Projektarbete i Datorteknik

Webbserver för media lagring

- Hur en webbserver implementeras från grunden

Författare: Mikael Andersson och Carl Johansen
Kurs: 2DT301
Termin: VT/HT 2015
Ämne: Datavetenskap

Sammanfattning

Det här projektet går in på hur man skall gå tillväga för att utveckla en så kallad *cloud* service för media lagring. Vilka funktionaliteter och krav som behövs. Dessa inkluderar vilka metoder i http protokollet servern skall hantera, samt hur servern skall hantera eventuella problem. Servicen som utvecklats uppfyller dessa krav med bra felhantering och stabilitet.

Innehåll

1 Introduktion

1.1 Bakgrund

I dagens samhälle är tillgängligheten till internet samt mängden data individen och företag behöver lagra stor. Ett sätt att använda den här kopplingen finns i formen av *Cloud* services. Dessa ger användaren tillgång till att lagra sina filer online, vilket har ett flertal fördelar som tillgänglighet och mindre krav på lagringskapacitet på deras lokala maskiner.
Färdiga tjänster erbjuds av ett flertal stora företag som Google drive, dropbox, oneDrive och iCloud för online media lagring.

1.2 Problemformulering

Vad det här projektet skall gå in på är hur en webbaserad service kan implementeras från grunden. Servicen kommer att bestå av tre stycken huvuddelar: En server som hanterar information och olika förfrågan, en klient som kommer att agera som ett *User Interface* för att skicka förfrågan till servern, och sista delen är hur kommunikationen kommer att ske mellan server och klient.

Server och klient kommer att skrivas helt från grunden, utan användning av färdigt *framework*. Istället kommer färdiga *APIs* i form av bibliotek användas för att få bättre kontroll och inblick över hantering av information.

Vad en service behöver för att fungera dock är en plattform där den kan vara uppe permanent. Att använda en persondator är därmed inte optimalt eftersom den spenderar datorkraft på annat samt är inte permanent uppe. Vad som behövs är därmed en dedikerad maskin för att driva servern.

1.2 Motivation

Anledning varför vi har valt att utveckla en *Cloud* service bottnar i ett flertal anledningar. Den första är att använda sin egen service betyder att den kommer vara skräddarsydd efter ens egna preferenser och krav.

Ett problem med färdiga tjänster är att större mängd data kräver ofta mer betalning, med egen hårdvara är det enklare att anpassa hur mycket data som går att lagra eftersom minne blir allt billigare.

Sista anledningen är att nätverkskommunikation och utveckling av system är ämnesområden där vi har haft lite utbildning i men vill öka kunskaperna.

1.3 Avgränsningar

Vad det här projektet inte kommer gå in på är säkerhetsaspekten i att lägga upp en server som är åtkombar online. För kommunikation är det http protokollet som har använts, eftersom det är en av de mest vanliga överförings protokollen. Alternativet är *https* som bygger på *http,* skillnaden är att den skickar data på en krypterad linje. Eftersom projektet ska handla om hur strukturen och vilka krav en *Cloud* service har så har det inte funnits direkt behov av *https.* Däremot skulle tjänsten publiceras bör det övervägas att växla protokoll från *http* till *https*.

2 Metod

Verktyg som användes:

- IntelliJ Idea [9]
- Webstorm [10]
- Raspberry PI 3
- Raspbian OS [8]
- Putty/Mac OS X Terminal

2.1 Metodbeskrivning

Det valda programmeringsspråket för servern är Java skrivet i IntelliJ Idea från Jetbrains. Java valdes p.g.a. att det är ett mångsidigt programmeringsspråk som har många tredjeparts bibliotek men även väldigt många inkluderade bibliotek som t.ex. com.sun.httpserver[1] som används i detta projekt. Till klientsidan användes även där Jetbrains, Webstorm för att skriva Javascript[4] och HTML[2] samt CSS[3].

Kommunikationsprotokollet som används i detta projekt är HTTP. Det är en av de vanligaste och mest använda protokollen som kommunicerar till en TCP-port vanligen port 80 men i detta arbete används port 8080.
Anledningen till att HTTP valdes är mycket p.g.a. det inkluderar de förfrågningarna vi ville använda. Exempel på dessa förfrågningar är GET och POST.

En Raspberry används för att hosta servern p.g.a. att den är liten och smidig och är lätt att gömma undan om man vill ha igång servern 24/7 samt att den är billig (ca 400kr).
Versionen av Raspberryn som används i detta projekt är Raspberry PI 3. Den stödjer både WiFi och Bluetooth vilket än en gång gör det enkelt att förvara utom synhåll och inget mer än en strömkabel kommer vara inkopplad till PIn.

3 Implementation

3.1 Server

Med server till klient kopplingen så är det viktigt att bestämma hur ansvaret skall vara bestämt för båda. Det är en viktig punkt vid utveckling att se till att servern skall göra all hantering av information. Klienten skall vara given så lite ansvar som möjligt, anledningen är att klientsidan av operationen är oskyddad och enklare att manipulera. Servern ligger skyddad bakom skydd, bearbetning av information går inte att påverka för en användare om den hanteras av servern. Det enda klienten kommer ha tillgång till är att skicka förfrågan till servern i olika former, som att hämta en specifik fil i deras mapp.

Servern har implementerats baserat på com.sun.httpserver inkluderat i javas bibliotek.[1] Den inkluderar funktionen att lyssna på förfrågan kopplat till en port och en URL. Servern skapar en så kallad *hanterare* beroende på vad klienten kan behöva från servern. Denna hanteraren tar emot förfrågan och behandlar den på lämpligt sätt.

Vad som inkluderas i behandlingen av förfrågan är: Felhantering om klienten skickar ogiltig förfrågan som exempelvis filer som inte finns, läsa in *headern* i http paketet, skicka respons till klienten och hantering av data som antingen skickats från klienten eller skall skickas till klienten (POST/GET).

Den data som skickas och tas emot av servern är huvudsakligen filer och mappar, klienten skall kunna förfråga diverse bilder och filer så länge den inte tillhör andra klienter. I det här fallet så vad servern gör är att läsa in datan från filerna och skriva ut dem i ett *http paket* till klienten.

Servern ansvarar även för att skicka ut filerna som är ansvariga för det grafiska gränssnittet hos klienten. *HTML*, *CSS* och *JavaScript* är dessa filer och har varsitt ansvar, dessa är: Den första är *HTML* som ansvarar för innehållet på hemsidan, främst filerna användaren har tillgång till[1]. CSS sköter stylingen på hemsidan[3]. Sist är JavaScript som står för funktionalitet på hemsidan, exempelvis vad som ska hända när användaren klickar på en knapp[4].

Ett problem som uppstår är att servern har behov av skriva om *HTML* filen, varav *HTML* är ett format som java inte stödjer. För att läsa problemet har ett externt bibliotek vid namn *Jsoup* använts[5]. Med det här biblioteket kan servern kolla igenom filer som tillhör användaren, skriva ut det och returnera ut det till användaren.

3.2 Klient

Som beskrivet i tidigare del så har inte klienten tillgång till att göra mer än att skicka förfrågan till servern. Men vad som behövs är implementation att skicka förfrågan till servern. I http protokollet finns det ett flertal olika förfrågningsmetoder[6], endast GET och POST funktionerna har implementerats. Med JavaScript har lyssnare skapats på elementen från html koden, dvs. om användaren klickar på ett element skickas en GET förfrågan till servern från klienten via lyssnare som är fäst på det elementet.

Utöver funktionaliteten på klient sidan krävs även implementation av någon form av styling, det här sköts av *CSS* filen kopplade till de olika *HTML* dokumenten. CSS filens enda ansvar är helt enkelt att se till att hemsidan är användarvänlig grafisk, exempelvis lägga till effekter om användaren håller musen över en länk.

3.3 Autentisering och databas

Eftersom servicen är webbaserad och tillgänglig för mer än en användare har ett auktoriseringssystem implementerats. Anledningen är simpel, varje användare skall bara få tillgång till sina egna filer.

För att uppnå det här har två komponenter blivit applicerade till koden på serversidan. Den första är en metod inkluderat i samma bibliotek använt för att skriva förfrågansmetoderna på servern, den sätter en spärr så att användare måste verifiera sig själv innan åtkomst till sitt filsystem är tillåtet. Skickas en förfrågan att logga in från klientsidan kommer denna funktionen skicka svar på att den vill ha giltigt användarnamn med lösenord.

Nästa del är när servern har fått respons med användarnamn och lösenord, den öppnar upp en anslutning till databasen som lagrar användarnamn, lösenord och sökvägen till användarens filer. Databasen för användarinformationen är skriven i SQL vilket är en relationsbaserad databas struktur[7]. När anslutning har kopplats till databasen börjar den söka igenom sitt *table* för det förfrågade användarnamnet, finns användaren returnerar den till servern informationen kopplad till användaren. Vad servern gör sedan är att jämföra lösenord med det som skickats från klienten, är den giltig skickar servern användaren vidare till sitt filsystem.

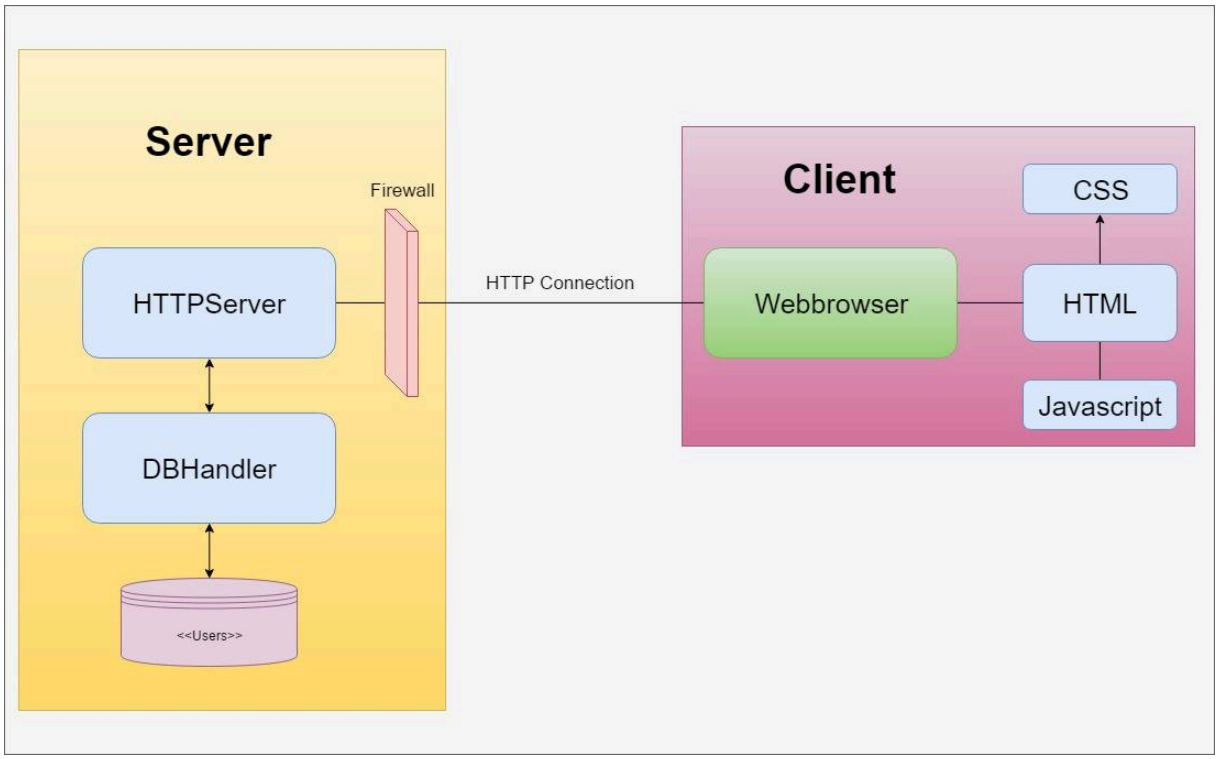
3.4 Raspberry Pi v.3

Sista delen av servicen är hårdvaran servern kommer att köras på, i det här fallet en Raspberry pi version 3. Anledningen är enkelheten och prisvärdigheten kopplade till Raspberryn. För att *hosta* webbservern behövs det som nämnt tidigare en maskin som kan vara igång oavbrutet och helt dedikerad till servern så att den inte kan få avbrott pga. Andra tjänster. Med enkel modifikation har datorn även blivit uppsatt så att den går att fjärrstyra vilket ger tillgänglighet så länge ansvarig för servern har tillgång till internet.
En brandvägg har även blivit implementerad för att blockera anslutningar från oönskade portar. Oönskade i det här fallet är alla förutom porten servern körs på samt port 22 som används för fjärrstyrning.

4 Resultat

4.1 Överblick

Vad som kan ses nedan är en överblick hur hela servicen är uppbyggd. Allt inom Serversidan är det som är implementerat på Raspberry pin. Det inkluderar en brandvägg som skyddar mot oönskad trafik, java servern som hanterar olika förfrågan och databasen som hämtar användare till servern för validering. På klientsidan via webbrowsern får användaren sin information via förfrågan till servern, som visas upp i form av HTML dokumenten. De olika JavaScript filerna hanterar funktionalitet och CSS filen styling.



Figur 4.1: UML diagram över arkitekturen på webbservicen.

4.2 Kod

https://www.facebook.com/

Första exemplet visar kod från servern hur en GET funktion till filer är implementerad. När servern har fått en förfrågan och validerat om filen finns eller inte läser den in headern i output strreamen, sedan datan från filen och skickar klartecken till klienten när allt är skickat. Se bilaga 1.

Nästa exempel är även den från servern, den visar hur servern går igenom filsystemet för att skapa hanterare för filer och mappar tillhörande ägaren. Hittar den en mapp går den i mappen och repeterar rekursivt funktionen till det finns hanterare på varje fil och mapp.
Se bilaga 2.

Sista exemplet från serversidan visar hur databasen hanterar förfrågan från servern vid validering av användare. Den kollar igenom sitt table för användaren, den returnerar sedan ut lösenord och sökväg till servern för jämförelse mellan det inskrivna och det som finns i databasen.
Se bilaga 3.

Nästa tre exempel visar hur klientsidan är uppbyggd, den första visar HTML koden kan se ut när en användare är inne och kollar sina filer. HTML dokumentet är uppbyggt att hämta JavaScript och CSS filerna från servern, och innehållet är uppbyggt av element som hänvisar till filer och mappar tillhörande användaren. Dokumentet innehåller även menyer med alternativ som exempelvis ta bort en mapp, lägga till ny fil i sin sökväg samt lägga till ny mapp.
Se bilaga 4.

Nästa exempel visar hur CSS filen stylar om bakgrunden på hemsidan, den skickar en GET förfrågan till servern på en bakgrundsbild, lägger in den och sätter den så att den är utsträckt och centrerad i bakgrunden på hemsidan.
Se bilaga 5.

Sista exemplet från klientsidan visar hur JavaScript filen sätter lyssnare på varje fil i den mappen användaren är inne i. Den lyssnar specifikt på när användaren *Klickar* på länken, den läser sedan in filnamnet och skickar en GET förfrågan på den filen till servern.
Se bilaga 6.

4.3 Demonstration av hemsidan

Video finns externt som visar hemsidans funktioner.

5 Diskussion

Ett av de större problemen vi stötte på under projektets gång var att få POST att fungera, dvs funktionen som laddar upp filer på servern. När man skickar en fil skickas den genom en stream av bytes. För att kunna skicka en fil över HTTP lägger man till en del information om filen t.ex. filstorlek, filnamn och typ av fil. På grund av detta har vi skrivit metoder som plockar ur enbart payloaden från överföringen för att undvika att icke-relevant data kommer med när vi bygger ihop filen igen. Om detta hade hänt så hade filen blivit korrupt och oanvändbar.

Valet av att använda HTTP som kommunikationsprotokoll har fungerat som vi önskat med POST och GET. Däremot är det inte det enda protokollet som hade fungerat till en sådan här applikation. FTP hade antagligen fungerat lika bra.
HTTP stödjer också andra förfrågningar som t.ex. DELETE, detta protokoll hade vi kunnat använda för att t.ex. ta bort mappar. Problemet med det är att inte alla webbläsare stödjer alla förfrågningar som HTTP erbjuder (där DELETE är en av dem). På grund av detta valde vi att istället för att använda DELETE för att ta bort mappar använde vi en GET förfrågan som ber servern att ta bort önskad fil eller mapp.

Inloggningssystemet fungerar bra men webbläsaren utgör ett litet problem. Webbläsaren sparar nämligen lösenordet i sitt cache-minne. Detta gör att efter man loggat in och stängt ner hemsidan kan man komma åt filerna utan att logga in nästa gång du öppnar hemsidan. En temporär lösning är att manuellt tömma och radera cache-minnet och cookies.

Vi använder oss av Git och Github för att lägga över all server och klient filer till Raspberryn, dvs. .jar filen som kör servern. Så ingen kodning görs på Pin utan allt görs på våra laptops och sedan pushar vi upp det till vårat repository på Github som vi sedan laddar ner på Pin. Väl i Raspberryn startar vi servern med ett enkelt kommando ”java -jar serverStart.jar”.

6 Sammanfattning

Syftet med detta arbete var att från grunden skapa en cloud service lik dropbox/onedrive/google-drive vilket vi har lyckats med. Vi kan hämta och ladda upp filer till och från servern samt att kunna ta bort filer och mappar. Ett enkelt autentiseringssytem är också implementerat vilket gör att användare enbart kan se sina egna filer och ingen annans.

6.1 Framtida forskning

Det vi önskade vi hade hunnit med var att få till en snyggare inloggningssystem. Nuvarande är en pop-up som begär användarnamn och lösenord, detta hade vi viljat gjort snyggare t.ex. att inloggningen skedde på html-sidan istället för en pop-up. En utloggningsfunktion hade också implementerats.

Vi hade även implementerat ett system för att skapa nya konton vilket nuvarande inte finns alls, det måste göras manuellt i databasen.

Referenser

[1] *Package com.sun.net.httpserver*. Oracle.
url:https://docs.oracle.com/javase/8/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/package-summary.html

[2] *HTML*. 2016-12-10. Wikipedia.
url:https://sv.wikipedia.org/wiki/HTML

[3] *Cascading Style Sheets*. 2017-02-07. Wikipedia.
url:https://sv.wikipedia.org/wiki/Cascading_Style_Sheets

[4] *Javascript*. 2017-02-07. Wikipedia.
url:https://sv.wikipedia.org/wiki/Javascript

[5] *jsoup: Java HTML Parser*. Jsoup.
url:https://jsoup.org/

[6] *Hypertext Transfer Protocol*. 2017-03-09. Wikipedia.
url:https://sv.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[7] *Structured Query Language*. 2017-01-03. Wikipedia.
url:https://sv.wikipedia.org/wiki/Structured_Query_Language

[8] Operativsystemet Raspbian
https://www.raspberrypi.org/downloads/raspbian/

[9] Jetbrains IntelliJ IDEA
https://www.jetbrains.com/idea/

[10] Jetbrains Webstorm
https://www.jetbrains.com/webstorm/

Bilagor

Bilaga 1

```
private class GETFileHandler implements HttpHandler {
    private final String name;
    private final String currentFolder;
    public GETFileHandler(String name, String currentFolder) {
        this.name = name;
        this.currentFolder = currentFolder;
    }
    public void handle(HttpExchange exchange) throws IOException {
        System.out.println("Client requested: "+
currentFolder+name);
        Headers header = exchange.getResponseHeaders();
        header.add("Content-Disposition", "attachment;
filename=\""+name+"\"");
        header.add("Content-Type", "application/force-download");
        header.add("Content-Transfer-Encoding", "binary");
        try {
            File file = new File (currentFolder+name);
            if(!file.exists())
                throw new Exception();
            byte [] bytearray  = new byte [(int)file.length()];
            FileInputStream file_Stream = new FileInputStream(file);
            BufferedInputStream buf_Stream = new
BufferedInputStream(file_Stream);
            buf_Stream.read(bytearray, 0, bytearray.length);
            exchange.sendResponseHeaders(200, file.length());
            OutputStream out_stream = exchange.getResponseBody();

            out_stream.write(bytearray,0,bytearray.length);

            file_Stream.close();
            buf_Stream.close();
            out_stream.close();
        } catch (Exception e) {
            String response = "Error 404: File not found.";
            errorHandler(exchange, response, 404);
        }
    }
}
```

Bilaga 2

```
private void iterateFolders(String rootFolder,String folderURL,
String nextURL){
    /*Creates an array of files and folder from current directory*/
    File[] fileArr = new File(rootFolder+"/"+nextURL).listFiles();
    String folder ;
    /*Iterates all files and creates handlers depending of type of
file*/
    for (File file : fileArr) {
        /*If file is a folder*/
        if (file.isDirectory()) {
            folder = nextURL+file.getName()+"/";

            server.createContext(folderURL+folder, new
ContextHandler(rootFolder,folder));
            server.createContext(folderURL+folder + "download", new
GETFolderHandler(file.getName()));
            server.createContext(folderURL+ folder + "upload", new
POSTFileHandler());
            server.createContext(folderURL+folder+"deletefolder",
new DELETEFolderHandler(file.getName()));
            server.createContext(folderURL+folder+"addfolder", new
ADDFolderHandler(file.getName()));
            iterateFolders(rootFolder,folderURL,folder); //Recursive
call to iterate subfolder.
        }
        else {
            /*If file is a file*/
            if(!file.getName().equals(".DS_Store")){ //Type of file
created in MacOS in all directories, not wanted on the webserver

server.createContext(folderURL+nextURL+file.getName(), new
GETFileHandler(file.getName(), rootFolder+"/"+nextURL));

server.createContext(folderURL+nextURL+file.getName()+"/deletefile",
new DELETEFileHandler(file.getName()));
            }
        }
    }
}
```

Bilaga 3

```
public String[] getUserInformation(String username){
    String userName = "", password = "", root = "";
    try{
        /*Creates a connection to the database*/
        Class.forName("org.sqlite.JDBC");
        db_Connection =
DriverManager.getConnection("jdbc:sqlite:users.db");
        db_Connection.setAutoCommit(false);

        /*Query's the database from the table 'userBerrian' to find
information of user.*/
        db_Statement = db_Connection.createStatement();
        ResultSet rs = db_Statement.executeQuery("SELECT * FROM
userBerrian WHERE username ="+"'"+username+"'"+";");
        while(rs.next()){
            userName = rs.getString("username");
            password  = rs.getString("password");
            root = rs.getString("root");
        }
        rs.close();
        db_Statement.close();
        db_Connection.close();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    /*Returns the corresponding password and path to folder from the
user.*/
    String[] strarr = {userName, password, root};
    return strarr;
}
```

Bilaga 4

```
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <link rel="stylesheet" type="text/css" href="/css/>">
  <link href="https://fonts.googleapis.com/css?family=Open+Sans"
rel="stylesheet">
  <link rel="icon" href="/images/icon.png">
 </head>
 <body>
  <div id="padTop"></div>
  <header id="header">
  </header>
  <main>
   <div class="optionBar">
    <li id="downloadFolder"><a>Download Folder</a></li>
    <li id="addFolder">
     <form id="uploadFolder" action="" method="post"
enctype="multipart/form-data">
      <label for="folder">Add Folder: </label>
      <input id="folder" type="text" name="folder"
onchange="javascript:this.form.submit()">
     </form> </li>
    <li id="chooseFile">
     <form id="uploadFile" action="" method="post"
enctype="multipart/form-data">
      <input id="file" class="inputfile" type="file" name="file"
onchange="javascript:this.form.submit()">
      <label for="file">Upload File</label>
     </form> </li>
   </div>
   <h5 id="path"></h5>
   <div id="folders">
    <li class="folder"><a>Lectures</a></li>
    <li class="folder"><a>Old Exams</a></li>
    <li class="folder"><a>Projects</a></li>
    <li class="folder"><a>Workshops</a></li>
   </div>
   <div id="files">
    <li>
     <div class="fileSize">
      137.3KB
     </div><a>syllabus-1DV700-1.pdf</a>
     <div class="deleteFile"></div></li>
   </div>
  </main>
  <footer>
    Made By:
   <br>Mikael Andersson och Carl Johansen
  </footer>
  <script src="/SPAjs"></script>
 </body>
</html>
```

Bilaga 5

```
html{
    background: url("/images/background.jpg")no-repeat center center
fixed;
    -webkit-background-size: cover;
    -moz-background-size: cover;
    -o-background-size: cover;
    background-size: cover;
    width: 100%;
    height: 100%;
    margin: 0px;
    padding: 0px;
}
```

Bilaga 6

```
 function filesListener() {
    files.childNodes.forEach(function(file){            //Iterates
over all files in directory
        var aTag = file.childNodes[2];                  //Retrieves
filename
        if(aTag != null){
            aTag.addEventListener("click",function () {
                window.location.href =url+aTag.innerHTML;  //If user
clicks link, get request for file is sent to server.
            });
        }
    });
}
```