

Laboration 2



1DV403 – Webbt teknik 1

Laboration 2, Labby Mezzage



Observera

För att bli godkänd på laborationen ska din källkod följa den standard vad det gäller kommentering, val av variabelnamn m.m. som gått igenom på föreläsning. Ditt JavaScript måste även fungera enligt de krav som finns beskrivna i denna laborationshandledning.

För att kunna genomföra denna laboration måste du vara förberedd **innan** du går på handledningstillfället, det är inte förbjudet att göra laborationen i förväg. Tid att sätta sig in i hur JavaScripten ska fungera eller skrivas kommer **inte** att finnas under den lärlarleda handledningstiden.

Vid fuskmisstanke lämnas misstankar samt berörda dokument över till universitetets disciplinnämnd.

Läs igenom hela laborationshandledningen **innan** du påbörjar laborationen.

Laborationen ska lösas enskilt.



Läsanvisning

Föreläsninganteckningar #1-6 och dess hänvisningar till kurslitteraturen.

Inledning

Kommande laborationer ställer ganska höga krav på dig som laborant. Du kommer till stor del vara tvungen att leta upp information själv och fundera ut egna lösningar på problem som ställs.

I denna laboration kommer du att skriva en enklare meddelandehanteringstjänst.

Mål

Efter du genomfört och blivit godkänd på denna laboration kommer du kunna hantera skapandet av objekt i javascript. Du kommer vidare att ha goda kunskaper i att koppla händelsehanterare och arbeta med DOM.

Genomförande

Utför laborationens uppgifter och moment samt dokumentera vad du kommer fram till på de olika delarna. Vid redovisning av laborationen ska du kunna besvara frågor om **hur** du har löst de olika delarna och **varför** de är lösta på det sätt du löst dem på.

När du anser dig vara klar med laborationen, kontrollera att din källkod uppfyller laborationens **samtliga** krav.

Uppgift 1 – Labby mezzage

Denna laboration består av en enda uppgift som är uppdelad i steg. Du som känner dig hemma på javascriptprogrammering behöver inte följa stegen i detta laborationspm utan du kan själv välja hur du vill implementera laborationen. **Observera dock att samtliga krav som finns angivna i slutet av laborationshandledningen måste vara uppfyllda.**

Uppgiften går ut på att skapa en enklare meddelandetjänst där vi kan lägga till och ta bort meddelanden. Detta ska ske på ett för användaren enkelt och smidigt sätt. Dock sätter vi begränsningen på denna laboration att man måste ha javascript aktiverat för att den ska fungera. Senare laborationer kommer att ha kravet på sig att fungera även utan javascript.

Det är upp till dig själv att skapa layouten för sidan men du ska anstränga dig för att göra en grafiskt tilltalande applikation.

Tillsammans med denna handledning finns en demonstrationsfilm som visar hur applikationen ska fungera. Du kan även läsa om detta i de funktionella kraven.



Moment 1 – Kom igång

Denna uppgift är av lite större karaktär än vad du kanske är van vid. Nyckeln till att lösa större uppgifter som denna är att hela tiden ta små steg och testa applikationen ofta.

Det första vi bör göra är att skapa våra dokument och ett skal för vår applikation.

Det finns några krav för hur applikationen ska vara uppbyggd rent kodmässigt. Dessa krav hittar du under "Projektkrav". Läs igenom dem!

Vi ska kunna instansiera objekt utifrån en konstruktorfunktion Message. Detta kan vi se som att vi ska ha en "klass" Message som ska vara en behållare för våra meddelanden. Du bör därför skapa en konstruktorfunktion "Message" och lägga den i en separat js-fil som du döper till message.js. Konstruktorfunktionen låter vi vara tom än så länge.

Efter detta ska du skapa ett statiskt objekt (objektliteral) i vilket du lägger själva applikationens "motor". Detta statiska objekt lägger du i en egen js-fil.

Se till att länka in js-filerna i ditt HTML-dokument och koppla windowobjektets onload-event till en metod på ditt statiska objekt (exempelvis init). Denna metod blir nu den metod som kommer att köras då sidan laddats klart och också den metod i vilken du kan koppla en del event för att komma igång.

Testa att allt fungerar genom att skapa en alertruta i den metod som anropas av window.onload.

Moment 2 – Message-objektet

Innan vi börjar skapa texttrutor i vårt HTML-dokument ska vi se till att vi får rätt på vår meddelandeklass. Vi kan konstatera att ett meddelande kommer att bestå av två saker. Själva meddelandetexten och ett datum. Vi vill undvika att exponera dessa egenskaper direkt utan låter dessa vara ”privata” varför vi skapar getters och setters (privilegerade metoder) så att vi kan komma åt egenskaperna. Vi skapar därför två egenskaper i vår konstruktorfunktion.

Förutom dessa egenskaper behöver vi ett antal metoder som vi kan anropa på vårt objekt.

Dessa metoder läggs på prototyp-objektet som tillhör konstruktorfunktionen.

Följande metoder rekommenderas:

```

1  function Message(message, date) {
2
3      this.getText = function() {
4          return message;
5      }
6
7      this.setText = function(_text) {
8          message = text;
9      }
10
11     this.getDate = function() {
12
13     }
14
15     this.setDate = function(_date) {
16
17     }
18 }
19
20
21 Message.prototype.toString = function() {
22     return this.getText() + " (" + this.getDate() + ") ";
23 }
24
25 Message.prototype.getHTMLText = function() {
26
27 }
28
29 Message.prototype.getDateText = function() {
30
31 }
32
33

```

Medlemsmetoder:

toString()	En strängrepresentation av objektet.
getText() *	Hämtar meddelandetexten
setText() *	Sätter meddelandetexten
getDate() *	Hämtar datumet
setDate() *	Sätter datumet
getHTMLText()	Hämtar texten med \n utbytt mot

* Privilegerade metoder

```

Message.prototype.toString = function() {
    return this.getText() + " (" + this.getDate() + ") ";
}

```

Observera att du hela tiden måste gå via getters och setters för att kunna komma åt meddelandetexten och datumet.

När du skapat skalet till dina Message-objekt är det dags att testa dessa.

I den funktion som anropas av window.onload kan du nu testa att instansiera nya objekt och skriva ut dessa genom att använda new-operatorn och alert(). Om du implementerat toString korrekt så kan du skriva ut objektet direkt med alert.

```

var mess = new Message("Testmeddelande", new Date());
alert(mess); // Använder toString för utskrift
alert(mess.getText()); // Skriver enbart ut texten.
mess.setText("En annan text");
alert(mess); // Skriver ut meddelandet med ändrad text

```

Testa så att allt fungerar väl och laborera lite med dina objekt. Se till att du kan ändra texten på skapade objekt och att du kan läsa ut tiden.

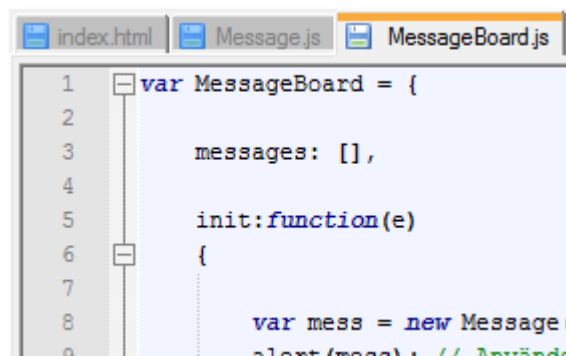
Lab2

Labby Mezzage
1DV403 – Webbteknik I

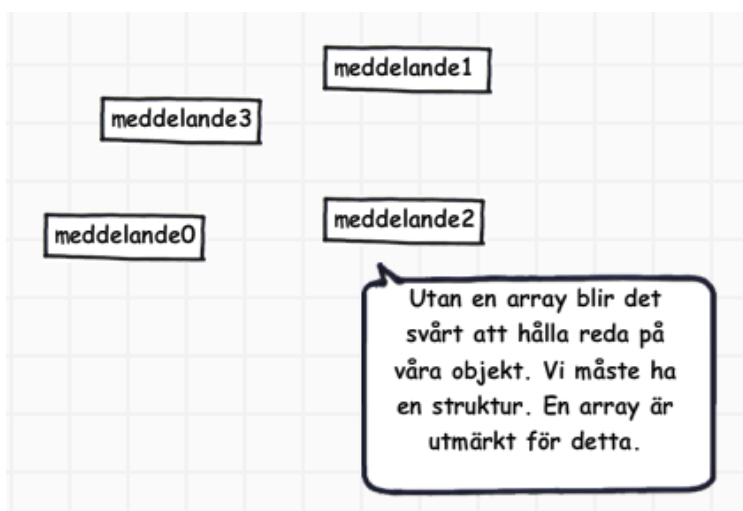
Moment 3 – Message-listan

Nu är det dags att se till att vi på vårt statiska objekt, vår applikation, kan hålla en lista med flera messageobjekt. Skapa därför en array som en egenskap på vårt statiska objekt. Döp denna lämpligtvis till **messages**.

Testa att lägga till objekt till arrayen. Undersök gärna metoden **push** som finns att tillgå på arrayer.



```
1 var MessageBoard = {  
2  
3     messages: [],  
4  
5     init: function(e)  
6     {  
7  
8         var mess = new Message(  
9             alert(mess); // Använd
```



Arrayens uppgift är att hålla reda på alla våra meddelanden. Arrayen kommer därför att schematiskt se ut så här:



Vi kan då enkelt komma åt texten på i det tredje meddelandet genom att skriva:

```
MessageBoard.messages[2].getText();
```

Moment 4 – Skapa HTML-kod

Nu börjar vi bli mogna att skapa HTML- och CSS-kod för vår sida. Skapa därför en layout som du är nöjd med och som innehåller ett lämpligt antal divtaggar och en `<textarea>` i vilken användaren kan skriva sitt meddelande. Använd CSS för att få ett attraktivt utseende. Under textfältet lägger du en knapp som användaren kan använda för att skicka meddelandet.

Lägg en div-tagga med lämpligt ID i vilken du sedan kan lägga ut dina meddelanden. Texten som skriver ut antal meddelanden behöver du inte skapa nu då den ska vara dynamisk.



Moment 5 – Skapa meddelande

Det är nu dags för det stora steget. Här kommer du inte att få så mycket vägledning utan din uppgift blir nu att koppla event till knappen så att meddelanden skapas då användaren klickar på den (onclick på knappen).

När användaren gjort detta ska ett nytt Message-objekt skapas. Detta objekt ska innehålla texten som användaren skrev i och den exakta tidpunkt då användaren skapade inlägget. Detta messageobjekt lägger du sedan till i arrayen med messageobjekt. (messages)

Nu är du en bra bit på väg.

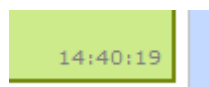
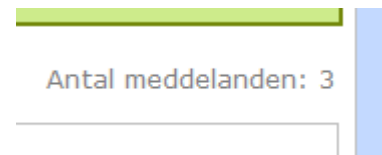
Tips. För att läsa ut texten från ett textfält så kan du använda egenskapen "value" som finns på textarea-objektet.

Moment 6 – Skriv ut meddelandet

Nu är det dags att skriva ut meddelandet så att användaren kan läsa det. Detta kan du göra på två sätt.

- 1) Radera alla befintliga meddelanden på sidan och skriv ut samtliga meddelanden i arrayen igen.
- 2) Lägg till det aktuella meddelandet sist i den div-tagga som innehåller dina meddelanden.

Du väljer själv hur du vill göra här men jag skulle rekommendera att skriva funktioner som gör att du kan göra på vilket sätt du vill. Glöm inte att uppdatera meddelanderäknaren som håller reda på hur många meddelanden som är utskrivna.



Se även till att skriva ut **tiden** för när meddelandet skapades.

RenderMessages:

Förslagsvis så skapar du en funktion (renderMessage) på ditt statiska objekt som ansvarar för att rita ut **ett** meddelande. Du kan sedan enkelt skapa en funktion (renderMessages) som loopar igenom arrayen och skriver ut samtliga meddelanden med hjälp av renderMessage.

```

42 renderMessages: function() {
43     // Remove all messages
44     document.getElementById("messagearea").innerHTML = "";
45
46     // Renders all messages.
47     for(var i=0; i < MessageBoard.messages.length; ++i){
48         MessageBoard.renderMessage(i);
49     }
50 }

```

RenderMessage:

RenderMessage har som uppgift att skriva ut ett specifikt meddelande (messageID). Här kan du välja att använda innerHTML för att skapa dina DOM-noder, men det är starkt

rekommenderat att använda metoder så som document.createElement, appendChild, et cetera för att förenkla framöver. Det blir mer kod, men du kommer att tjäna på det i slutändan.

```

53 renderMessage: function(messageID)
54 {
55     // ...

```

Dock behöver du använda innerHTML för att göra utskriften av själva meddelandetexten:

```

// Message text
var text = document.createElement("p");
text.innerHTML = MessageBoard.messages[messageID].getHTMLText();
div.appendChild(text);

```

Testa nu din applikation så att du kan lägga till flera meddelanden. Du bör nu kunna välja om du vill anropa renderMessage för att skriva ut meddelandet sist, eller renderMessages för att skriva om alla meddelanden.

Moment 7 – Radera meddelanden

Nu är det dags att lägga till funktionalitet så att vi även kan radera meddelanden. Detta betyder att du får modifiera den kod som sköter skapandet av ett meddelande (renderMessage) så att den även lägger till en länk och en bild som användaren kan klicka på. Koppla onclick till länken och se till att i den funktion som är kopplad till onclick radera aktuellt meddelande från din array.

```

imgClose.alt="Close";
imgClose.onclick = function() {
    MessageBoard.removeMessage(messageID);
}

```

Radera sedan alla meddelanden på sidan och skriv ut dem igen genom att anropa renderMessages.

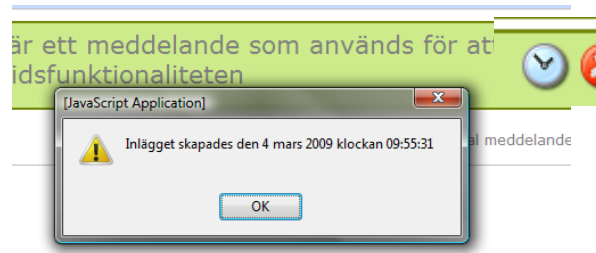
Testa så att du kan radera flera/alla meddelanden och lägga till nya. Se till att din meddelanderäknare hela tiden är uppdaterad.

Antal meddelanden: 3

Moment 8 – Visa tidsstämpel för meddelandet

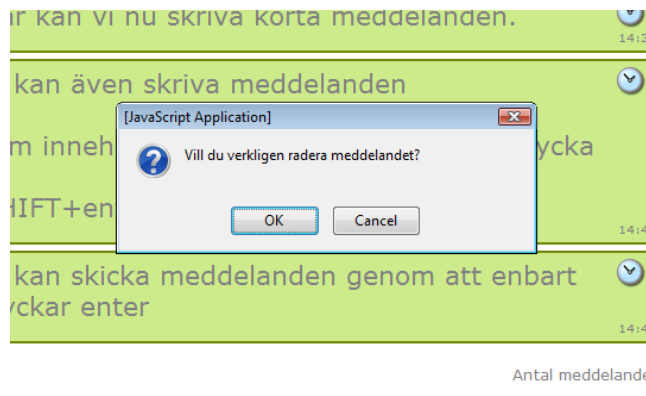
Lägg nu till kod för ännu en knapp på meddelandet. Denna knapp ska visa aktuell tidpunkt inklusive måndag, dag och tid i en alertruta.

Testa att allt fungerar.



Moment 9 – Färdigställ

Nu är vi nästan klara men vi har hittills utelämnat några krav som nu måste implementeras.



Det som nu ska göras är:

- Se till att användaren kan skicka meddelanden genom att trycka på entertangenten. Detta gör du genom att koppla en händelsehanterare till textfältets onkeypress-event.

```
ld.onkeypress = function(e) {
    if(!e) var e = window.event;
```

I händelsehanteraren får du sedan använda informationen som skickats till eventet (e) och läsa ut den tangent som användaren tryckt på med egenskapen **keyCode**

- När du fått detta att fungera ska du lägga till kod till testet så att användaren kan hålla nere SHIFT-tangenten utan att meddelandet skickas. Du kan ta reda på detta med `e.shiftKey`
- Nu ska du se till att byta ut alla radbrytningar som användaren skrivit mot `
`. Detta görs lämpligtvis i meddelandets `getHTMLText`-metod. Här kan vi använda reguljära uttryck

```
return this.message.replace(/\n\r/g, "<br />");
```

- När användaren raderar ett meddelande ska en varning först visas i vilken användaren kan välja att avbryta. Kika lite på `window.confirm()` så löser du nog detta kvickt.

Nu är du förhoppningsvis klar med applikationen. Testa den genom att skapa många och framförallt olika långa inlägg. Testa att radera och kontrollera tiderna. Fungerar allt?

Vad bra, kontrollera då kraven återigen på nästa sida. När det är gjort är du fri att lägga till extrafunktionalitet om du önskar. Kanske ska man kunna editera inlägg?

Labby mezzage



Här kan vi nu skriva korta meddelanden.

14:39:30

Vi kan även skriva meddelanden
som innehåller radbrytningar genom att trycka
SHIFT+enter

14:40:05

Vi kan skicka meddelanden genom att enbart
tryckar enter

14:40:19

Antal meddelanden: 3

Vi kommer även kunna radera våra meddelanden....

Observera räknaren som visar hur många meddelanden
som är skrivna.

skriv

Krav på uppgiften



Funktionella krav (boka av)

- ☐ Användaren kan skapa meddelanden genom att skriva dessa i textfältet och trycka på knappen "skriv"
- ☐ Användaren kan skapa meddelanden genom att skriva dessa i textfälten och avsluta med "enter"/"returtangenten"
- ☐ Användaren kan skapa radbrytningar i meddelanden genom att trycka SHIFT+ENTER
- ☐ Meddelandet presenteras med text och radbrytningar.
- ☐ Meddelandet innehåller en tidsstämpel på formatet (timmar:minuter:sekunder)
- ☐ Användaren kan radera ett inlägg genom att klicka på en, till meddelandet, tillhörande knapp.
- ☐ När ett meddelande ska raderas får användaren via en bekräftelsedialog göra valet om meddelandet ska raderas eller ej.
- ☐ Användaren kan genom att klicka på en, till meddelandet tillhörande knapp, få en dialogruta som presenterar datum och klockslag då meddelandet skapades.
- ☐ Applikationen fungerar och är testad i olika moderna webbläsare.



Icke funktionella krav (boka av)

- ☐ Applikationen är grafiskt tilltalande och tydlig



Projektkrav (boka av)

- ☐ HTML-dokumentet är validerat och godkänt enligt W3Cs rekommendation.
- ☐ Ingen javascriptkod är skriven direkt i HTML-dokumentet (varken i källkoden eller den genererade koden). Du får alltså **inte** ha kod liknande:
``
- ☐ document.write används inte
- ☐ Varje meddelande sparas i ett objekt instansierat från konstruktorfunktionen Message
- ☐ Meddelanden sparas i en array som alltid motsvarar de meddelanden som användaren ser.
- ☐ Hela applikationen är inkapslad i ett objekt.
- ☐ CSS-kod länkas in med ett externt dokument.
- ☐ Javascriptkoden sätter aldrig stilmallsinformation direkt utan CSS-klasser används.



För en större utmaning

(Detta steg ger inte högre betyg på denna laboration som enbart bedöms med U/G men kommer att hjälpa dig vid projektet)

Om man väljer att skapa uppgiften i ett statiskt objekt så märker man fort en begränsning, det går inte att köra flera instanser av meddelandetjänsten i samma HTML-dokument samtidigt. Har du ambition att satsa på högre betyg i denna kurs kan det därför vara bra att försöka lösa detta problem redan nu och få ovärderlig kunskap som du får nytta av i betygssteget 5 i projektet.

Något som vore trevligt att göra är att lyckas skapa obegränsat antal instanser av applikationen i ett och samma HTML-dokument. T.ex. skulle man kunna skriva så här för att starta våra applikationer:

```
window.onload = function() {  
  
    new MessageBoard("board1");  
    new MessageBoard("board2");  
  
};
```

index.html:

```
<div id="container">  
    <!-- Här kommer vår första instans att köras -->  
    <div id="board1"></div>  
    <!-- Här kommer vår andra instans att köras -->  
    <div id="board2"></div>  
</div>
```

I fallet ovan så skapas applikationerna genom att ett nytt objekt av "klassen" MessageBoard instansieras. Som argument tas id:t på den DOM-node i vilken applikationen ska skapas. När applikationen väl är skapad, sköter den sig sedan själv.

När vi enbart haft en instans av samma applikation som exekverats i ett och samma html-dokument har vi tidigare kunnat förlita oss till att läsa av ID:et på det element som klickats för att få reda på vilken, exempelvis, skicka-knapp som användaren klickat på. När vi nu har flera instanser av samma applikation så kan vi inte göra på detta sätt då samma knapp i de olika spelen i så fall får samma ID vilket inte är tillåtet.

Det finns då ett snyggare och på alla sätt bättre sätt att hålla reda på vad som klickats, och det är att använda closures. Detta är dock mer komplext, och kan vara svårt att greppa så därför visar jag hur det skulle kunna se ut när man skapar skickaknappen i applikationens konstruktorfunktion:

```
/* Send-button */  
var that = this;  
var inputButton = document.createElement("input");  
inputButton.type = "button";  
inputButton.value = "skriv";  
inputButton.onclick = function(e) {that.sendMessage(); return false;}  
div.appendChild(inputButton);
```

Här tvingas vi använda en closure mot variabeln `that` som då kommer att peka på vår instans. Hade vi skrivit `"this"` direkt i funktionen så kommer den att i runtime referera till `"inputButton"`.

Se nu alltså till att inte förlita dig på ID:n i html-koden. För att styra utseendet lämpar sig klasser lika bra.

Hängde du med? I så fall löser du detta i ett nafs!

Lycka till!