COMSATS UNIVERSITY, ISLAMABAD

DATA COMMUNICATION AND NETWORKS

Lab Project Report

FullStack FTP Server with Socket-based Intercommunication

CLASS: 5-B

Members:	**Reg. No:**
Syed Muhammad Irtesam	FA17-BCS-088
Mutahhar Bin Muzaffar	FA17-BCS-058
Uzayr Qamar	FA17-BCS-096

Project Supervisor: Ahmed Kakakhail

Contents

1 Introduction

This project intends to provide FTP-based authentication on a file server while allowing users to remotely download and upload files to the server. We will utilize sockets to simulate a connection between two FTP clients by providing a simple terminal interface for file transfer between FTPs. On the other hand, we will provide users of the FTP a fully functional GUI to download and upload files to a central FTP server without the need of a command-line interface. This solves a scenario where a company might utilize multiple FTP servers in their infrastructure. Thus, we are covering all paradigms of FTP usage in tech sectors. By building a neat and streamlined web interface for this project, we can do away with clunky and limited tools like FileZilla and SyncTrayzor.

2 Functional Requirements

This system solves the following functional requirements:

- Provide application-scaling user authentication
- Provide a streamlined interface for management of files
- Provide easy controls for file upload and download
- Display basic info about all files in tabulated form
- Provide dynamic and accurate front-end updates about file-system to the user
- Allow multiple FTP servers to inter-communicate
- Allow remote clients to access files through command-line
- Provide a simple RESTful API that can be easily exposed with proper OAuth

3 Tools and Technologies

3.1 Tools

- Visual Studio Code
- Ubuntu
- Windows Powershell
- apt/pacman/chocolatey
- cURL (HTTP Client)
- Windows Subsystem for Linux (WSL)
- IDEA IntelliJ
- Git Bash (Github)
- LaTeX/TeX Live

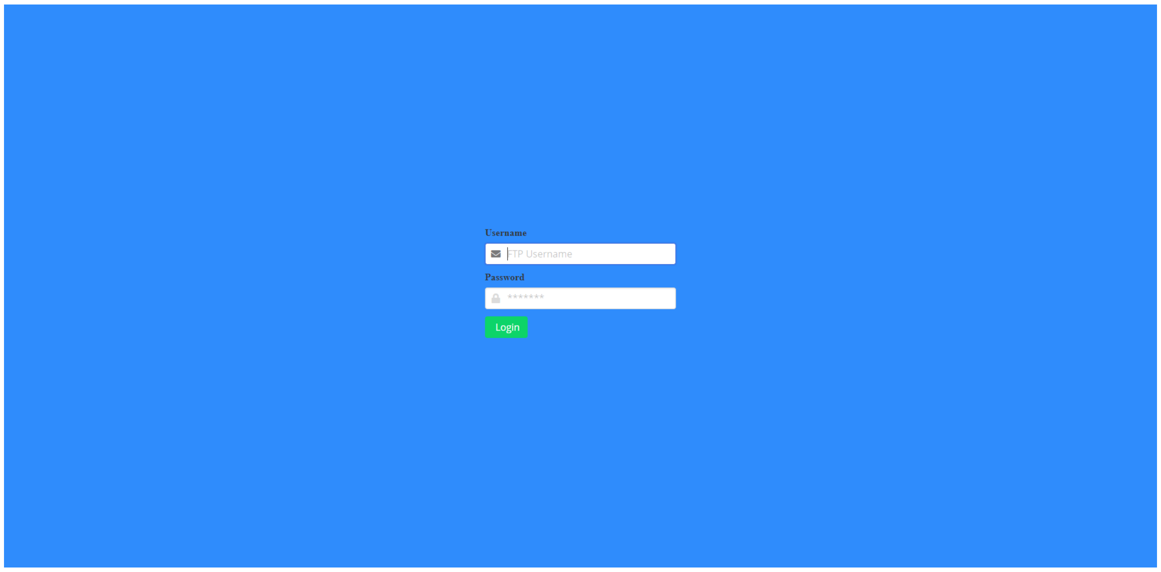
3.2 Technologies

- NodeJS
- ExpressJS
- VueJS
- OpenJDK
- Vuex/Vue Router
- bulma.io
- yarn

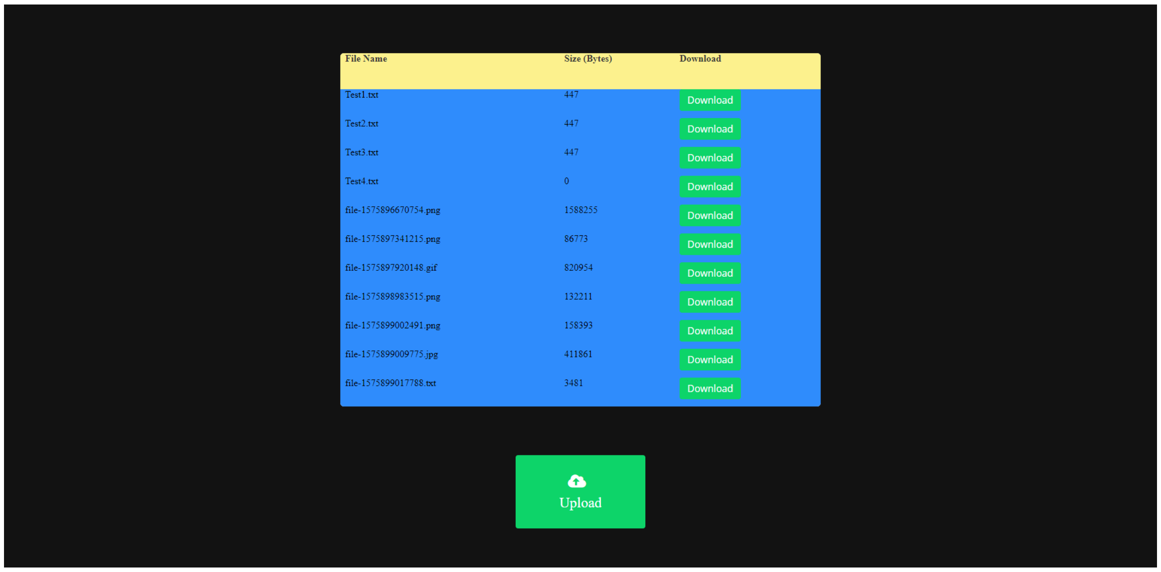
- ESLint
- webpack
- babel.js
- CSS Loader/SASS Loader/Vue Loader

4 Screenshots

4.1 FTP Login Page

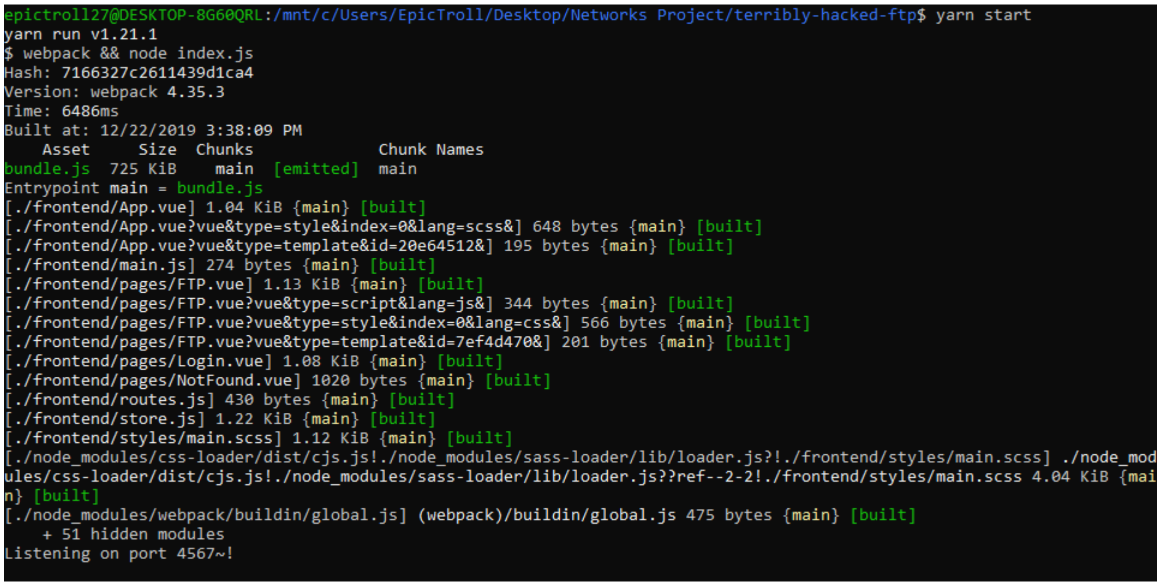


4.2 FTP Main Interface



File Name	Size (Bytes)	Download
Test1.txt	447	Download
Test2.txt	447	Download
Test3.txt	447	Download
Test4.txt	0	Download
file-1575896670754.png	1588255	Download
file-1575897341215.png	86773	Download
file-1575897920148.gif	820954	Download
file-1575898983515.png	132211	Download
file-1575899002491.png	158393	Download
file-1575899009775.jpg	411861	Download
file-1575899017788.txt	3481	Download

4.3 Terminal with FTP loaded



```
epictroll27@DESKTOP-8G60QRL:/mnt/c/Users/EpicTroll/Desktop/Networks Project/terribly-hacked-ftp$ yarn start
yarn run v1.21.1
$ webpack && node index.js
Hash: 7166327c2611439d1ca4
Version: webpack 4.35.3
Time: 6486ms
Built at: 12/22/2019 3:38:09 PM
    Asset      Size  Chunks             Chunk Names
bundle.js   725 KiB    main  [emitted]  main
Entrypoint main = bundle.js
[./frontend/App.vue] 1.04 KiB {main} [built]
[./frontend/App.vue?vue&type=style&index=0&lang=scss&] 648 bytes {main} [built]
[./frontend/App.vue?vue&type=template&id=20e64512&] 195 bytes {main} [built]
[./frontend/main.js] 274 bytes {main} [built]
[./frontend/pages/FTP.vue] 1.13 KiB {main} [built]
[./frontend/pages/FTP.vue?vue&type=script&lang=js&] 344 bytes {main} [built]
[./frontend/pages/FTP.vue?vue&type=style&index=0&lang=css&] 566 bytes {main} [built]
[./frontend/pages/FTP.vue?vue&type=template&id=7ef4d470&] 201 bytes {main} [built]
[./frontend/pages/Login.vue] 1.08 KiB {main} [built]
[./frontend/pages/NotFound.vue] 1020 bytes {main} [built]
[./frontend/routes.js] 430 bytes {main} [built]
[./frontend/store.js] 1.22 KiB {main} [built]
[./frontend/styles/main.scss] 1.12 KiB {main} [built]
[./node_modules/css-loader/dist/cjs.js!./node_modules/sass-loader/lib/loader.js?!./frontend/styles/main.scss] ./node_mod
ules/css-loader/dist/cjs.js!./node_modules/sass-loader/lib/loader.js??ref--2-2!./frontend/styles/main.scss 4.04 KiB {mai
n} [built]
[./node_modules/webpack/buildin/global.js] (webpack)/buildin/global.js 475 bytes {main} [built]
    + 51 hidden modules
Listening on port 4567~!
```

4.4 Java FTP Server Socket

4.5 Java FTP Client Socket

5 Getting Started

After installing NodeJS, yarn, IDEA IntelliJ, follow these instructions from the README.md file included in the project:

```
#Copy and edit config
$ cp sample.config.js config.js && $EDITOR config.js
# Install dependencies
$ yarn
# Build and run
$ yarn start
```

After the application is deployed, you should start the Java server and open a client connection to receive files from other FTPs. The ports and IP addresses must be edited manually inside the Java files.

6 Source Code

6.1 index.js

```
const fs = require('fs');
const path = require('path');
const express = require('express'); // Web server
const config = require('./config'); // Generic configuration
const sirv = require('sirv'); // Static file middleware
const api = require('./routes/api');

const indexPage = fs.readFileSync(path.join(config.publicDir, 'index.html'));
```

```
// Define the main application
const app = express({
onNoMatch: (request, response) => response.end(indexPage),
});

// Set up global middlewares
app.use(sirv(config.publicDir, {dev: true}));

// Register the API routes and auth routes
app.use('/api', api);
app.set('view engine', 'ejs');

// Start the server
app.listen(config.port, () => {
console.log(`Listening on port ${config.port}~!`);
});
```

6.2 api.js

```
const apiApp = require('express')();
const fs = require('fs');
const config = require('../config');
const path = require('path');
const util = require('util');
const multer = require('multer');

const storage = multer.diskStorage({
destination: config.serverDir,
filename: function(req, file, cb){
  cb(null,file.fieldname + '-' + Date.now() + path.extname(file.originalname));
}
  });

  const upload = multer({
storage: storage,
  }).single('file');

apiApp.get('/dirlist', (req, res) => {
let dirArr = [];
let files = [];
const readDir = util.promisify(fs.readdir);
const stat = util.promisify(fs.stat);

readDir(config.serverDir).then(async (items)=> {
files = items.slice(items);
}).then(async () => {
var count = 0;
for (var i=0; i<files.length; i++) {
var file = config.serverDir + '/' + files[i];
const promise = await stat(file).then((data)=> {
return {
"id" : ++count,
"name": path.basename(file),
"size": data["size"].toString(),
};
});
dirArr.push(promise);
}
}).then(()=> {
```

```
res.json(dirArr);
res.end();
});
});

apiApp.get('/download/:fileName', (req,res) => {
const file = config.serverDir + '/' + req.params.fileName;
res.download(file);
});

apiApp.post('/upload', (req,res) => {
upload(req, res, (err) => {
if (err) {
throw err;
}
res.end();
  });
});




function checkFileType(file, cb){
// Allowed ext
const filetypes = /jpeg|jpg|png|gif/;
// Check ext
const extname = filetypes.test(path.extname(file.originalname).toLowerCase());
// Check mime
const mimetype = filetypes.test(file.mimetype);

if(mimetype && extname){
  return cb(null,true);
} else {
  cb('Error: Images Only!');
}
  }

module.exports = apiApp;
```

6.3 Vuex Store

```
import Vue from 'vue';
import Vuex from 'vuex';
const axios = require('axios');

Vue.use(Vuex);

async function makeRequest (path, method = 'GET', body) {
if (typeof body === 'object' && body != null) {
body = JSON.stringify(body);
}
try {
const result = await fetch(path, {method, body});
if (!result.ok) {
const json = await result.json();
throw json.error;
}
if (result.status === 204) {
return;
}
return await result.json();
```

```
} catch (error) {
window.alert(error);
throw error;
}
}

const store = new Vuex.Store({
state: {
fileData: null,
auth: false,
downloadedFile: null,
},
mutations: {
GET_FILES (state,fileData) {
state.fileData = fileData;
},
},
getters: {
isAuth(state) {
return state.auth;
},
},
actions: {
async getFiles({commit}) {
const files = await makeRequest('/api/dirlist');
commit('GET_FILES',files);
},
async downloadFile({commit},fileName) {
setTimeout(() => {
const response = {
  file: ‘/api/download/${fileName}‘,
};
window.open(response.file);
  }, 50);
},
},
});

export default store;
```

7 Login.vue

```
<template>
<section class="hero is-primary is-fullheight">
  <div class="hero-body">
<div class="container">
  <div class="columns is-centered">
<div class="column is-5-tablet is-4-desktop is-3-widescreen">
<div class="field">
  <label for="" class="label">Username</label>
  <div class="control has-icons-left">
<input type="text" placeholder="FTP Username"
 v-model="username" class="input" required>
<span class="icon is-small is-left">
  <i class="fa fa-envelope"></i>
</span>
  </div>
</div>
<div class="field">
  <label for="" class="label">Password</label>
  <div class="control has-icons-left">
<input type="password" placeholder="*******"
 v-model="password" class="input" required>
<span class="icon is-small is-left">
  <i class="fa fa-lock"></i>
</span>
  </div>
</div>
<div class="field">
  <button @click="doAuth" class="button is-success">
Login
  </button>
</div>
<div class="field">
  <i class="label">{{message}}</i>
</div>
</div>
  </div>
</div>
  </div>
</section>
</template>

<script>
const config = require('../../config');

export default {
  data () {
return {
  username: '',
  password: '',
  message: '',
}
  },
  methods: {
doAuth() {
  const user = config.users.admin;
  if (user.username==this.username && user.password == this.password) {
this.$router.push({path:'/FTP'});
  }
  else {
this.message = "Wrong username or password.";
  }
},
  },
}
</script>
```

7.1 FTP.vue

```
<template>
<section class="hero has-background-black-bis is-fullheight">
  <div class="hero-body">
<div v-if="fileData" class="container">
<table class="is-full-width is-scrollable">
  <thead>
<tr>
  <th>File Name</th>
  <th>Size (Bytes)</th>
  <th>Download</th>
```

```
</tr>
  </thead>
  <tbody v-for="files in fileData" :key="files.id"
   class="has-background-primary">
<tr>
  <td>{{files.name}}</td>
  <td>{{files.size}}</td>
  <td><button v-on:click="downloadFileSubmit(files.id)"
   class="button is-success">Download</button></td>
</tr>
  </tbody>
</table>
  </div>
  </div>
  <div class="container">
  <div class="file is-boxed is-centered is-large is-success">
  <label class="file-label">
  <input v-on:change="uploadFile" class="file-input" type="file"
   name="file" id="file">
  <span class="file-cta">
<span class="file-icon">
  <i class="fas fa-cloud-upload-alt"></i>
</span>
<span class="file-label">
  Upload
</span>
  </span>
  </label>
</div>
</div>
```

```
</section>
</template>

<script>
import {mapState, mapGetters, mapActions} from 'vuex';
const axios = require('axios');

export default {
  data () {
return {
  file: '',
}
  },
  computed: {
...mapState(['fileData','auth']),
...mapGetters(['isAuth']),
  },
  methods: {
...mapActions(['getFiles','downloadFile']),
downloadFileSubmit (fileId) {
  setTimeout(async () => {
const file = async () => {
  return new Promise ((resolve,reject) => {
const file = this.fileData.find(file => file.id == fileId);
resolve(file.name);
  });
};
var promise = file();
promise.then(async (result)=> {
  await this.downloadFile(result);
})
  });
},
uploadFile () {
  var formData = new FormData();
  var imagefile = document.querySelector('#file');
  formData.append("file", imagefile.files[0]);
  try {
setTimeout (async () => {
  axios.post('/api/upload', formData, {
headers: {
  'Content-Type': 'multipart/form-data'
}
  });
});
  }
  finally {
setTimeout(async () => {
  await this.getFiles();
});
  }
},
  },
  mounted : function () {
if (!this.fileData) {
  setTimeout(async () => {
await this.getFiles();
  });
  }
  },
}
</script>

<style>
table {
  border-spacing: 1;
  border-collapse: collapse;
  background: white;
  border-radius: 6px;
  overflow: hidden;
  max-width: 800px;
  width: 100%;
  margin: 0 auto;
  position: relative;
}
table * {
  position: relative;
}
table td, table th {
  padding-left: 8px;
}
table thead tr {
  height: 60px;
  background: #FFED86;
  font-size: 16px;
}
table tbody tr {
  height: 48px;
  border-bottom: 1px solid #E3F1D5;
}
```

```
table tbody tr:last-child {
  border: 0;
}
table td, table th {
  text-align: left;
}
table td.l, table th.l {
  text-align: right;
}
table td.c, table th.c {
  text-align: center;
}
table td.r, table th.r {
  text-align: center;
}

@media screen and (max-width: 35.5em) {
  table {
display: block;
  }
  table > *, table tr, table td, table th {
display: block;
  }
  table thead {
display: none;
  }
  table tbody tr {
height: auto;
padding: 8px 0;
  }
  table tbody tr td {
padding-left: 45%;
margin-bottom: 12px;
  }
  table tbody tr td:last-child {
margin-bottom: 0;
  }
  table tbody tr td:before {
position: absolute;
font-weight: 700;
width: 40%;
left: 10px;
top: 0;
  }
  table tbody tr td:nth-child(1):before {
content: "Code";
  }
  table tbody tr td:nth-child(2):before {
content: "Stock";
  }
  table tbody tr td:nth-child(3):before {
content: "Cap";
  }
  table tbody tr td:nth-child(4):before {
content: "Inch";
  }
  table tbody tr td:nth-child(5):before {
content: "Box Type";
  }
}
```

```
body {
  background: #9BC86A;
  font: 400 14px 'Calibri','Arial';
  padding: 20px;
}
</style>
```

7.2 SASS Stylesheet

```
// This file is loaded by webpack as a prelude to all other sass snippets. It
// should not contain any actual styles, because any styles here will be
// included with every other sass snippet in the app. Needless to day, that is
// not how we want that to work.

// Customize Bulma variables
$family-sans-serif: "Open Sans", "Helvetica", "Arial", sans-serif;
$family-monospace: "Source Code Pro", "Menlo", "Consolas", monospace;
$primary: #0094FF;

// Smaller vertical section padding
$section-padding: 1.5rem;

$gl-ms          : "screen and (max-width: 23.5em)"; // up to 360px
$gl-xs          : "screen and (max-width: 35.5em)"; // up to 568px
$gl-sm          : "screen and (max-width: 48em)";   // max 768px
$gl-md          : "screen and (max-width: 64em)";   // max 1024px
$gl-lg          : "screen and (max-width: 80em)";   // max 1280px

// table style
```

```
table        {
  border-spacing: 1;
  border-collapse: collapse;
  background:white;
  border-radius:6px;
  overflow:hidden;
  max-width:800px;
  width:100%;
  margin:0 auto;
  position:relative;

  *               { position:relative }

  td,th           { padding-left:8px}

  thead tr        {
height:60px;
background:#FFED86;
font-size:16px;
  }

  tbody tr        { height:48px; border-bottom:1px solid #E3F1D5 ;
&:last-child  { border:0; }
  }

 td,th  { text-align:left;
&.l  { text-align:right }
&.c  { text-align:center }
&.r  { text-align:center }
}
}



@media #{$gl-xs}              {

  table              { display:block;
  > *,tr,td,th             { display:block }

thead                    { display:none }
tbody tr                 { height:auto; padding:8px 0;
  td                     { padding-left:45%; margin-bottom:12px;
&:last-child           { margin-bottom:0 }
&:before               {
  position:absolute;
  font-weight:700;
  width:40%;
  left:10px;
  top:0
}

&:nth-child(1):before { content:"Code";}
&:nth-child(2):before { content:"Stock";}
&:nth-child(3):before { content:"Cap";}
&:nth-child(4):before { content:"Inch";}
&:nth-child(5):before { content:"Box Type";}
  }
}
  }
}
```

```
// body style

body              {
background:#9BC86A;
font:400 14px 'Calibri','Arial';
padding:20px;
  }

  blockquote {
color:white;
text-align:center;
  }

// Load other Bulma variables
@import "~bulma/sass/utilities/all";
```

7.3 Java Client Multithreading Class

```
package ftp_server;

import java.io.*;
import java.net.*;
import java.util.logging.Level;
import java.util.logging.Logger;

public class CLIENTConnection implements Runnable {

private Socket clientSocket;
private BufferedReader in = null;
private String UPLOAD_PATH = "..\\..\\Server\\";

public CLIENTConnection(Socket client) {
this.clientSocket = client;
}

@Override
public void run() {
try {
in = new BufferedReader(new InputStreamReader(
clientSocket.getInputStream()));
String clientSelection;
while ((clientSelection = in.readLine()) != null) {
switch (clientSelection) {
case "1":
receiveFile();
break;
case "2":
String outGoingFileName;
while ((outGoingFileName = in.readLine()) != null) {
sendFile(outGoingFileName);
}

break;
default:
System.out.println("Incorrect command received.");
break;
}
in.close();
break;
```

```
}

} catch (IOException ex) {
Logger.getLogger(CLIENTConnection.class.getName()).log(Level.SEVERE, null, ex);
}
}

public void receiveFile() {
try {
int bytesRead;

DataInputStream clientData = new DataInputStream(clientSocket.getInputStream());

String fileName = clientData.readUTF();
File directory = new File(UPLOAD_PATH);
if (!directory.isDirectory()){
directory.mkdir();
directory  = new File(UPLOAD_PATH + fileName);
}
else{
directory = new File(UPLOAD_PATH + fileName);
}
OutputStream output = new FileOutputStream(directory);
long size = clientData.readLong();
byte[] buffer = new byte[1024];
while (size > 0 && (bytesRead = clientData.read(buffer, 0, (int) Math.min(buffer.len
output.write(buffer, 0, bytesRead);
size -= bytesRead;
}
```

```
output.close();
clientData.close();

System.out.println("File " + fileName + " received from client.");
} catch (IOException ex) {
System.err.println("Client error. Connection closed.");
}
}

public void sendFile(String fileName) {
try {
//handle file read
File myFile = new File(UPLOAD_PATH + fileName);
byte[] mybytearray = new byte[(int) myFile.length()];

FileInputStream fis = new FileInputStream(myFile);
BufferedInputStream bis = new BufferedInputStream(fis);
//bis.read(mybytearray, 0, mybytearray.length);

DataInputStream dis = new DataInputStream(bis);
dis.readFully(mybytearray, 0, mybytearray.length);

//handle file send over socket
OutputStream os = clientSocket.getOutputStream();

//Sending file name and file size to the server
DataOutputStream dos = new DataOutputStream(os);
dos.writeUTF(myFile.getName());
dos.writeLong(mybytearray.length);
dos.write(mybytearray, 0, mybytearray.length);
dos.flush();
System.out.println("File " + fileName + " sent to client.");
} catch (Exception e) {
System.err.println("File does not exist!");
}
}
}
```

7.4 Java Server Socket Class

```
package ftp_server;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class FileServer {

private static ServerSocket serverSocket;
private static Socket clientSocket = null;

public static void main(String[] args) throws IOException {



try {
serverSocket = new ServerSocket(4444);
System.out.println("Server started.");
} catch (Exception e) {
System.err.println("Port already in use.");
System.exit(1);
}

while (true) {
try {
clientSocket = serverSocket.accept();
System.out.println("Accepted connection : " + clientSocket);

Thread t = new Thread(new CLIENTConnection(clientSocket));

t.start();

} catch (Exception e) {
System.err.println("Error in connection attempt.");
}
}
}
}
```

7.5 Java Client Socket Class

```
package ftp_server;

import java.io.*;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;

public class FileClient {

private static Socket sock;
private static String fileName;
private static BufferedReader stdin;
private static PrintStream os;
private  String DOWNLOAD_PATH = ".\\Client\\";

public static void main(String[] args) throws IOException {
try {
sock = new Socket("localhost", 4444);
stdin = new BufferedReader(new InputStreamReader(System.in));
} catch (Exception e) {
System.err.println("Cannot connect to the server, try again later.");
System.exit(1);
}

os = new PrintStream(sock.getOutputStream());

try {
switch (Integer.parseInt(selectAction())) {
case 1:
os.println("1");
sendFile();
break;
case 2:
os.println("2");
System.out.print("Enter file name: ");
fileName = stdin.readLine();
os.println(fileName);
(new FileClient()).receiveFile(fileName);
break;
}
```

```
} catch (Exception e) {
System.err.println("not valid input");
}



sock.close();
}

public static String selectAction() throws IOException {
System.out.println("1. Upload file.");
System.out.println("2. Download file.");
System.out.print("\nMake selection: ");

return stdin.readLine();
}

public static void sendFile() {
try {
System.out.print("Enter file name: ");
fileName = stdin.readLine();
fileName = fileName.replace("\"","");
File myFile = new File(fileName);
byte[] mybytearray = new byte[(int) myFile.length()];

FileInputStream fis = new FileInputStream(myFile);
BufferedInputStream bis = new BufferedInputStream(fis);
//bis.read(mybytearray, 0, mybytearray.length);

DataInputStream dis = new DataInputStream(bis);
dis.readFully(mybytearray, 0, mybytearray.length);
```

```
OutputStream os = sock.getOutputStream();

//Sending file name and file size to the server
DataOutputStream dos = new DataOutputStream(os);
dos.writeUTF(myFile.getName());
dos.writeLong(mybytearray.length);
dos.write(mybytearray, 0, mybytearray.length);
dos.flush();
System.out.println("File "+fileName+" sent to Server.");
} catch (Exception e) {
System.err.println("File does not exist!");
}
}

public  void receiveFile(String fileName) {
try {
int bytesRead;
InputStream in = sock.getInputStream();

DataInputStream clientData = new DataInputStream(in);
System.out.println(fileName);
fileName = clientData.readUTF();
System.out.println(fileName);

File directory = new File(DOWNLOAD_PATH);
System.out.println(directory.getPath());
if (!directory.isDirectory()){
directory.mkdir();
directory = new File(DOWNLOAD_PATH + fileName);
}
else{
directory = new File(DOWNLOAD_PATH + fileName);
}
OutputStream output = new FileOutputStream((directory));
long size = clientData.readLong();
byte[] buffer = new byte[1024];
while (size > 0 && (bytesRead = clientData.read(buffer, 0, (int)
Math.min(buffer.length, size))) != -1) {
output.write(buffer, 0, bytesRead);
size -= bytesRead;
}

output.close();
in.close();

System.out.println("File "+fileName+" received from Server.");
} catch (IOException ex) {
Logger.getLogger(CLIENTConnection.class.getName()).log(Level.SEVERE, null, ex);
}
}
}
```

8 Lessons Learnt

We learned quite a few lessons from the implementation of this project. We learnt the limitations and strengths of manual socket programming and how the HTTP protocol is the perfect evolution of individual communication sockets. By developing a full-stack application, we also developed a deeper understanding of RESTful APIs, NodeJS and front-end development. By using version control like GitHub, we learned how to collaborate easily between three different people.

9 Conclusion

Overall, the FTP Server is an essential tool in most tech companies and its implementation was invaluable for us at this stage. Furthermore, file transfer over sockets is also another essential implementation within socket programming that was very easily digestable through this project's implementation.