tycker att React arbetar som ett ramverk men fungerar som ett bibliotek under ytan. Ett citat jag sett vid jämförelser mellan Angular och React är "*Angular lets you control Modal, View and controller (MVC) in front-end of any web application while React provides just the view part*". Tanken är att React fokuserar på det synliga lagret istället för helheten vilket betyder att det ska räknas som ett bibliotek. Ett annat är "*If you have included a framework, you have to follow the rules and guidelines of that framework to build your application*".
Jag kommer räkna React som ett ramverk i denna redogörelse för att de åsikter jag sett mest av, samt dem jag håller med om anser React som ett ramverk.

JavaScript-Ramverk

JavaScript ramverk används som skelettet för applikationer, dem tillåter utvecklare fokusera mer på att skapa mer avancerade gränssnitt och mindre på kodstruktur genom att bidra med regler och riktlinjer som man måste följa.

Några exempel på JS ramverk är; **Angular.js**, **React.js** och **Vue.js**.

Angular.js fördelar; Ändringar bland data i applikationen ändras direkt hos det användaren ser, Angular tillåter utvecklare att skapa appar i MVC arkitekturen enklare.
Nackdelar; Utvecklare måste förstå MVC för att använda Angular, kan vara svårt att lära sig.

Reacts fördelar; Användandet av ett virtuellt DOM tillåter snabba uppdateringar på delar av en applikation vilket bidrar till skapandet av dynamisk UI. Ett exempel är Facebook, när man använder chattfönstret kan nyhetsflödet uppdateras i bakgrunden utan att störa resten av applikationen. React har även ett stort community, vilket bidrar till mycket frågor och svar inom allt angående React. Snabb utveckling, vilket bidrar till att man kan uppnå mycket med React.
Nackdelar; Dock på grund av den snabba utvecklingen av React så har dokumentationen hamnat efter en del, vilket påverkar inlärningen av det nya inom React.

Vues fördelar; Ett otroligt litet framework som är snabbt att ladda ner och installera. Vue använder sig också av ett virtuellt DOM för snabba uppdateringar. Vue är lätt att lära sig ifall man har grundkunskap inom HTML, CSS och JavaScript.
Nackdelar; På grund av att stor popularitet av Vue är från Kina så är en stor del av innehållet och diskussioner om Vue skrivet på Kinesiska. Vilket påverkar när man söker efter saker om Vue, då kan man få problem med att tyda förklaringar och instruktioner på forum. Begränsningen av plugins i jämförelse med React t.ex. är är även ett problem.

Jag ser stora fördelar med att jag lär mig React som mitt "första ramverk", speciellt med att för det första är en av de mest populära ramverk/bibliotek som finns. Det andra för att det fortfarande växer, längre fram kommer jag ha möjligheten att snabbt skapa dynamiska projekt med hjälp av React.

DOM

I grunden är en hemsida ett HTML dokument där strukturen är uppbyggd och stilen formad av CSS. Webbläsaren skapar en representation av HTML dokumentet som kallas Document Object Model (DOM) som bidrar med metoder som kan användas för att drastiskt ändra på den. DOM:en kan integreras och ändras på detta sätt med hjälp av JavaScript.
T.ex.

```
let largeBlueText = document.getElementsByTagName("h1")[0];
largeBlueText.style.color = "blue";
```

Genom att använda sig av "document" selekterar jag DOM:en, "getElementsByTagName" är en av metoderna som kan interagera med DOM:en. Med metoden bestämmer koden att "largeBlueText" är den första h1 elementet som finns inom DOM:en, och sedan ändrar dess färg till blå.

Det finns en till slags DOM som kallas "The Virtual DOM" vilket är en slags representation av HTML DOM:en jag nyss nämnt. Den Virtuella DOM:en används av React som en kopia av HTML DOM:en där React kör alla sina beräkningar istället för i den "riktiga DOM:en" där ändringar är långsammare. Den Virtuella DOM:en kan ta emot React element som kan ändras på innan dem konverteras från React komponenter med hjälp av komponenternas "state".

Detta fungerar genom att när en komponents state ändras så åter renderas den igen, vilket konverterar den till ett element som sätts in i den Virtuella DOM:en där elementet jämförs och uppdateras genom Reacts “diff algorithm”. Komponenterna kan ändras mycket genom deras state, t.ex. färg och storlek.

Fördelen med den Virtuella DOM:en är just att enskilda “delar” av en hemsida uppdateras när de ändras. Som jag nämnde tidigare med ett exempel om hur facebook fungerar i “Reacts fördelar” under “JavaScript-Ramverk”. Uppdatering kan ske i “bakgrunden” av andra delar på en hemsida utan att påverka det man aktivt använder. I min portfolio påverkas “Virtual DOM” på detta sätt två gånger. Först när jag visar upp “Openweathermap” api:et. Efter en “fetch” görs när landningssidan laddas, där all information från api länken hämtas och när det väl har hämtats ändras innehållet i en komponent och dess state med den hämtade datan vilket ändrar variablerna jag använt. Och sedan kör en ny rendering på innehållet vilket React jämför i Virtual DOM och ändrar det som skall.
Det andra är när meddelandet “Did you like my portfolio” ändras samt “Yes!” knappen försvinner, vilket är lite mer simpelt men fungerar under samma princip.

AJAX

AJAX är ett koncept, en metod för att växla data med en server/ databas där datan som hämtas kan uppdatera delar av en webbsida utan att behöva ladda om hela sidan. Tidigare har “XMLHttpRequest” varit en av de alternativ för att använda sig av AJAX. Fetch API är en inbyggt verktyg i JavaScript som är ett nytt och modernt alternativ till XMLHttpRequest. Nackdelen med Fetch är att ingen version av Internet Explorer stöder Fetch och tidigare versioner av andra webbläsare kan uppleva problem med Fetch. Detta påverkar de som har klienter/ användare som använder dessa webbläsare, vilket gör att Fetch inte fungerar som alternativ för vissa.

Jag använder mig av AJAX i portfolion när jag använder Fetch verktyget för att hämta data från Api:et *Openweathermap*. Med tanke på att min portfolio kommer troligen mest ses över av andra som använder moderna webbläsare så är Fetch ett bra alternativ för min portfolio.

HTTP 1.1 / 2.0

HTTP 1.1 är den uppdaterade versionen av HTTP 1.0 och har varit den metoden som använts när webbläsare skickar och hämtar/ tar emot information över nätet mellan servrar och klienter i många år. HTTP 1.1 är dock begränsad i att bara ett “request” per TCP (Transmission Control Protocol) länk kan bearbetas, vilket tvingar webbläsare att använda flera TCP länkar för att bearbeta flera requests ifall de behövs samtidigt. Vilket saktar ner nätverksprestanda genom att hemsidor som behöver skicka flera requests använder sig av mer nätverksresurser och ställer andra väntande requests i kö. Det är här HTTP 2.0 kommer in, eftersom fler hemsidor skapas och uppdateras som behöver

tillgång till mer resurser, fokuserade skaparna av HTTP 2.0 effektivisering av hur information kommuniceras mellan servrarna och alla klienter. Ett sätt som som HTTP 2.0 förbättrar är att det använder bara en TCP koppling för att dra ner på användningen av nätverks resurserna medan HTTP 2.0 tillåter användandet av flera kallade “strömmar” där flera requests kan bearbetas.

När jag använder mig av Fetch i min portfolio skickar det ett request med api nyckeln till *Openweathermap*, det bearbetas och servern skickar sedan ett “response” med informationen ifall allt stämmer i requestet. Fetch jobbar asynkront, vilket betyder att medan servern tar emot mitt request, bearbetar och skickar ett response, fortsätter fetch koden att köras igenom. När serverns respons väl tagits emot så körs koden igen men med datan hämtat från api databasen. På grund av att min portfolio bara skickar ett requests är det fortfarande funktionellt att den använder HTTP 1.1.

Hemsidan

http://viktorhultman.surge.sh/