Mittuniversitetet

MID SWEDEN UNIVERSITY

Projektrapport

Projektuppgift

MITTUNIVERSITETET

DSV Östersund

Examinator	Erik Ström	Erik.Strom@miun.se
Handledare	Robert Jonsson	Robert.Jonsson@miun.se
Författare	Sebastian Strindlund	sest1601@student.miun.se
Utbildningsprogram	Programvaruteknik, 180hp	
Kurs	DT0146G, Webbprogrammering med HTML5, CSS3 och JavaScript	
Huvudområde	Datateknik	
Termin, år	HT, 2017	

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Enligt projektbeskrivningen på moodle[1] så ska en webbapplikation som består utav en HTML-sida, extern stilmall och JavaScript implementeras. Layouten för sidan har vissa krav på att animationer ska användas, element som ska användas, hur vissa delar av sidan måste se ut samt krav på att inte använda JavaScript eller style inline i HTML-koden.

1.2 Syfte

Projektet lägger fokus på JavaScript eftersom målet med projektet är att applikationen som implementeras skall mäta användarens skrivhastighet och träffsäkerhet vid användning av tangentbord. Problemen som behöver lösas för användarinställningar är att kunna ladda in texter tillsammans med titel, författare, språk och även kunna använda sig utav dessa på ett bra sätt, skapa en textväljare som fylls med alternativ från inlästa texter och även uppdatera sidan med text som väljs, alternativ för att endast visa svenska eller engelska texter och ignorera versaler vid inmatning. Dessutom behövs lösningar för tidtagning, avläsning utav användarens inmatning via tangentbord, uträkning av skrivhastighet och träffsäkerhet.

2 Genomförande

2.1 Utrustning

- *Operativsystem: Windows 10 Pro.*
- *Text editor: Visual Studio Code Version 1.17.2.*
- *Fonts: https://fonts.google.com/*
- *Paint 3D.*
- *Webbläsare: Chrome Version 61.*
- *Chrome extensions: Livepage.*
- *Validator: www.validator.w3.org*
- *SFTP klient: WINSCP*
- *GIF verktyg: screentogif*

2.2 Tillvägagångssätt

Till att börja med så skapades en enkel layout med HTML utan någon funktionalitet enligt projektbeskrivningens rubrik ”3.1 Basimplementation” för att få en översikt av webbapplikationen. Därefter skapades animationen för applikationens rubrik med hjälp av CSS keyframes[2[. Animationen kallas genom `animation: colorChange 20s infinite;` vilket betyder att animationen körs i 20 sekunder och börjar om oändligt många gånger. Animationen är av ett simpelt slag och ändrar endast färg varje 10% av den totala animationstiden.

För att lösa problemet med att ladda in texter från en XML-fil användes Fetch API[3] istället för XMLHttpRequest[4] som tipsades om i projektbeskrivningen eftersom att fetch liknande XMLHttpRequest men har ett kraftfullare och mer flexibelt set av funktioner. Datan som tas emot av fetch() blir tyvärr endast en lång sträng som innehåller all text som fanns i XML dokumentet som datan hämtas från. För att enkelt lösa detta kan ett ”dummy” XML element skapas med Document.createElement()[5] för att sedan sätta detta elements inre HTML till att vara den sträng med XML data som togs emot. Tack vare detta kan nu innehållet från texten enkelt filtreras med hjälp av Document.getElementsByTagName()[6] som returnerar en samling av alla elements som hittats. Nu kan enkelt JavaScript objekt skapas som innehåller titel, författare, språk och text som hör ihop skapas. Därpå

skapades en eventlistener[7] på språkväljaren genom att först hitta språkväljarens HTML-element med Document.querySelector()[8] och sedan läggs eventlistener till på det elementet med eventTarget.addEventListener()[9] när detta är gjort kommer en definierad funktion att köras varje gång engelska eller svenska väljs. När det väl görs så hittas först språkväljarens element med querySelector(), därefter töms den på alternativ, alla objekt med texter loopas igenom och kontrolleras ifall textens språk stämmer överens med det valda språket, om så är fallet så läggs den textens titel till bland språkväljarens alternativ.

Nu när texter kan laddas in så verkade det lämpligt att lösa användarens inmatning och användarens progression av texten. Först och främst så skapades 3st CSS klasser, ”nextChar” som visar vilken tangent användaren bör klicka på genom en blinkande CSS animation som skapats med keyframes, ”failedchar” som ger texten röd färg och ”completedChar” som gör texten grå så att användaren kan se hur långt den kommit på ett bättre sätt.

Därpå delades hela texten upp med spans[10] runt varje bokstav eftersom att varje bokstav kommer att behöva kunna stylas individuellt. Detta gjordes genom att dela upp den aktuella strängen på varje tecken till en array med string.split()[11] och sedan lägga till ett span runt varje bokstav. Sedan används querySelector för att hitta elementet som användaren matar in sin text i och en eventListener av typen ”keypress”[12] som endast reagerar när tangenter som genererar ett tecken trycks ned samt en eventListener av typen ”keydown”[13] för att hålla koll på om användaren trycker backspace så att detta kan ignoreras. Varje gång användaren trycker på en tangent nu kallas en definierad funktion som kommer att jämföra bokstaven användaren tryckt på mot bokstaven som finns på aktuellt index för texten användaren ”spelar mot” just nu.

Skriver användaren rätt så plussas index på så att index är korrekt nästa gång funktionen kallas. Dessutom uppdateras progressionen genom att querySelector hittar det element som just nu är ”nextChar”, tar bort den klassen från elementet med element.classlis[14], lägger till klassen ”completedChar” och sist men inte minst ges nästa bokstav klassen ”nextChar”. Skulle användaren däremot skriva fel bokstav så får nuvarande bokstav klassen ”failedChar”, ett errorljud spelas upp[15] , antal errors plussas på och användaren får helt enkelt försöka på samma bokstav ännu en gång.

Om användaren har bockat för att ignorera versaler så kontrolleras det med en enkel if-sats och där inuti görs tecknet användaren matat in samt textens aktuella tecken om till gemener med string.toLowerCase()[16] innan jämförelsen av tecknen.

För att kunna räkna ut Gross WPM och Net WPM utgås det enligt projektbeskrivning från att den genomsnittliga ordlängen är 5 och så används formlerna

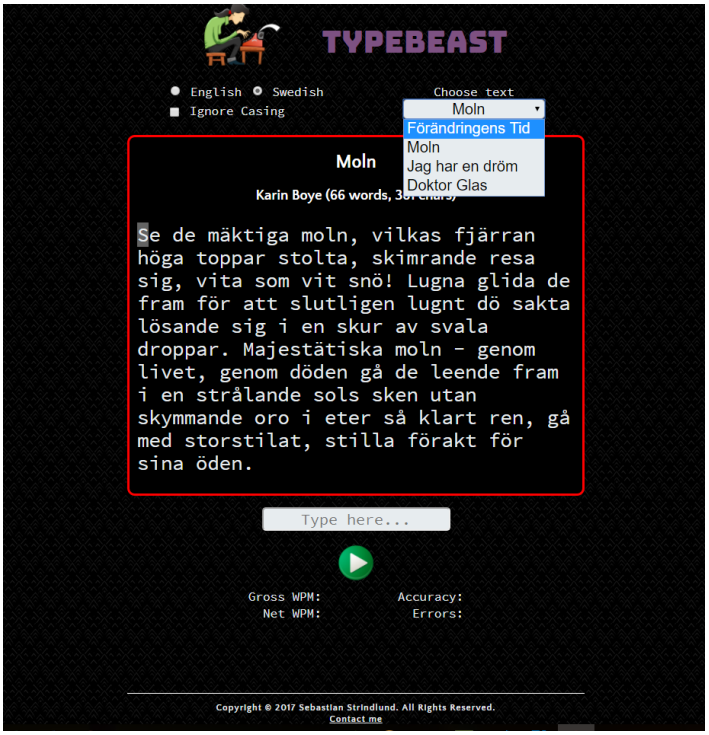
`gross = (((correctChars.length + errors.length) / 5) / elapsed_minutes);` och

`net = (gross - (errors.length / elapsed_minutes));` även dessa enligt projektbeskrivningen. För att kunna ha någon form av tid så startas en timer när användaren klickar på play med setInterval()[17] som varje sekund plussar på tiden med en sekund och uppdaterar statistiken enkelt visuellt genom att bara hitta elementen med querySelector och ersätta dess text med den nya statistiken. Denna timer tas bort med clearInterval()[18] när användaren klickar på stop.

För att lägga upp projektet på http://studenter.miun.se/~sest1601/projekt/ användes en enkel SFTP-client[19] för att överföra filerna.

3 Resultat

Resultatet syns i figurerna. Kolla även på ”demostration.gif” i projektets rot-mapp.



Diskussion

Som det går att se i resultatet så är nu webbapplikationen implementerad. Den har animation i rubriken, vilket faktiskt inte var speciellt svårt att få till med keyframes i CSS. Det verkar gå att göra väldigt avancerade animationer om man även tar hjälp av JavaScript, jag valde dock att endast göra så att min rubrik växlar mellan färger. Att ladda filer med hjälp av fetch var till en början väldigt krångligt, fick sitta ett bra tag och konsol logga allt för att förstå hur det egentligen fungerade och förstod snart att det inte var möjligt att hämta filen som ett dokument utan istället fick man en lång sträng med allt innehåll från dokumentet. Kom då på lösningen att skapa ett ”dummy” XML-element där man sätter in all text i elementets inre HTML så att man får ganska precis som ett vanligt XML dokument och kan använda document.get funktionerna. Fungerade mycket bra så det var jag väldigt nöjd med.

Att jämföra användarens inmatning mot den aktiva texten var däremot betydligt enklare än jag hade trott, JavaScript är otroligt bra för det uppdraget. Valde att använda mig av olika klasser som läggs till och tas bort för att hålla koll på hur användarens progression ser ut, vet inte om det är ett bra eller dåligt sätt egentligen men det fungerade väldigt bra tillsammans med querySelector.

För att räkna tid kanske jag borde ha använt mig av JavaScripts egna funktioner för att hämta tid men jag fick för mig att det skulle vara ”bättre” om jag hade en intervall som bara uppdaterade tiden med 1 sekund samt uppdaterade statistiken i det intervallet istället för att uppdatera på varje tangenttryck som användaren gör.

Jag är mycket nöjd med resultatet av uppgiften och tyckte att det var riktigt skoj att göra! Hade dock gärna sett att man fritt fått bestämma layouten.. Tycker att jag har fått lära mig väldigt mycket om Javascript i uppgiften och kursen men ser fram emot att lära mig ytterligare om Javascript.

Referenser

[1] Mittuniversitetet, "Projektuppgift", 2017. [Online]
Tillgänglig: https://elearn20.miun.se/moodle/mod/resource/view.php?id=360009
[Åtkomst: 18 December 2017]

[2] Mozilla "@keyframes", 2017. [Online]
Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/CSS/@keyframes
[Åtkomst: 18 December 2017]

[3] Mozilla "Using Fetch", 2017. [Online]
Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
[Åtkomst: 18 December 2017]

[4] Mozilla, "XMLHttpRequest", 2017. [Online]
Tillgänglig: **https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest**
[Åtkomst: 18 December 2017]

[5] Mozilla, "Document.createElement(), 2017. [Online]
Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement
[Åtkomst: 18 December 2017]

[6] Mozilla, "Document.getElementsByTagName()", 2017. [Online]
Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementsByTagName
[Åtkomst: 18 December 2017]

[7] Mozilla, "EventListener", 2017. [Online]
Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/API/EventListener
[Åtkomst: 18 December 2017]

[8] Mozilla, "document.querySelector", 2017. [Online]
Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector
[Åtkomst: 18 December 2017]

[9] Mozilla "EventTarget.addEventListener()", 2017. [Online]
Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener
[Åtkomst: 18 December 2017]

[10] Mozilla "", 2017. [Online]
Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/span
[Åtkomst: 18 December 2017]

[11] Mozilla "String.prototype.split()", 2017. [Online]
Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/split
[Åtkomst: 18 December 2017]

[12] Mozilla "keypress", 2017. [Online]
Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/Events/keypress
[Åtkomst: 18 December 2017]

[13] Mozilla, "keydown", 2017. [Online]
Tillgänglig: **https://developer.mozilla.org/en-US/docs/Web/Events/keydown**
[Åtkomst: 18 December 2017]

[14] Mozilla, "Element.classList", 2017. [Online]

Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/API/Element/classList
[Åtkomst: 18 December 2017]

[15] Mozilla, "Cross-browser audio basics", 2017. [Online]
Tillgänglig: https://developer.mozilla.org/en-US/Apps/Fundamentals/Audio_and_video_delivery/Cross-browser_audio_basics
[Åtkomst: 18 December 2017]

[16] Mozilla, "String.prototype.toLowerCase(), 2017. [Online]
Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/toLowerCase
[Åtkomst: 18 December 2017]

[17] Mozilla, "WindowOrWorkerGlobalScope.setInterval()", 2017. [Online]
Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setInterval
[Åtkomst: 18 December 2017]

[18] Mozilla, "WindowOrWorkerGlobalScope.clearInterval()", 2017. [Online]
Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/clearInterval
[Åtkomst: 18 December 2017]

[19] Mozilla, "Free SFTP for Windows", 2017. [Online]
Tillgänglig:https://winscp.net/eng/docs/free_sftp_client_for_windows
[Åtkomst: 18 December 2017]