Modern Application Building

With Javascript and HTML

Joseph Bonds

Modern Application Building with HTML and Javascript

Joseph Bonds

This book is for sale at http://leanpub.com/modern-app-building

This version was published on 2017-10-08

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Joseph Bonds by spreading the word about this book on Twitter!

The suggested tweet for this book is:

learn to build apps

The suggested hashtag for this book is #modernappbuilding.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#modernappbuilding

To my exceptionally tolerant friends and relatives

Contents

About This Book

Javascript Is Now a General Purpose Application Building Language

Javascript entered the scene in the realm of building web pages enabling authors to add dynamic effects to web pages. It has grown up to become a language capable of expressing the entire application. Now in order to program an application that will run anywhere (any hardware, any operating system) you need only know one procedural language (javascript), html, and css. No longer is it necessary to know a different language, e. g. PHP, Java, Ruby, or Perl for server side "backend" logic. That is the point of this book.

I will demonstrate in the book how to build an application using just javascript and html that will run as a web page or local application (with no need for an internet connection. And you can run it on a Windows machine, a Mac, or a Linux machine without changes. You can run it on mobile devices, too.

Any application can be written this way, major number crunching, impressive graphics, data base intensive applications.

No doubt, the legacy languages will survive for quite some time for various reasons. Seldom will performance be the actual determinant. Javascript has become fast and is becoming faster as time goes on.

This book deliberately emphasizes the minimalist basics. Using the techniques presented here will enable development of sophisticated applications without the need to go far afield. Yes, you can add various frameworks to simplify your life as a developer, but it is still all just javascript. If you understand this book, you are well equipped to tackle the many frameworks that have evolved.

The code for the samples is available and is downloaded during the "Getting Started" chapter. You may key it in yourself if doing so helps you to learn (recommended), but the downloaded code is there for comparison if you get stuck on some stupid syntax error.

Node JS is used to implement server side code in this book. Most of the logic in the samples is client side, the little server side code could easily be accomplished in PHP or any other server side language.

I use Heroku as my web host. Heroku is developer friendly with free accounts for developers, and friendly scaling to paid production traffic volumes. I encourage you to get an account so that you may show off your accomplishments.

HTML5 is used to define the pages of the samples

The book consists of three examples: a file editor, a chat window, and a graphics editor. The file editor illustrates the relationship of a page's html to the client side javascript and the server side function. The chat window allows multiple clients to chat to illustrate how to implement multiple client applications such as competitive game playing or collaboration. The graphics editor illustrates how to draw grapics and animate them.

About the Author

I began programming in 1970. For many of you, this was way back in the dark ages, before PCs and Macs, and way before Mobile anything. During all of this time I did system programming and application development using a wide variety of programming languages, operating systems, and tools. I was pretty much on the bleeding edge the whole time, eagerly learning new languages and technologies as they happened. I expect this to continue and if anything accelerate. I expect the web to be with us for some time to come, thus I expect the contents of this book to remain relevant for a good while.

I have found programming to be terribly addictive. Google is my best friend. Despite receiving a reasonable retirement income, I just cannot seem to quit programming. I guess it is somewhat like playing video games. There is an objective and a burst of adrenaline when the objective is achieved. Since most projects can be broken into a series of easily achievable objectives, the flow is constant. I am just an adrenaline junky.

I hope you will forgive me if I should infect you with such an addiction. It has been a profitable one for me. I wish you similar success.

– Joseph Bonds

danceswithdolphin@gmail.com

Getting Started

To use this book effectively to learn to develop applications, you will need a few tools: a machine, an editor, a browser, etc. One of the good things about this approach is that you have a lot of latitude.

Questions?

I would encourage you to email any questions you have about this book to danceswithdolphin@gmail.com. Please place the word "book" in the subject.

Machine

You may choose to develop on just about any machine: Windows, Mac, Linux, Raspberry PI. I am using a HP Windows 10 laptop as I write this and develop the sample applications. It all works pretty much the same way on any of the other platforms. I suspect that most of my readers will be on either Windows or OS X. I will endeavor to point out differences within the text where differences exist.

Program Editor

Your choice is somewhat contingent on your choice of platform. I personally use VIM which is available on all the platforms. On Windows I use the GVIM variant. This is available for download for a variety of platforms here[1]. Basically, any text editor will do. I have been using VIM for years in the Linux and Windows environments. The handy features an editor should have are syntax coloring, matching parenthesis and brace highlighting, search and replace, and differencing.

GIT

GIT is a version control system that has become insanely popular. You will use it in this book to download the samples for this book and to push your projects to the heroku server if you wish to share your application on-line so that you can show off your handiwork to others. You can download it here[2].

Signup for a free account at github.com[3], you will need the credentials to download the sample code.

[1] http://www.vim.org
[2] http://git-scm.com
[3] https://github.com

Chrome Browser

Your web browser includes the javascript interpreter which will execute all of your client side logic. I use Chrome to have access to an up to date version of HTML5 and javascript (which is still a rapidly evolving language). The Chrome javascript engine is also at the heart of Node.js (see below) which is the engine that drives server side logic and provides access to your file system. Thus by using Chrome, you have the same engine driving both the client and server side logic. In theory, you can use any recent version of another browser. But I prefer Chrome because it has built in developer tools which are very useful in debugging your code. You can download chrome here[4].

Node.js

As mentioned above, your server side logic will be executed by the javascript engine within node. Node consists of three parts: node, npm, and the Node.js Command prompt. Node is the javascript engine. Npm is the node package manager. Node.js command prompt is the command window within which you issue node, npm, and native commands. As of this writing, the most recent version of node (v6.11.1) does not include a separate command component. For windows, you simply use the "cmd" shell. Node may be downloaded at https://nodejs.org[5].

Express

Express is a npm package which provides web server functionality to node. I will detail the express installation in the sample program. It is installed within the Node.js Command prompt with the npm install command.

Heroku

Heroku is a web hosting provider which provides free web hosting to newbies. If your application becomes insanely popular, you may become a paid subscriber and scale up the number of servers running your code to meet your demand. Heroku is here[6].

Download The Sample Applications

Open the Node.js Command Window. On Windows 10, type "cmd" in the "Ask me anything" box. Select "Node.js command prompt" or "Command Prompt" (v6.11.1).

[4] https://www.google.com/chrome/browser/desktop/index.html
[5] https://nodejs.org
[6] https://heroku.com

Now create or select a directory on your machine where you want the sample

username). “mysample” is the name of the subdirectory to be created for the cloned project. You will be prompted for your Github user name and password.

- Then, at the “Node.js command prompt”, type the following:

```
  cd Documents
  git clone https://github.com/danceswithdolphin/Modern-App-Building-Examples.gi\
t mysample
  cd mysample
  node server.js
```

- Your output should look something like this:

```
  C:\Users\joe>cd Documents
  C:\Users\joe\Documents>git clone  https://github.com/danceswithdolphin/Modern-\
App-Building-Examples.git mysample
  Cloning into 'mysample'...
  remote: Counting objects: 452, done.
  remote: Compressing objects: 100% (323/323), done.
  Receiving objects:  82% (371/452)    419 (delta 75)     eceiving objects:  81%\
 (367/452)
  Receiving objects: 100% (452/452), 346.32 KiB | 0 bytes/s,    done.
  Resolving deltas: 100% (86/86), done.
  Checking connectivity... done.
  C:\Users\joe\Documents>cd mysample
  C:\Users\joe\Documents\mysample>node server.js
  Node app is running on port 5000
```

When you see “\” at the end of a line of code, this line and the following line should be joined and interpreted as one long line, leaving out the “\”.

Open your browser and type “localhost:5000” to view the application window. You are now running the application off-line in your browser. At this point you can play with the sample application to your hearts content. You may skip forward to the testing sections in the following chapters and run the tests here to develop an understanding of the applications. Explore the source code to see how much of it makes sense intuitively.

In the following chapters we will build the apps in another directory step by step.

At the end of the book you will understand how to build applications with these tools.

File Editor Sample

General Application (App) Requirements

An application (app) should be capable of the following:

1. Presenting a user interface (UI) to the app user which can display information to the user, accept information from the user, and provide app navigation to the user (button clicks, etc).
2. Retrieve information (from a file system, a database, etc.).
3. Store information (in a file system, a database, etc.).
4. Maintain state information ("remember") the user's prior responses).

This example illustrates all of this and may be run offline (with no internet connection, on laptops and desktops which use a variety of operating systems and hardware) or online from a webserver as a browser app on a mobile device, laptop, or desktop.

The Server Component

In any case, this app has a server component. This server component runs on the laptop or desktop for offline use or run on the server for online use. The server component provide access to file systems and databases and possibly other resources of the host.

Setting Up the Server Component

We will first setup the server component of this application, later we will implement the user interface for the application.

First create a directory for your project.

- Open the Command Prompt Window. Create a new directory to contain the project. I used the following commands to create the directory "mysample" in my home directory and a "public" subdirectory for static assets and a subdirectory "data" of "public" for data files:

```
md mysample
cd mysample
md public
cd public
md data
cd ..
```

- Then initialize your project with the “npm init” command:

```
npm init
```

- You may simply hit enter to answer all the questions presented. This updates the package.json file in your directory. Now, to install express, enter the following command:

```
npm install express --save
```

- The “–save” option adds an entry for express in the package dependences of package.json. Your directory is now prepared to contain the elements of your applications, which we will add to this directory and subdirectories.
- Your Command Prompt window should look something like this at this point:

```
C:\Users\joe>md mysample
C:\Users\joe>cd mysample
C:\Users\joe\mysample>md public
C:\Users\joe\mysample>cd public
C:\Users\joe\mysample\public>md data
C:\Users\joe\mysample\public>cd ..
C:\Users\joe\mysample>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.
See `npm help json` for definitive documentation on these fields
and exactly what they do.
Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.
Press ^C at any time to quit.
name: (mysample)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
```

```
About to write to C:\Users\joe\mysample\package.json:
{
  "name": "mysample",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
    },
  "author": "",
  "license": "ISC"
}
Is this ok? (yes)
C:\Users\joe\mysample>npm install express --save
mysample@1.0.0 C:\Users\joe\mysample
`-- express@4.14.1
  +-- accepts@1.3.3
  | +-- mime-types@2.1.14
  | | `-- mime-db@1.26.0
  | `-- negotiator@0.6.1
  +-- array-flatten@1.1.1
  +-- content-disposition@0.5.2
  +-- content-type@1.0.2
  +-- cookie@0.3.1
  +-- cookie-signature@1.0.6
  +-- debug@2.2.0
  | `-- ms@0.7.1
      +-- depd@1.1.0
  +-- encodeurl@1.0.1
  +-- escape-html@1.0.3
  +-- etag@1.7.0
  +-- finalhandler@0.5.1
  | +-- statuses@1.3.1
  | `-- unpipe@1.0.0
  +-- fresh@0.3.0
  +-- merge-descriptors@1.0.1
  +-- methods@1.1.2
  +-- on-finished@2.3.0
  | `-- ee-first@1.1.1
  +-- parseurl@1.3.1
  +-- path-to-regexp@0.1.7
  +-- proxy-addr@1.1.3
```

```
| +-- forwarded@0.1.0
| `-- ipaddr.js@1.2.0
+-- qs@6.2.0
+-- range-parser@1.2.0
+-- send@0.14.2
| +-- destroy@1.0.4
| +-- http-errors@1.5.1
| | +-- inherits@2.0.3
| | `-- setprototypeof@1.0.2
| +-- mime@1.3.4
| `-- ms@0.7.2
+-- serve-static@1.11.2
+-- type-is@1.6.14
| `-- media-typer@0.3.0
+-- utils-merge@1.0.0
    `-- vary@1.1.0
npm WARN mysample@1.0.0 No description
npm WARN mysample@1.0.0 No repository field.
C:\Users\joe\mysample>
```

The Server Code

Using your text editor, key the following code into a file “server.js” in your application directory (“mysample” above).

```
// server.js
// load the things we need
var express = require('express');
var app = express();
var fs = require("fs");
var path = require('path');


var fileName = "";
var dirName = "";
var mydata = "";
var subtasks = 0;
var subdirs = [];
var files = [];
function f_stats_complete (res, subdirs, files) {
  res.send('?subdirs='+subdirs+'&files='+files);
```

```
}
function createCallback (_fullpath, _res) {
  return function (err, stats) {
    if (err) {
      console.log('fs.stat err='+err);
    } else if (stats.isDirectory()) {
      subdirs.push(_fullpath);
    } else if (stats.isFile()) {
      files.push(_fullpath);
    }
    if (--subtasks === 0) {
      f_stats_complete (_res, subdirs, files);
    }
  }
}

app.set('port', (process.env.PORT || 5000));

// set the view engine to ejs
app.set('view engine', 'ejs');
app.use(express.static(__dirname + '/public'));

app.get('/xhr-stat',function(req,res){
  fileName=req.query.filename;
  fs.stat(fileName,(err,stats) => {
  res.send('?err='+err+'&stats='+JSON.stringify(stats));
  });
});

app.get('/xhr-unlink-file', function(req, res){
  fileName=req.query.filename;
  fs.unlink(fileName, (err) => {
    if (err) throw err;
    res.send('deleted '+ fileName + ' successfully');
  });
});

app.get('/xhr-write', function(req, res){
  fileName=req.query.filename;
  mydata=req.query.mydata;
  fs.open(fileName, 'w', (err,fd) => {
    if (err) {
```

```
      console.log('err.code='+err.code);
      console.log('err='+err);
      if (err.code === 'ENOENT'){
        console.log('error='+err);
        res.send('error=ENOENT')
      } else {
        throw err;
      }
    }
    else {
      mydata = mydata.replace(/[\r]/gm,'');
      fs.writeFile(fd, mydata, (err, data) => {
        if (err) throw err;
        fs.close(fd, (err) => {
          if (err) throw err;
        });
        res.send('written successfully');
      });
    }
  });
});

app.get('/xhr-read', function(req, res){
  fileName=req.query.filename;
  fs.open(fileName, 'r', (err,fd) => {
    if (err) {
      if (err.code === 'ENOENT'){
        res.send('error=ENOENT')
      } else {
        res.send('error='+err);
      }
    } else {
      fs.readFile(fd, (err, data) => {
      if (err) throw err;
      mydata = data.toString();
      fs.close(fd, (err) => {
      if (err) throw err;
      });
      res.send(mydata);
      });
    };
  });
```

```
101 });
102
103 app.get('/xhr-mkdir', function(req, res){
104   dirName=req.query.dirname;
105   fs.mkdir(dirName, (err) => {
106     if (err) {
107       res.send('?error='+err);
108     } else {
109       res.send();
110     }
111   });
112 });
113
114 app.get('/xhr-rmdir', function(req, res){
115   dirName=req.query.dirname;
116   fs.rmdir(dirName, (err) => {
117     if (err) {
118       res.send('?error='+err);
119     } else {
120       res.send('OK');
121     }
122   });
123 });
124
125 app.get ('/xhr-readdir', function(req,res){
126   if (req.query.dirname) {
127     dirName=req.query.dirname;
128     subdirs=[];
129     files=[];
130     fs.readdir(dirName, (err,data) => {
131       if (err) {
132         if (err.code === 'ENOENT'){
133           res.send('error=ENOENT')
134         } else {
135           console.log('error='+err)
136           res.send('error='+err);
137           throw err;
138         }
139       } else {
140         var arrLength=data.length;
141         subtasks = arrLength;
142         if (subtasks === 0){
```

```
143           res.send('error=no entries in directory');
144         }
145         for (var i =0; i < arrLength; i++) {
146           var fullpath=dirName+path.sep+data[i];
147           var stats = null;
148           fs.stat(fullpath, createCallback (fullpath, res));
149         }
150       }
151     });
152   } else {
153     res.send('error=dirname missing from xhr-readdir parameters');
154   }
155 });
156
157 app.listen(app.get('port'), function() {
158   console.log('Node app is running on port', app.get('port'));
159 });
```

Server Code Commentary

Lines 3-6 load the required modules into the javascript engine. The node require function loads a module to make the functionality of the module available to the code which follows. More information on node modules may be found here[7]. Require may be used to load code from various sources, packages installed with NPM, core modules provided with node, modules you have written yourself, etc.

Express provides a lot of functionality: it establishes a web server which can listen for http requests on a port, it serves static assets such as your html and images, and it provides a routing mechanism to code in your file.

Line 4 creates an object "app" for your webserver. This allows the following code to refer to the webserver.

Line 5, creates your node file system object, which permits your code to access your machine's file system. The node file system api is documented here[8].

Line 6, creates a path object. We use this only to discover the proper path separator to use between directory names for the underlying operating system. I. E. '\' for Windows, '/' for everyone else.

Lines 9-14 declare global variables which will be used in the following code.

Line 15-17 define a function "f_stats_complete" which will be called asnchronously when all the entries in a directory have been classified as files or subdirectories. The first parameter is the response

[7] https://nodejs.org/docs/v0.4.1/api/modules.html
[8] https://nodejs.org/api/fs.html

object captured at the time createCallback was called. The second and third parameters, subdirs and files, are arrays of subdirectories and files that are used by the client side logic to build the user interface.

Lines 18-31 is a factory which generates unnamed functions which are used as asynchronous callbacks for file system stat calls which are made for all the entries in a directory. This factory method of creating callbacks utilizes the concept of closures to preserve the values of the parameters of createCallBack until the events occur.

Line 33 sets the port that the express webserver should listen to for incoming http requests. In a server environment such as Heroku, this is taken from an environment variable provided by the service provider, else in a development environment, such as your laptop, a constant of 5000 is assigned.

Line 36 sets the view engine to "ejs", embedded javscript. This is not utilized in this example, but is useful if your application uses templating. The other choice is "jade".

Line 37 sets the subdirectory from which static assets, such as html and images, javascript, and images are served. References to assets are relative to this subdirectory. Thus html files would be served from C:\Users\joe\mysample\public.

Lines 39-44 illustrate the "routing" feature of express which associates http requests with javascript code. A http get request for "/xhr-stat" will execute the unnamed function defined here. The parameters provided to the unnamed function, "req" and "res" are objects used by express to encapsulate the http request and the http response, respectively.

Line 40 the file name to be "stat"ed is retrieved from the request.

Line 41 invokes the file system stat function passing the file name and a function to be executed asynchronously on completion. The completion event passes an error object and a stats object to the completion function. The "() ⇒ {}" construct is an es6 arrow function definition. The arrow function definition is a new shorthand notation for anonymous function definition. Though arrow functions and anonymous functions are similar, there are some differences in scoping and interpretation. I suggest googling "javascript arrow function" to develop an understanding of this esoterica.

Line 42 sends the html response back to the requestor. Note that the string sent back looks like a html query, i.e. url?parm1=val1&parm2=val2. A good explanation of JSON embedded in a good es5 javascript book is here[9].

Lines 46-52 is the handler for a request to delete (unlink) a file. This and the following handlers are so similar to the one above that I do not feel compelled to labor the detailed explanation.

Lines 54-79 is the handler for a write request. Something new here is the "throw" statement which displays the unexpected error on the console in the Command Prompt window.

Lines 81-101 is the read file handler. The ENOENT error code indicates that the file does not exist during the open. Open errors are sent back to the client as an "error=" clause in the query string since these errors are likely to actually be encountered and the client should be able to respond sensibly. The entire content of the file is read into a buffer and passed as the data parameter to the

[9] http://speakingjs.com/es5/ch22.html

asynchronous callback of the read. This buffer is converted to a string which is sent back to the client after the file is closed.

Lines 103-112 is the handler for creating a new directory. If an error is encountered it is sent to the client.

Lines 114-123 is the handler for removing a directory. If an error is encountered, it is sent to the client.

Lines 125-155 handles a request to read a directory and builds arrays of the files and subdirectories contained within it. This it accomplishes by launching fs.stat calls for each entries with callbacks created by createCallback above.

Lines 157-159 starts the server. Line 158 logs the message announcing the port it is running on.

Server Summary

The bulk of the server code are response functions that are “routed” from http urls by express. For example “http://localhost:5000/xhr-read” is routed to the function defined beginning at line 81. Each of the response functions accepts a request parameter “req”, and a “response” parameter “res”. Each response function retrieves information from the request parameter which elaborates the details of the desired action. The response function processes by executing the desired action by invoking “fs” file system calls. Finally the response function sends the response with “res.send”. The “xhr-“ prefix is intended to signal that the client will be utilizing XMLHttpRequest to invoke it. The server we just built will permit us to build an application that is capable of browsing directories, reading and writing text files, creating directories, and deleting directories and files.. It illustrates the nature of asynchronous processing which is so central to the Node philosophy. This server can be be reused as the heart of a large class of web applications that require reading and writing host files and serving html resources.

Testing the Server

Serving an HTML Page

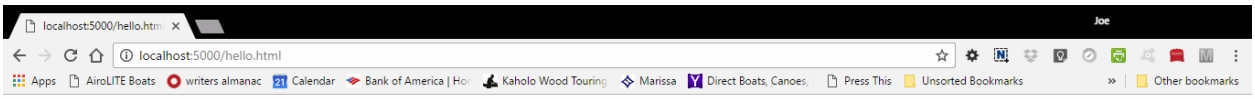
Place a simple html page in the public subdirectory of your project. For example, “hello.html”.

```
1  <h1>Hello, World</h1>
```

- Navigate to your project in the Command Prompt window and start the server:

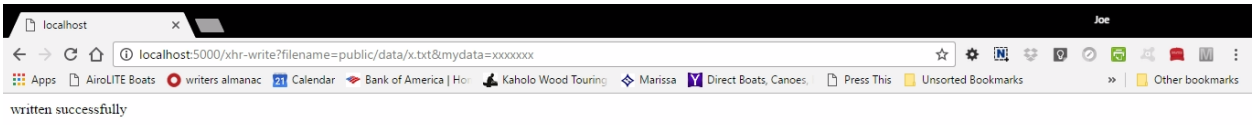
```
C:\Users\joe>cd mysample

C:\Users\joe\mysample>node server.js
Node app is running on port 5000
```

Now open a Chrome browser window and key “localhost:5000/hello.html” in the Chrome Address bar and hit the enter key. You should see:

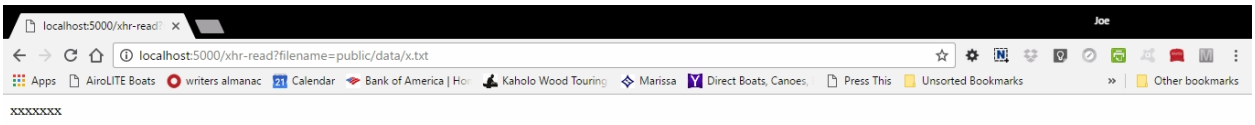


Hello

Now key “localhost:5000/xhr-write?filename=public/data/x.txt&mydata=xxxxxxx” in the Chrome Address bar and hit the enter key. You should see:



Now key “localhost:5000/xhr-read?filename=public/data/x.txt” in the Chrome Address bar and hit the enter key. You should see:



Note that filename is relative to the project directory. Thus the file just written and read is C:\Users\joe\mysample\public\data\x.txt.

The other xhr functions can be similarly tested.

Here we will proceed to build the sample applications user interface as a client web page which will exercise the the remaining functions.

The Client Code

With your text editor, key the following file “fileeditor.html” and save it in your public directory

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=2">
<title>File Editor</title>
<style>
body {font-size:large;}
button {font-size:large;}
.button {font-size:large;}
input {font-size:large;}
textarea {font-size:large;}
</style>
<script>
window.onerror = function (errorMsg, url, lineNumber, column, errorObj) {
    alert('Error: ' + errorMsg + ' Script: ' + url + ' Line: ' + lineNumber
    + ' Column: ' + column + ' StackTrace: ' +  errorObj);
}

var responseText = [];
var errors = [];
var subdirs = [];
var subdirsi = '';
var files = [];
var innerhtml='';
var filesep='\\';
var the_text = '';

  function handleFileSelect(evt) {
    var upfiles = evt.target.files; // FileList object
    var f = upfiles[0];
    var reader = new FileReader();
    reader.onload = (function(theFile) {
      return function(e) {
```

```
        document.getElementById('textarea').value = e.target.result;
        document.getElementById('file_name').value = document.getElementById('dir_name'\
).value + '/' + escape(f.name);
        f_save();
        // f_write();
      };
    })(f);
    reader.readAsText(f);
  }


function f_download() {
  var myfile=document.getElementById("file_name").value;
  myfile = myfile.replace('public/', '');
  innerhtml = '<a href="' + myfile + '" id="download" download>Download</a>';
  document.getElementById("download_div").innerHTML = innerhtml;
  document.getElementById("download").click();
}

function f_up() {
  var mydir = document.getElementById("dir_name").value;
  var i = mydir.lastIndexOf('/');
  if (i > 0) {
    mydir = mydir.substring(0,i);
  } else {
    mydir = '/';
  }
  document.getElementById("dir_name").value = mydir;
  f_save();
  f_load();
}

function f_write(){
  var myfile=document.getElementById("file_name").value;
  var mydata=document.getElementById("textarea").value;
  var params="filename="+myfile+"&mydata="+encodeURIComponent(mydata);
  var url="xhr-write?"+params;
  var http=new XMLHttpRequest();
  http.open("GET", url, true);
  http.onreadystatechange = function()
  {
    if(http.readyState == 4 && http.status == 200)
```

```
    {
      responseText = http.responseText;
      responseText = responseText.replace(/\\/g,'/');
      var urlParams = new URLSearchParams (responseText);
      if (urlParams.has('error')) {
        document.getElementById("response").innerHTML = responseText;
      } else {
        f_load();
      }
    }
  }
  http.send(null);
}

function f_makedir(){
  var mydir=document.getElementById("dir_name").value;
  var params="dirname="+mydir;
  var url="xhr-mkdir?"+params;
  var http=new XMLHttpRequest();
  http.open("GET", url, true);
  http.onreadystatechange = function()
  {
    if(http.readyState == 4 && http.status == 200)
    {
      responseText = http.responseText;
      var urlParams = new URLSearchParams (responseText);
      if (urlParams.has('error')) {
        document.getElementById("response").innerHTML =urlParams.get('error');
      }
      else
      {
        f_load();
      }
    }
  }
  http.send(null);
}

function f_select_file (selfile) {
  document.getElementById("response").innerHTML = '';
  document.getElementById("file_name").value = selfile;
  window.localStorage["filename"] = selfile;
```

```
119   var url = "xhr-read";
120   var params = "filename=" + selfile;
121   var http=new XMLHttpRequest();
122   http.open("GET", url+"?"+params, true);
123   http.onreadystatechange = function()
124   {
125     if(http.readyState == 4 && http.status == 200)
126     {
127       document.getElementById("textarea").value = http.responseText;
128     }
129   }
130   http.send(null);
131 }
132 
133 function f_delete_dir () {
134   document.getElementById("response").innerHTML = '';
135   var seldir = document.getElementById("dir_name").value;
136   var url = "xhr-rmdir";
137   var params = "dirname=" + seldir;
138   var http=new XMLHttpRequest();
139   http.open("GET", url+"?"+params, true);
140   http.onreadystatechange = function()
141   {
142     if(http.readyState == 4 && http.status == 200)
143     {
144       f_load();
145     }
146   }
147   http.send(null);
148 }
149 
150 function f_delete_file () {
151   document.getElementById("response").innerHTML = '';
152   var selfile = document.getElementById("file_name").value;
153   var url = "xhr-unlink-file";
154   var params = "filename=" + selfile;
155   var http=new XMLHttpRequest();
156   http.open("GET", url+"?"+params, true);
157   http.onreadystatechange = function()
158   {
159     if(http.readyState == 4 && http.status == 200)
160     {
```

```
161         document.getElementById("response").innerHTML = "deleted/loading";
162         f_load();
163       }
164     }
165     http.send(null);
166   }
167
168   function f_select_subdir (subdir) {
169     window.localStorage['dirname'] = subdir;
170     document.getElementById("dirs_div").innerHTML = 'There are no subdirectories';
171     document.getElementById("files_div").innerHTML = 'There are no files in this d\
172   irectory';
173     f_load();
174   }
175
176   function f_readdir () {
177     var dirname=window.localStorage['dirname'];
178     document.getElementById("dirs_div").innerHTML = 'Working, Please Wait....';
179     document.getElementById("files_div").innerHTML = '';
180     var url = "xhr-readdir";
181     var params = "dirname=" + dirname;
182     var http=new XMLHttpRequest();
183     http.open("GET", url+"?"+params, true);
184     http.onreadystatechange = function() {
185
186       if (http.readyState == 4 && http.status == 200)
187       {
188         responseText = http.responseText;
189         responseText = responseText.replace(/\\/g,'/');
190         var urlParams = new URLSearchParams (responseText);
191         if (urlParams.has('error')) {
192           document.getElementById("dirs_div").innerHTML = responseText;
193         } else {
194           innerhtml = 'Subdirectories: ';
195           if (urlParams.has('subdirs')){
196             subdirs = urlParams.get('subdirs');
197             subdirs=subdirs.split(",");
198             var n = subdirs.length;
199             if (n > 0 && subdirs[0] > '') {
200               for (var i = 0; i< n; i++) {
201                 subdirsi = subdirs[i];
202                 innerhtml += '<button type="button" onclick="f_select_subdir(\''+subdirsi\
```

```
+'\');return false">'+subdirs[i]+'</button>';
            }
          } else {
            innerhtml += "There are no subdirectories";
          }
        } else {
          innerhtml += "There are no subdirectories";
        }
        document.getElementById("dirs_div").innerHTML = innerhtml;


        if (urlParams.has('files')){
          innerhtml='Files: ';
          files = urlParams.get('files');
          files = files.split(",");
          n = files.length;
          if (n > 0 && files[0] > '') {
            for (var i= 0; i < n; i++) {
              filesi = files[i];
              innerhtml += '<button type="button" onclick="f_select_file(\''+fil\
esi+'\');return false">' + files[i] + '</button>';
            }
          } else {
            innerhtml += 'There are no files in this directory';
          }
        } else {
          innerhtml += 'There are no files in this directory';
        }
        document.getElementById("files_div").innerHTML = innerhtml;


      }
    } else if (http.readyState == 4 && http.status == 0)
    {
        document.getElementById("dirs_div").innerHTML = 'Subdirectories: There a\
re no subdirectories';
        document.getElementById("files_div").innerHTML = 'Files: There are no files in \
this directory';
    }
  }
  http.send(null);
}
```

```
245
246 function f_load () {
247   document.getElementById("response").innerHTML = '';
248   if (window.localStorage["dirname"] !== undefined) {
249     document.getElementById("dir_name").value =window.localStorage["dirname"];
250   } else {
251     window.localStorage["dirname"] = document.getElementById("dir_name").value;
252   }
253   if (window.localStorage["filename"] !== undefined) {
254     document.getElementById("file_name").value =window.localStorage["filename"];
255   } else {
256     window.localStorage["filename"] = document.getElementById("file_name").value\
257 ;
258   }
259   f_readdir();
260 document.getElementById('upfilesid').addEventListener('change', handleFileSelect\
261 , false);
262 }
263
264 function f_save () {
265   window.localStorage["dirname"]=document.getElementById("dir_name").value;
266   window.localStorage["filename"]=document.getElementById("file_name").value;
267 }
268 </script>
269 </head>
270 <body onload="f_load();">
271 <h1>File Editor</h1>
272 Directory: <input type="text" size="80" id="dir_name" value="public/data" class=\
273 "button">
274 <button type="button" onclick="f_save();f_load();" class="button">Load</button>
275 <button type="button" onclick="f_up();" class="button">Up</button>
276 <button type="button" onclick="f_makedir();" class="button">Create Subdirectory<\
277 /button>
278 <button type="button" onclick="f_delete_dir();" class="button">Delete directory<\
279 /button><br>
280 <br>
281 <div id="dirs_div">
282 </div>
283 <div id="files_div">
284 </div>
285 <div id="text_area">
286         <textarea id="textarea" rows="18" cols="97" class="button"></textarea>
```

```
</div>
File: <input type="text" class="button" size="80" id="file_name" value="">
<button type="button" onclick="f_save();f_write();" class="button">Write</button>
<button type="button" onclick="f_delete_file();" class="button">Delete</button><\
br>
<br>
<button type="button" onclick="f_download();" class="button">Download</button>
<div id="response"></div>
<div style="visibility:hidden;" id="download_div"></div>
Upload: <input type="file" class="button" id="upfilesid" name="upfiles[]" />
</body>
</html>
```

Local Storage

This example used html5 local storage to maintain state; that is to remember the user's input from screen to screen and session to session. Thus we dodge all the issues surrounding "cookies", session identifiers, etc. Local Storage is an object automatically maintained by the browser as a window object (in a mysterious location on your client's file system). It is accessed as a normal javascript object. Thus you may set or get the value associated with "mykey" as window.localStorage["mykey"]. The implementation of local storage varies a bit from browser to browser, but I think you can count on being able to store up to 5 megabytes of data this way on all modern browsers. More information on this feature is available here[10].

XMLHttpRequest

This example uses XMLHttpRequest to transfer data between client and server. This allows us to invoke the functions in our server above to read and write files to the servers file system. This is an asynchronous protocol defined here[11]. In general this works as follows: a request is constructed by calling the constructor XMLHttpRequest(), the open method is used to initialize the request with the url and parameters, an anonymous unnamed function is assigned to the request's onreadystatechange property to handle the completion event, the request's send method transmits the request, the comletion event waits for the completion and processes the response. XMLHttpRequest.readyState == 4 means the request is done. http.status == 200 means OK successful.

[10] https://www.w3schools.com/html/html5_webstorage.asp

[11] https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

File upload

Access to a local file for upload is achieved by html5 facility for selecting a file to upload and reading the file. This is described here[12]. Reading client files this way is deemed secure because the user has chosen the file.

Client Code Commentary

Line 1 Declares this file as a html5 document.

Line 2 The opening html tag.

Line 3 The opening head tag. The javascript script will be in the head section.

Line 4 The meta tag declares the character encoding of the file to be UTF-8.

Line 5 This meta tag defines the viewport width and initial scale.

Line 6 Sets the window title to “File Editor”.

Lines 7-13 CSS style rules setting font-size to “large”. This is particularly important for small screen mobile devices.

Line 14 The beginning script tag. Javascript code follows.

Lines 15-18 Assign an unnamed function to the window’s error event. This will cause an alert popup if an error occurs on the page. Useful debugging information is included in the popup.

Lines 20-27 The global variable declarations.

Lines 29-43 Defines an event listener as a named function handleFileSelect which is invoked when a file is selected for uploading from the client’s machine. evt.target.files is an array of the client’s files selected. A FileReader object is created which will populate the text area with the text read from the selected file. The FileReaders load event occurs asynchronously. The contents of the text area may then be written to the file system by updating the file-name text element and clicking the write button. The FileReader is started on line 42. Also during the load event, the directory name and file name from the Directory and File text fields are saved in local storage.

Lines 46-52 This code provides the ability to download the contents of a file which has been read from the server and is being displayed in the text area. The filename in the file_name text elements is the path to the file relative to the server’s application directory. HTML file names are relative to the directory from which the page is displayed, i.e. “public”. Hence the “public” part of the path is removed in line 48 before constructing an “<a” anchor element in lines 49-50. In line 51, the anchor element is “click”ed and the file downloads.

[12] https://developer.mozilla.org/en-US/docs/Using_files_from_web_applications

Lines 54-65 The up button provides access to the parent of the directory currently being displayed by stripping the subdirectory at the end of the path displayed in the dir_name text element, saving it in local storage, and reloading by calling f_load.

Lines 67-89 Implements the Write button which writes the contents of the textarea to the file specified in the file_name text element. Line 70 constructs the parameters for the XMLHttpRequest. Note the use encodeURIComponent to encode the data from the text area to insure proper transmission of the data. Line 71 constructs the XMLHttpRequest. Line 72 initializes the request. Line 74 defines the completion function. Line 76 waits for successful completion of the request. Line 84 invokes f_load to return the screen to a waiting for action state. Line 88 fires the request.

Lines 91-113 Implements the Create Subdirectory button. The parameters and url for the XMLHttpRequest are constructed in Lines 93 and 94. The XMLHttpRequest is constructed in line 95. The request is initialized in line 96. The completion event is set in lines 97-111. The response from the request is processed beginning in line 93. The response is processed as a URLSearchParams object beginning in line 102. If the response has an error component, it is displayed in the response element in the html document in line 104, else f_load prepares for the next user activity. Line 112 sends the request to the server.

Lines 115-131 If the user clicks on one of the buttons created for the files displayed, the response display is cleared in line 116, the file_name text field is set to the selected file and stored in localStorage in line 117 and 118. The XMLHhttpRequest is setup to read the selected file in lines 119-121. The completion function in lines 123-129 displays the contents of the file in the textarea. The XMLHttpRequest is sent in line 130.

Lines 133-148 Processes the Delete directory button.

Line 150-166 Processes the Delete button.

Lines 168-174 Processes a selected subdirectory button by initializing the dirs_div and files_div with "there are no" messages, then invokes f_load to add buttons to these divs with the subdirectories and files found for the selected directory.

Lines 176-244 The f_readdir funtion is invoked from the f_load function to display the entries in the directory recorded in localStorage as buttons within the dirs_div and files_div divisions of the page. It retrieves the directory name from localStorage in line 177. Then it sets up the XMLHttpRequest xhr-readdir in lines 180-182. The request completion routine starting at 184 retrieves the response text which is formatted as a URLSearch string, Line 189 changes backslashes to slashes (for Windows). Line 190 constructs an URLSearchParams object from the response text. If there is an error parameter, the error is displayed in the dirs_div division else the subdirs parameter is split at the commas into an array at line 197. Within the for loop, a button is constructed with an onclick routine specified as f_select_subdir with the parameter set to the subdirectory name at line 202. If there are no subdirectorys, the innerhtml is set to "There are no subdirectories" at lines 206 and 209. At line 211 the innerhtml of dirs_div is set to the value accumulated above.

Starting at line 214, the files returned from the XMLHttpRequest are processed in a manner similar to the subdirectories above. The innerhtml of the files_div is set at line 231.

At 235 a test for an error status (0), is made. If there is an error, “there are no” messages are displayed.

The XMLHttpRequest is sent at line 243.

Line 246-262 The f_load routine is invoked when the page is loaded and when a restart is desired without reloading the page. The default dir_name is set from localStorage. If the dirname has a value, the localStorage is set to that value. Similarly the default filename comes from localStorage or localStorage is set from the screen.

Line 259, the population of the screen is started with f_readdir here.

Line 260 The upload file event listener is set here for the change event.

Line 264 The f_save routine saves the dir_name and file_name from the screen to localStorage.

Line 268 end of script, the html starts now.

Line 269 end of head tag.

Line 270 body tag with onload event specified as f_load.

Line 271 h1 heading tag for the screen.

Line 272 Text field dir_name default value “public/data”.

Line 274 “Load” button. Invoke f_save and f_load on click.

Line 275 “Up” button. Invoke f_up to go to parent directory.

Line 274 “Create Subdirectory” button. Invoke f_makedir.

Line 278 “Delete directory” button. Invoke f_delete_dir.

line 280 break tag. end of line.

Line 281 dirs_div div tag. The division where subdirectory buttons are displayed.

Line 283 files_div div tag. The division where file buttons are displayed for files within the directory.

Line 285 text_area div tag.

Line 286 textarea tag. The text area in which the selected file contents is displayed and edited. Line 288 The File text area displays the name of the file in the text area. It may be edited to write to a new file.

Line 289 The Write button to write the text in the textarea to the file specified in the File text area. Invokes f_save and f_write.

line 290 The Delete button. Deletes the file specified in the File Text area. Invokes f_delete_file.

Line 293 The Download button. Invokes f_download.

Line 295 The hidden download division. The download button works by creating a hidden download anchor button and programmatically clicking it.

Line 296 the Upload element. This permits the browsing for a file to upload to the host.

Line 294 Response div. Displays error responses.

Line 297 end of body tag.

line 298 end of html tag.

Client Code Summary

The client code above illustrates how to use XMLHttpRequest to invoke our "xhr-" response routines of our server, how to define and access html text and textarea fields, and how to upload and download text files.

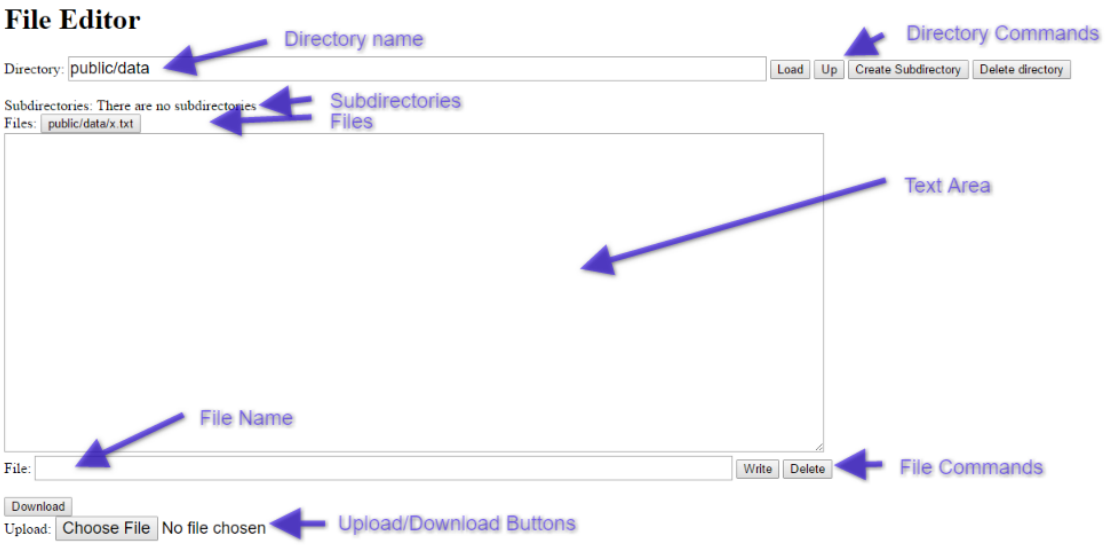
Testing the Application

- Navigate to your project in the Command Prompt window and start the server:

```
C:\Users\joe>cd mysample

C:\Users\joe\mysample>node server.js
Node app is running on port 5000
```

Now open a Chrome browser window and key "http://localhost:5000/fileeditor.html" in the Chrome Address bar and hit the enter key. You should see:

Test 1 File Editor Screen



The Directory Name entry box.

This box governs much of the operations of this application. The value in this box is “remembered” in local storage from invocation to invocation. Thus on invocation, the screen will look very similar to the last session for this client. The subdirectories and files of this directory are shown below. If you would like to see a different directory, you may key the path here and click the “Load” button”. The path is relative to the application directory, absolute paths may also be entered here if you have access to the directory.

The Directory Commands buttons

The “Load” button allow you to view a different directory by first entering the desired path in the Directory Name entry box. The “Up” button allows you to view the parent directory of the currently displayed directory. The “Create Subdirectory” button allows you to create a subdirectory after modifying the Directory Name entry box with the path of the desired new directory. The “Delete directory” button permits you to delete the displayed directory.

The Subdirectories display area

The subdirectories of the current directory are displayed here as buttons. Clicking one of these makes that subdirectory the current directory. If there are no subdirectories in the current directory, the message “There are no subdirectories” appears in this area.

The Files display area

The names of the files in the current directory are displayed as buttons. Clicking one of the buttons, loads the contents of the file into the Text Area below where it may be edited.

The Text Area

This area displays contents of the current file and may be edited.

The File Name entry box

This displays the name of the current file and may be modified to copy the file. This file name is also saved in local storage so that it is “remembered” from session to session.

The File Commands buttons

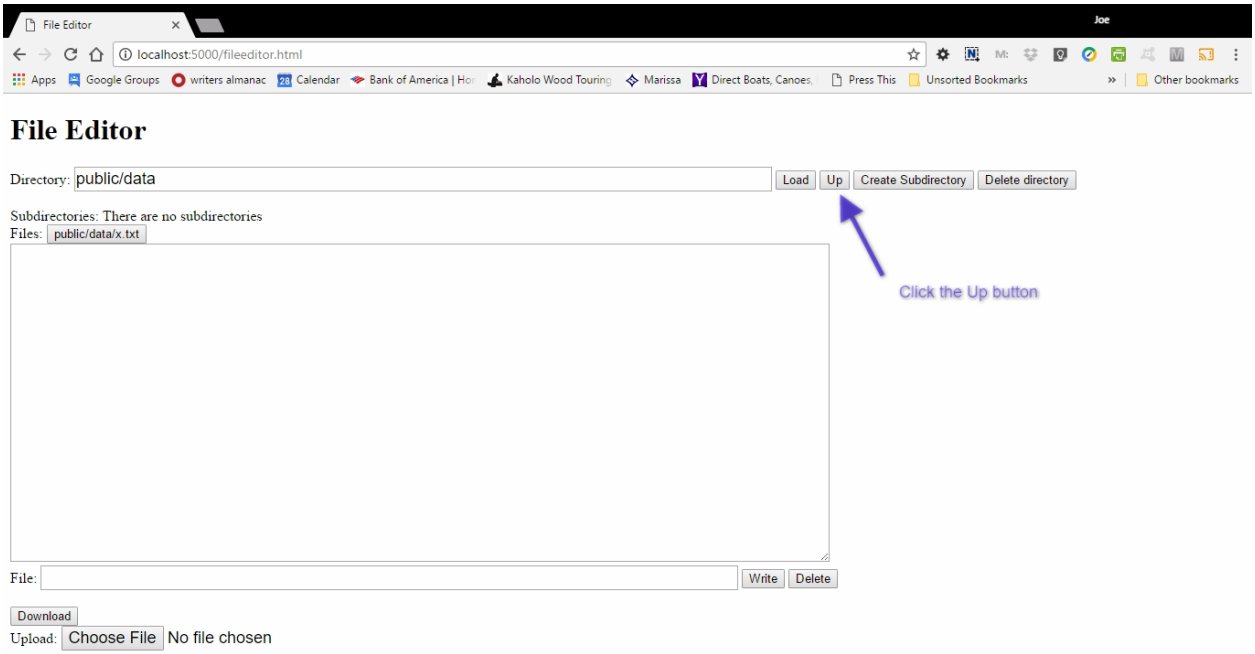
The “Write” button writes the contents of the Text Area to the file currently in the File Name entry box. The “delete” button deletes the file.

The Upload and Download Buttons

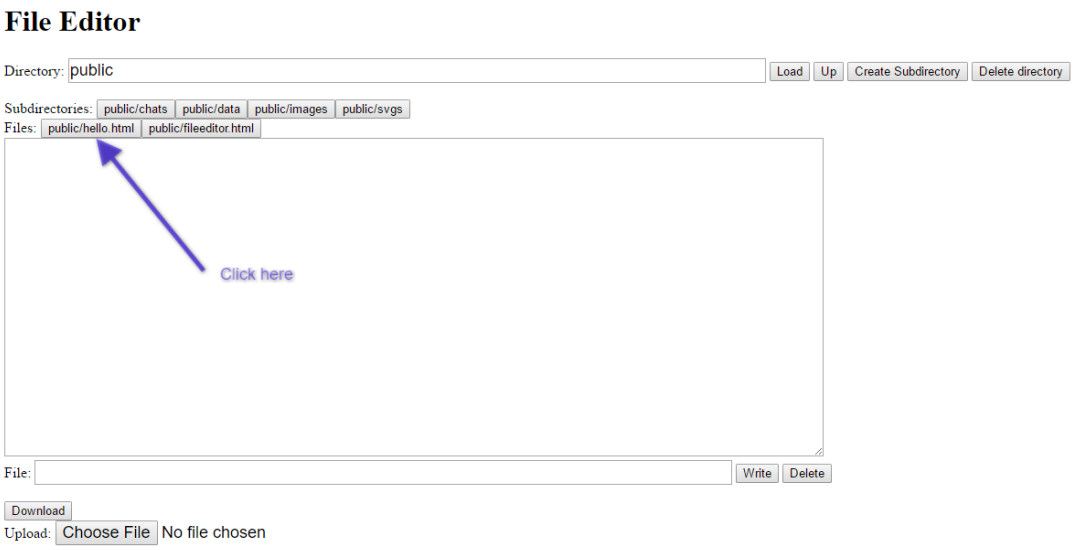
These buttons may be used to upload files from the client's machine and to download files to the client's machine.

Test 2 "Up" button

Click the "Up" Directory Command Button.

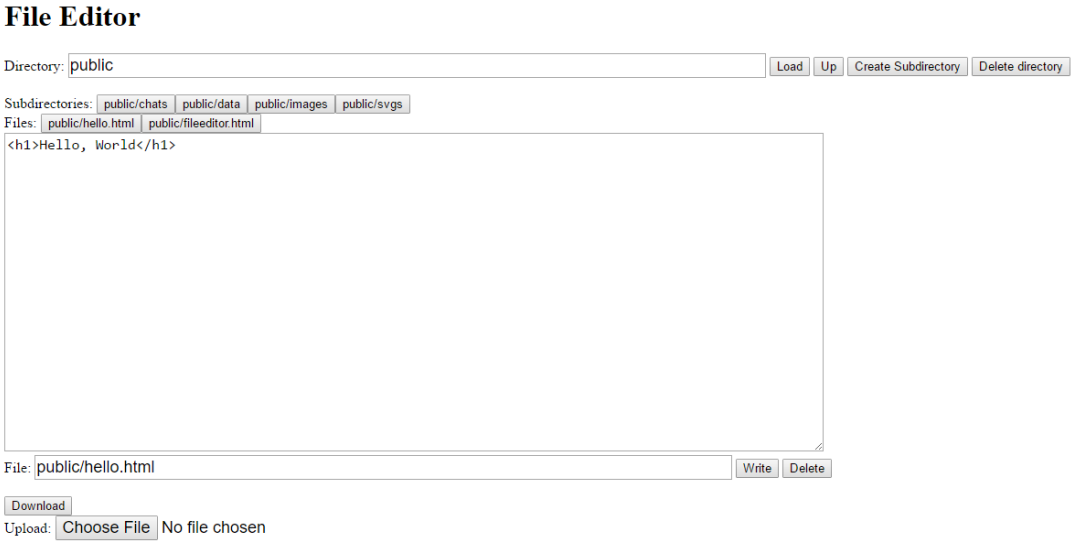


You should see



Click a file button

Click “public/hello.html”. You should see



Test 3 Edit the content and write a new file

Test 4 Edit and save a file

File Editor

Directory: public/data Load Up Create Subdirectory Delete directory

Subdirectories: There are no subdirectories
Files: public/data/x.txt

Click public/data/x.txt file button

```
<h1>Oh, Brave New World, that has such creatures in it!</h1>
```

File: public/tempest.html Write Delete

Download
Upload: Choose File No file chosen

File Editor

Directory: public/data Load Up Create Subdirectory Delete directory

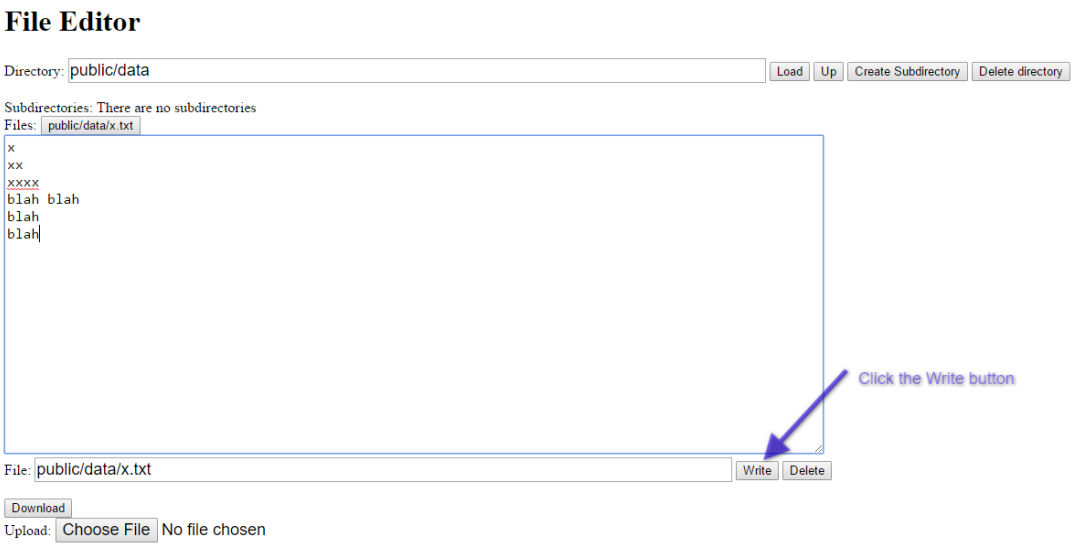
Subdirectories: There are no subdirectories
Files: public/data/x.txt

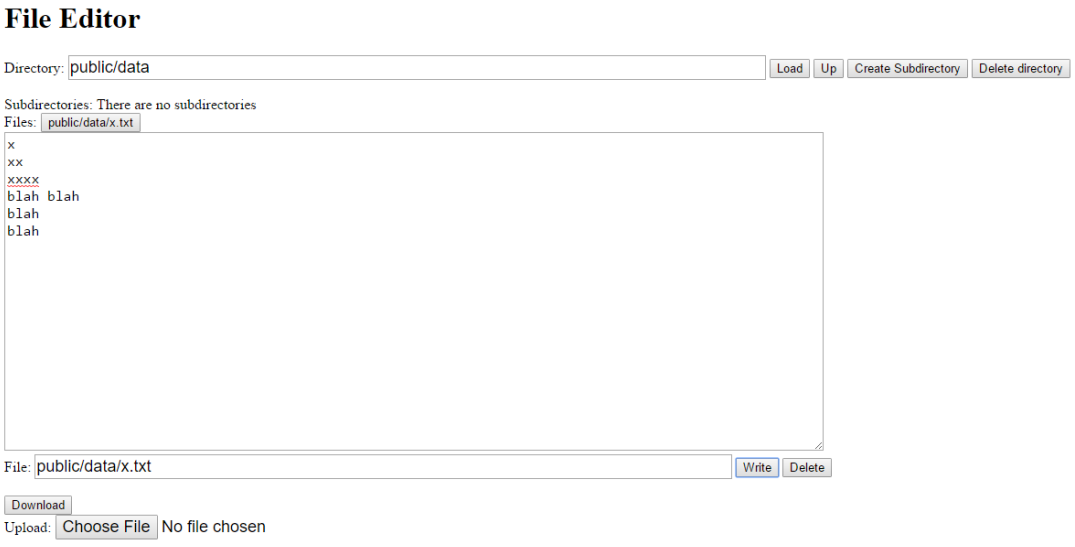
```
x
```

Edit the contents

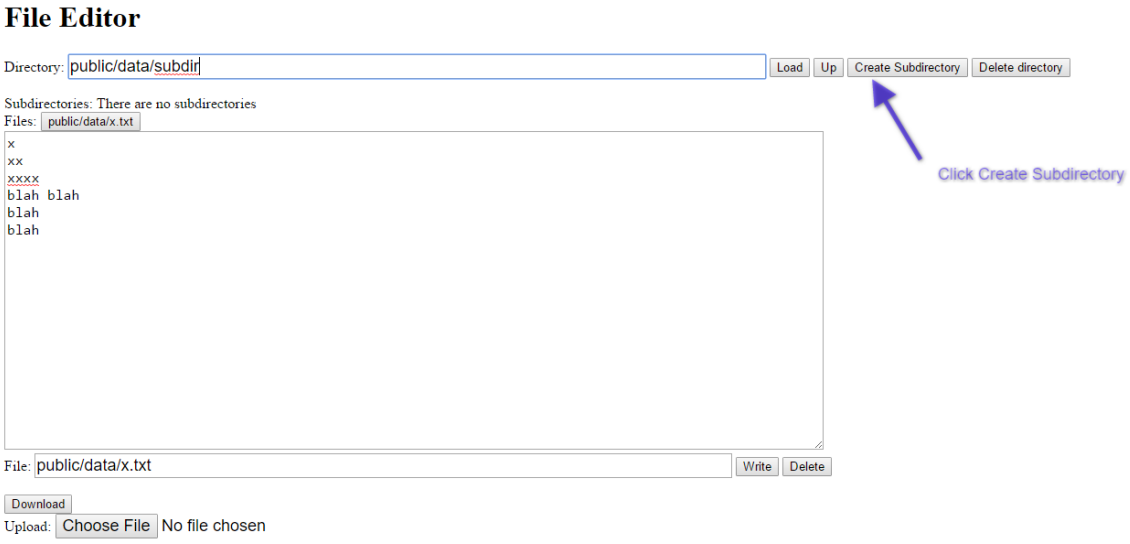
File: public/data/x.txt Write Delete

Download
Upload: Choose File No file chosen

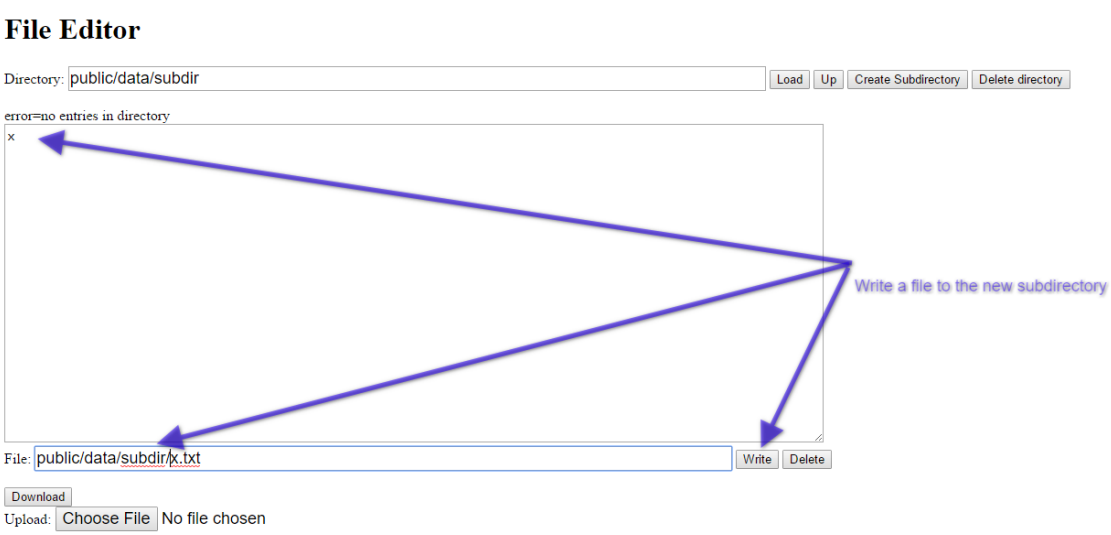

File Editor
Directory: public/data
Load
Up
Create Subdirectory
Delete directory
Subdirectories: There are no subdirectories
Files: public/data/x.txt
x
xx
xxxx
blah blah
blah
blah
Click the Write button
File: public/data/x.txt
Write
Delete
Download
Upload: Choose File No file chosen

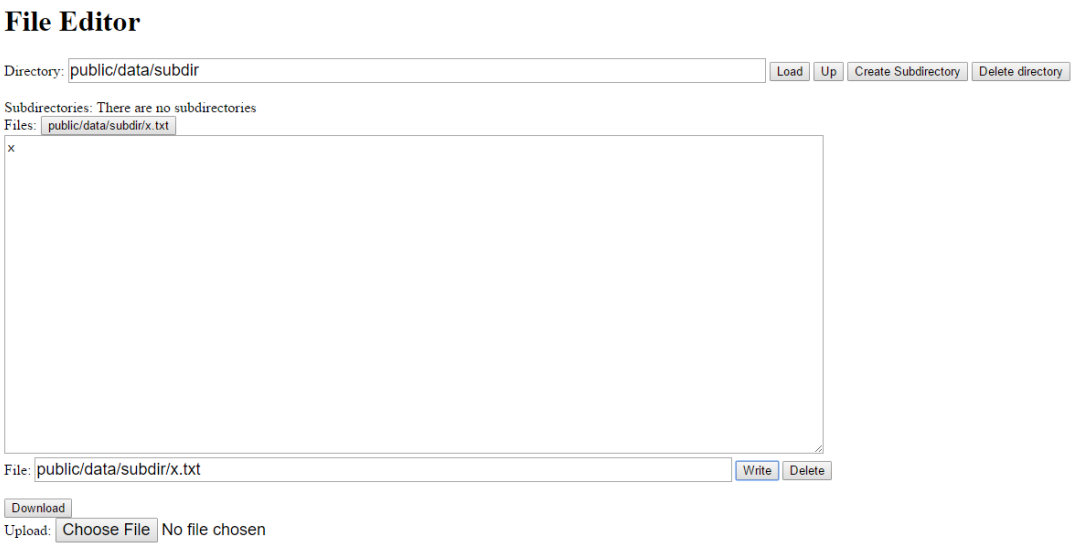

File Editor
Directory: public/data
Load
Up
Create Subdirectory
Delete directory
Subdirectories: There are no subdirectories
Files: public/data/x.txt
x
xx
xxxx
blah blah
blah
blah
File: public/data/x.txt
Write
Delete
Download
Upload: Choose File No file chosen

Test 5 create a subdirectory




File Editor
Directory: public/data
Load
Up
Create Subdirectory
Delete directory
Subdirectories: public/data/subdir
Files: public/data/x.txt
Click the new subdirectory
x
xx
xxxx
blah blah
blah
blah
File: public/data/x.txt
Write
Delete
Download
Upload: Choose File No file chosen


File Editor
Directory: public/data/subdir
Load
Up
Create Subdirectory
Delete directory
error=no entries in directory
x
Write a file to the new subdirectory
File: public/data/subdir/x.txt
Write
Delete
Download
Upload: Choose File No file chosen



Test 6 Delete a File

File Editor

Directory: public/data/subdir Load Up Create Subdirectory Delete directory

error=no entries in directory

x

File: public/data/subdir/x.txt Write Delete

Download

Upload: Choose File No file chosen

Test 7 Delete a Directory

File Editor

Directory: public/data/subdir Load Up Create Subdirectory Delete directory

error=no entries in directory

File: public/data/subdir/x.txt Write Delete

Download

Upload: Choose File No file chosen

File Editor

Directory: public/data/subdir Load Up Create Subdirectory Delete directory

error=ENOENT

Click the Up button

File: public/data/subdir/x.txt Write Delete

Download

Upload: Choose File No file chosen

File Editor

Directory: public/data Load Up Create Subdirectory Delete directory

Subdirectories: There are no subdirectories

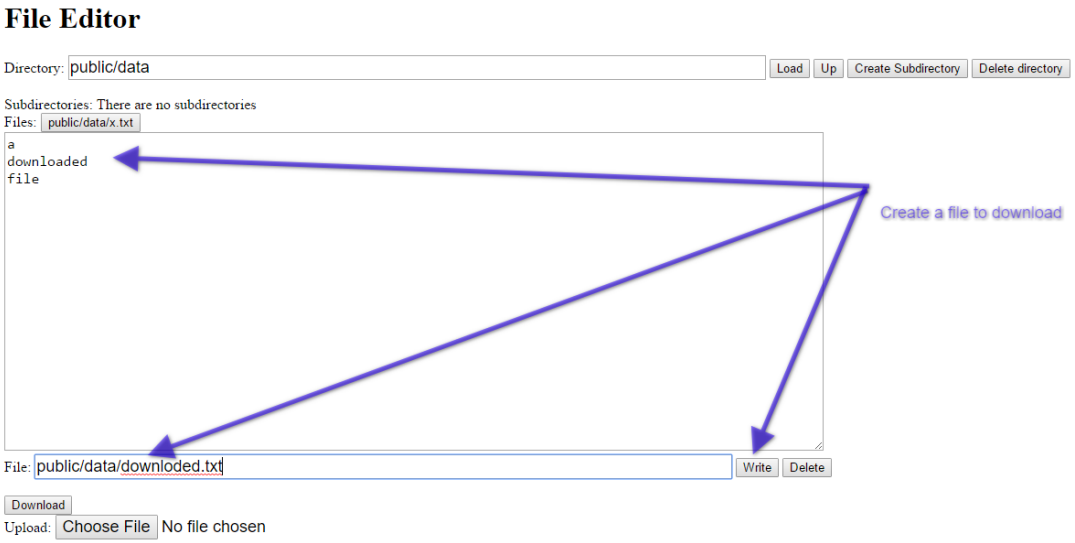
Files: public/data/x.txt

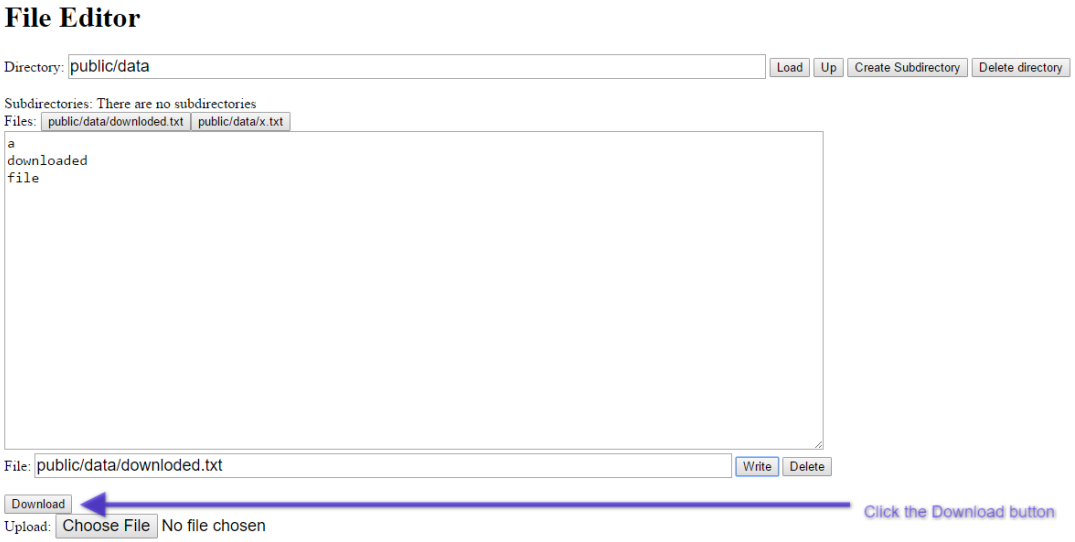
File: public/data/subdir/x.txt Write Delete

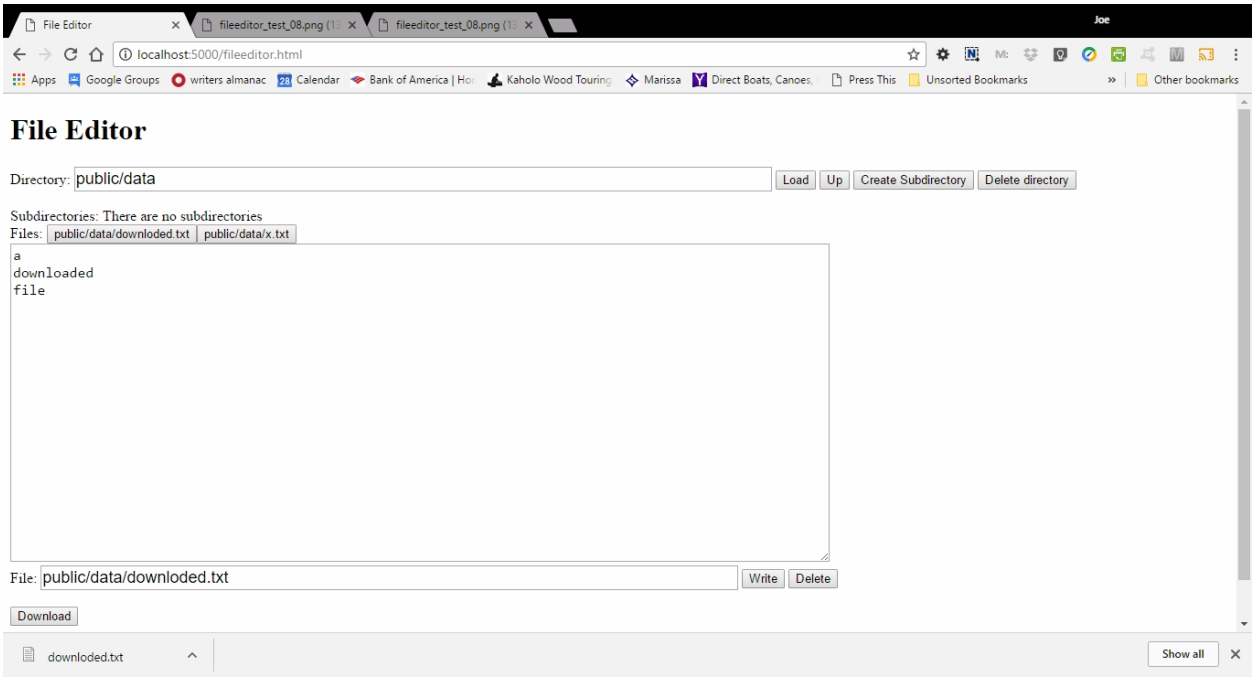
Download

Upload: Choose File No file chosen

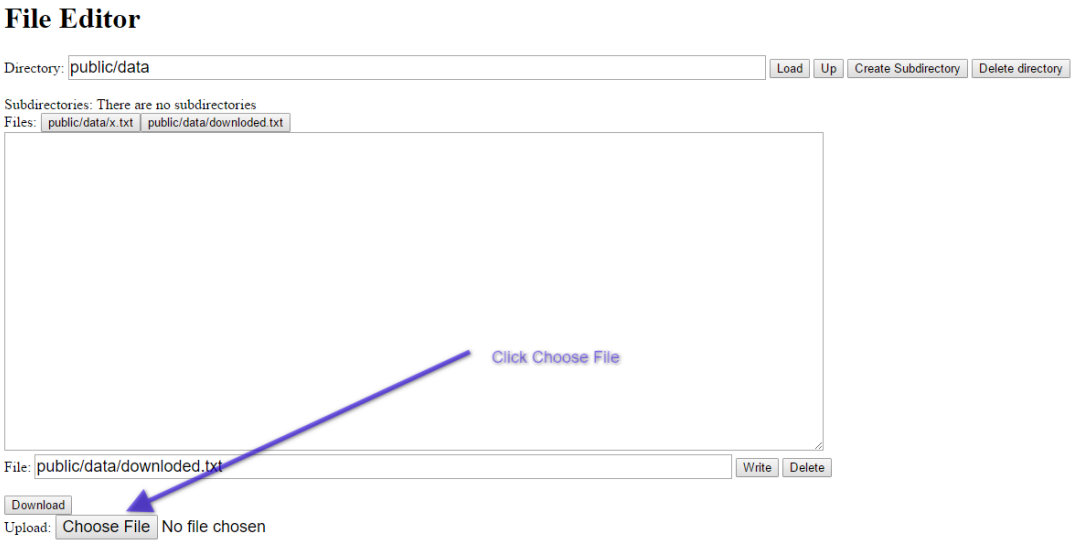
Test 8 Download a File

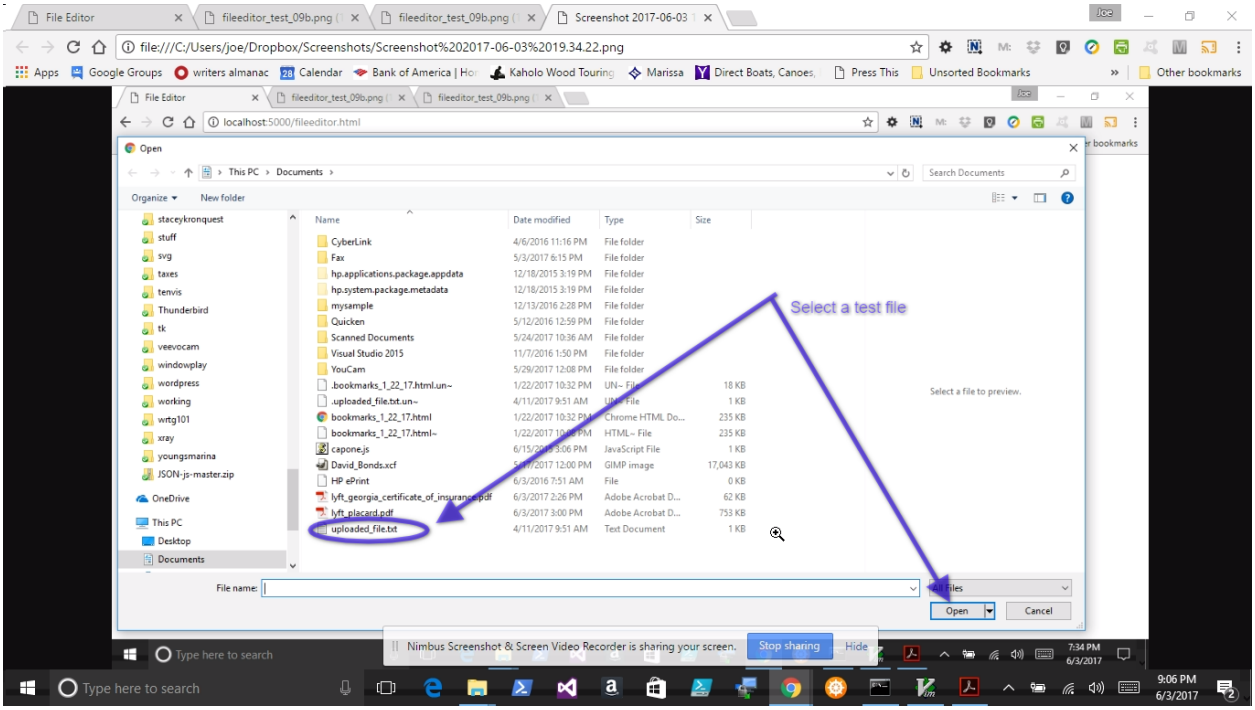


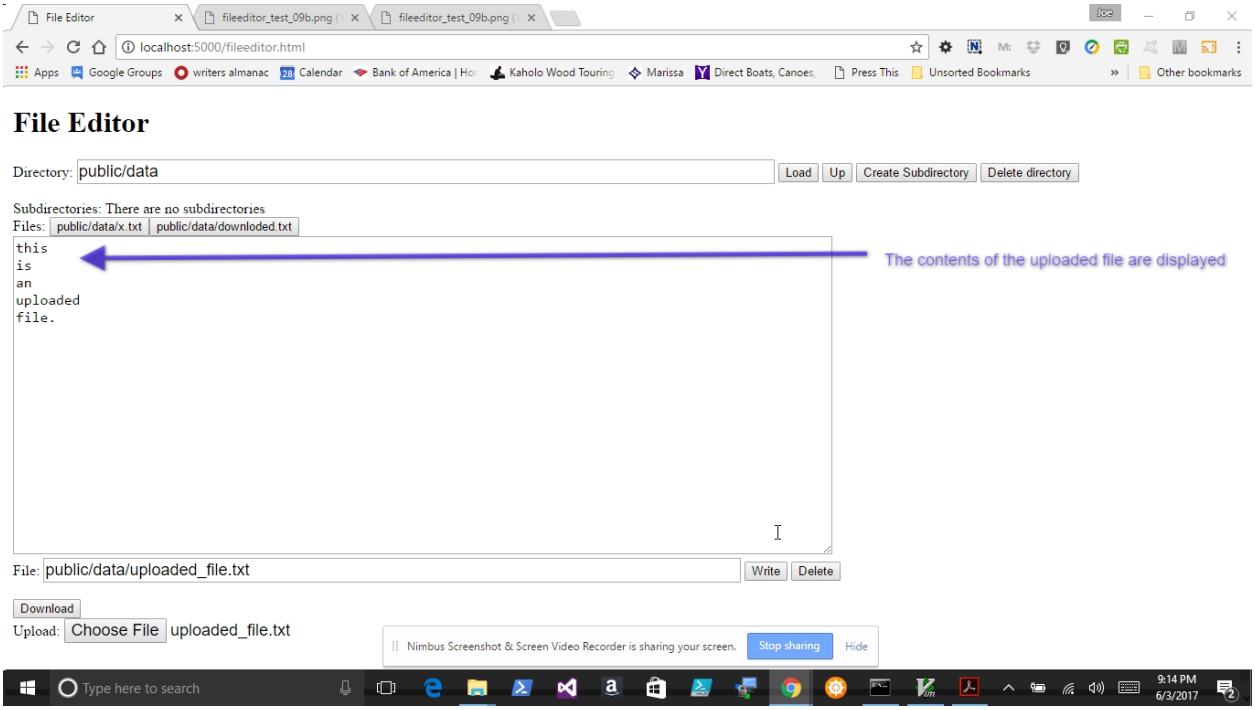




Test 9 Upload a File







The selected file has been uploaded and is now displayed in the text area. You may now write the file to your file system. Fill in the path and file name click write.

Build Your Own Application

You may reuse the server and roll your own client to implement your own application at this point using the techniques you have learned building the sample.

Load Your Application to the Server

Congratulations, you are now an application developer. The following chapters will add a few more tools to your tool chest. I would encourage you to continue learning, perhaps by pursuing the various frameworks for javascript development, mastering graphics packages, etc.

- Now push your application to the server, using the following commands (jrb-sampleapp is your choice of heroku application name):

```
heroku login
heroku create jrb-sampleapp
git add .
git commit -m "initial commit"
git push heroku master
```

- This should look something like this:

```
C:\Users\joe\Dropbox\jrb-sampleapp>heroku login
Enter your Heroku credentials:
Email: danceswithdolphin@gmail.com
Password: **********
Logged in as danceswithdolphin@gmail.com
C:\Users\joe\Dropbox\jrb-sampleapp>heroku create jrb-sampleapp
Creating jrb-sampleapp... done
https://jrb-sampleapp.herokuapp.com/ | https://git.heroku.com/jrb-sampleapp.git
C:\Users\joe\Dropbox\jrb-sampleapp>git add .
warning: LF will be replaced by CRLF in backup/public/chat.html.bak.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in backup/public/data/foo.txt.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in backup/public/data/footies.txt.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in backup/public/foo - Copy.txt.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in backup/public/foo.txt.
The file will have its original line endings in your working directory.
```

```
warning: LF will be replaced by CRLF in backup/public/footies.txt.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in backup/public/index.html.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in backup/root/package.json.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in package.json.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/chat.html.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/data/downloaded.txt.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/images/screenandfish.svg.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/index.html.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/svgplay.svg.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/svgplay2.html.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/svgplay2.svg.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/svgplay3.html.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/svgs/cutedolphin.svg.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/svgs/framed_screen.svg.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/svgs/simple_animation.svg.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in server.js.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in svgs/cutedolphin.svg.
The file will have its original line endings in your working directory.
C:\Users\joe\Dropbox\jrb-sampleapp>git commit -m "initial commit"
[master (root-commit) e3500de] initial commit
warning: LF will be replaced by CRLF in backup/public/chat.html.bak.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in backup/public/data/foo.txt.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in backup/public/data/footies.txt.
The file will have its original line endings in your working directory.
```

```
warning: LF will be replaced by CRLF in backup/public/foo - Copy.txt.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in backup/public/foo.txt.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in backup/public/footies.txt.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in backup/public/index.html.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in backup/root/package.json.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in package.json.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/chat.html.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/data/downloaded.txt.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/images/screenandfish.svg.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/index.html.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/svgplay.svg.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/svgplay2.html.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/svgplay2.svg.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/svgplay3.html.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/svgs/cutedolphin.svg.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/svgs/framed_screen.svg.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in public/svgs/simple_animation.svg.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in server.js.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in svgs/cutedolphin.svg.
The file will have its original line endings in your working directory.
 61 files changed, 3170 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 Procfile
 create mode 100644 arrayplay.js
```

```
create mode 100644 backup/public/chat.html
create mode 100644 backup/public/chat.html.bak
create mode 100644 backup/public/chats/slim pickings.txt
create mode 100644 backup/public/chats/x.txt
create mode 100644 backup/public/chats/y.txt
create mode 100644 backup/public/data/cashflow.txt
create mode 100644 backup/public/data/foo.txt
create mode 100644 backup/public/data/foot.txt
create mode 100644 backup/public/data/footie.txt
create mode 100644 backup/public/data/footies.txt
create mode 100644 backup/public/data/test.txt
create mode 100644 backup/public/data/x.txt
create mode 100644 backup/public/fileeditor - Copy (2).html
create mode 100644 backup/public/fileeditor - Copy.html
create mode 100644 backup/public/fileeditor.html
create mode 100644 backup/public/filelist - Copy.html
create mode 100644 backup/public/filelist.html
create mode 100644 backup/public/foo - Copy.txt
create mode 100644 backup/public/foo.txt
create mode 100644 backup/public/footies.txt
create mode 100644 backup/public/index.html
create mode 100644 backup/public/upload.html
create mode 100644 backup/root/.gitignore
create mode 100644 backup/root/Procfile
create mode 100644 backup/root/package.json
create mode 100644 backup/root/server - Copy.js
create mode 100644 backup/root/server.js
create mode 100644 listassignment.js
create mode 100644 package.json
create mode 100644 public/backup/fileeditor.html
create mode 100644 public/chat.html
create mode 100644 public/chats/Diary.txt
create mode 100644 public/data/downloaded.txt
create mode 100644 public/fileeditor.html
create mode 100644 public/graphicseditor.html
create mode 100644 public/horse.mp3
create mode 100644 public/horse.ogg
create mode 100644 public/image_with_path.svg
create mode 100644 public/images/cutedolphin.png
create mode 100644 public/images/screenandfish.svg
create mode 100644 public/index.html
create mode 100644 public/listassignment.html
```

```
 create mode 100644 public/listassignment.js
 create mode 100644 public/mouseplay.html
 create mode 100644 public/svgplay.svg
 create mode 100644 public/svgplay2.html
 create mode 100644 public/svgplay2.svg
 create mode 100644 public/svgplay3.html
 create mode 100644 public/svgs/arc.svg
 create mode 100644 public/svgs/cutedolphin.svg
 create mode 100644 public/svgs/framed_screen.svg
 create mode 100644 public/svgs/mdn_animatemotion_example.svg
 create mode 100644 public/svgs/simple_animation.svg
 create mode 100644 public/twirlly.html
 create mode 100644 public/twirlly.svg
 create mode 100644 public/whinny.html
 create mode 100644 server.js
 create mode 100644 svgs/cutedolphin.svg

C:\Users\joe\Dropbox\jrb-sampleapp>git push heroku master
Counting objects: 66, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (53/53), done.
Writing objects: 100% (66/66), 97.91 KiB | 0 bytes/s, done.
Total 66 (delta 7), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Node.js app detected
remote:
remote: -----> Creating runtime environment
remote:
remote:        NPM_CONFIG_LOGLEVEL=error
remote:        NPM_CONFIG_PRODUCTION=true
remote:        NODE_VERBOSE=false
remote:        NODE_ENV=production
remote:        NODE_MODULES_CACHE=true
remote:
remote: -----> Installing binaries
remote:        engines.node (package.json):  unspecified
remote:        engines.npm (package.json):   unspecified (use default)
remote:
remote:        Resolving node version 6.x via semver.io...
remote:        Downloading and installing node 6.10.2...
```

```
remote:        Using default npm version: 3.10.10
remote:
remote: -----> Restoring cache
remote:        Skipping cache restore (new runtime signature)
remote:
remote: -----> Building dependencies
remote:        Installing node modules (package.json)
remote:        jrb-sampleapp@1.0.0 /tmp/build_b4968d8bfa1d70bfba5ee733138424b8
remote:        +-- express@4.15.2
remote:        +-- accepts@1.3.3
remote:        ¦ +-- mime-types@2.1.15
remote:        ¦ ¦ +-- mime-db@1.27.0
remote:        ¦ +-- negotiator@0.6.1
remote:        +-- array-flatten@1.1.1
remote:        +-- content-disposition@0.5.2
remote:        +-- content-type@1.0.2
remote:        +-- cookie@0.3.1
remote:        +-- cookie-signature@1.0.6
remote:        +-- debug@2.6.1
remote:        ¦ +-- ms@0.7.2
remote:        +-- depd@1.1.0
remote:        +-- encodeurl@1.0.1
remote:        +-- escape-html@1.0.3
remote:        +-- etag@1.8.0
remote:        +-- finalhandler@1.0.1
remote:        ¦ +-- debug@2.6.3
remote:        ¦ +-- unpipe@1.0.0
remote:        +-- fresh@0.5.0
remote:        +-- merge-descriptors@1.0.1
remote:        +-- methods@1.1.2
remote:        +-- on-finished@2.3.0
remote:        ¦ +-- ee-first@1.1.1
remote:        +-- parseurl@1.3.1
remote:        +-- path-to-regexp@0.1.7
remote:        +-- proxy-addr@1.1.4
remote:        ¦ +-- forwarded@0.1.0
remote:        ¦ +-- ipaddr.js@1.3.0
remote:        +-- qs@6.4.0
remote:        +-- range-parser@1.2.0
remote:        +-- send@0.15.1
remote:        ¦ +-- destroy@1.0.4
remote:        ¦ +-- http-errors@1.6.1
```

```
remote:        ¦ ¦ +-- inherits@2.0.3
remote:        ¦ +-- mime@1.3.4
remote:        +-- serve-static@1.12.1
remote:        +-- setprototypeof@1.0.3
remote:        +-- statuses@1.3.1
remote:        +-- type-is@1.6.15
remote:        ¦ +-- media-typer@0.3.0
remote:        +-- utils-merge@1.0.0
remote:        +-- vary@1.1.1
remote:
remote:
remote: -----> Caching build
remote:        Clearing previous node cache
remote:        Saving 2 cacheDirectories (default):
remote:        - node_modules
remote:        - bower_components (nothing to cache)
remote:
remote: -----> Build succeeded!
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:        Done: 13.8M
remote: -----> Launching...
remote:        Released v3
remote:        https://jrb-sampleapp.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/jrb-sampleapp.git
 * [new branch]      master -> master

C:\Users\joe\Dropbox\jrb-sampleapp>
```

You should now be able to enter “http://jrb-sampleapp.herokuapp.com” in the url bar of chrome and see your application in action.

Congratulations, you are now an application programmer.