Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 6.1

по курсу «Компьютерные системы и сети»

«Разработка дашборда для FTP-клиента»

Студент группы ИУ9-32Б Федуков А. А.

Преподаватель Посевин Д. П.

26 ноября 2024 г.

Цель работы

Целью данной работы является разработка веб консоли для FTP-клиента.

Задание

1. Форма ввода реквизитов доступа:

- ftp-сервер;
- ftp-логин;
- ftp-пароль.

2. Консоль ftp команд, передача команд выполняется асинхронно.
3. Обновление состояния текущей дирректории асинхронно.
4. Визувально выводить отличие лирректории от файла.
5. Выводить пермишины на дирректорию или файл.
6. Фронтенд и бэкэнд на разных серверах запускается.
7. Проверяет преподаватель самостоятельно заходя на URL фронтенда.
8. Тестирование проводится на

- ftp-host: students.yss.su
- login: ftpiu8
- passwd: 3Ru7yOTA

Реализация

Я изменил код FTP-клиента и запустил сайт, который удаленно управлял FTP-клиентом.

Код FTP-клиента

Листинг 1: Файл main.go

```
1  package main
2
3  var ftpCommandChan chan string
4  var wsMessageChan chan Message
5
6  var ftpCurrentDir = "/"
7  var ftpServer string    // students.yss.su:21
8  var ftpLogin string     // ftpiu8
9  var ftpPassword string // 3Ru7yOTA
10
11 var myIP = "localhost"
12 var wsPort = ":1429"
13
14 func main() {
15   myIP = getip2()
16   go startSocket()
17
18   // ftpServer = "students.yss.su:21"
19   // ftpLogin = "ftpiu8"
20   // ftpPassword = "3Ru7yOTA"
21   // client, err := connectFTP()
22   // fmt.Println(err)
23   // go runFTP(client)
24   // ftpCommandChan <- "ls"
25   select {}
26 }
```

Листинг 2: Файл ftp.go

```
1  package main
2
3  import (
4    "bytes"
5    "fmt"
6    "os"
7    "path/filepath"
8    "strings"
9    "text/tabwriter"
10
11   "github.com/secsy/goftp"
12 )
13
14 func sendMsg(msg string) {
15   fmt.Print(msg)
16   wsMessageChan <- Message{Type: "display", Body: msg}
17 }
```

```
18
19 func sendMsgln(msg string) {
20   msg += "\n"
21   sendMsg(msg)
22 }
23
24 func updatePrompt() {
25   prompt := fmt.Sprintf("%s@%s:[%s]", ftpLogin, ftpServer, ftpCurrentDir
       )
26   wsMessageChan <- Message{Type: "setPrompt", Body: prompt}
27 }
28
29 func connectFTP() (*goftp.Client, error) {
30   ftpCommandChan = make(chan string)
31
32   // FTP connection configuration
33   config := goftp.Config{
34     User:     ftpLogin,     // FTP username
35     Password: ftpPassword, // FTP password
36   }
37
38   client, err := goftp.DialConfig(config, ftpServer)
39   return client, err
40 }
41
42 func runFTP(client *goftp.Client) {
43   defer client.Close()
44   updatePrompt()
45   for commandLine := range ftpCommandChan {
46     if commandLine == "Exit" {
47       break
48     }
49
50     input := strings.Fields(commandLine)
51     if len(input) == 0 {
52       continue
53     }
54     command := input[0]
55
56     // Handle each command
57     switch command {
58     case "help":
59       sendMsgln("upload <local_path> <remote_path>\n" +
60         "download <remote_path> <local_path>\n" +
61         "mkdir <remote_path>\n" +
62         "rm <remote_path>\n" +
```

```
63          "ls <remote_path>\n" +
64          "cd <remote_path>\n" +
65          "rmdir <remote_path>\n" +
66          "rmdirall <remote_path>\n" +
67          "exit")
68      case "upload":
69        if len(input) != 3 && len(input) != 2 {
70          sendMsgln("Usage: upload <local_path> <remote_path>")
71          continue
72        }
73        localPath := input[1]
74        remotePath := ""
75        if len(input) == 2 {
76          remotePath = ftpCurrentDir + filepath.Base(input[1])
77        } else {
78          remotePath = resolvePath(input[2])
79        }
80
81        err := uploadFile(client, localPath, remotePath)
82        if err != nil {
83          sendMsg(fmt.Sprintf("Failed to upload file: %v\n", err))
84        } else {
85          sendMsgln("File uploaded successfully." + "\n")
86        }
87
88      case "download":
89        if len(input) != 3 && len(input) != 2 {
90          sendMsgln("Usage: download <remote_path> <local_path>")
91          continue
92        }
93        remotePath := input[1]
94        localPath := ""
95        if len(input) == 2 {
96          localPath = filepath.Base(input[1])
97        } else {
98          localPath = resolvePath(input[2])
99        }
100
101       err := downloadFile(client, remotePath, localPath)
102       if err != nil {
103         sendMsg(fmt.Sprintf("Failed to download file: %v\n", err))
104       } else {
105         sendMsgln("File downloaded successfully.")
106       }
107
108     case "mkdir":
```

```
109         if len(input) != 2 {
110           sendMsgln("Usage: mkdir <remote_path>")
111           continue
112         }
113         remotePath := resolvePath(input[1])
114         _, err := client.Mkdir(remotePath)
115         if err != nil {
116           sendMsg(fmt.Sprintf("Failed to create directory: %v\n", err))
117         } else {
118           sendMsgln("Directory created successfully.")
119         }
120
121       case "rm":
122         if len(input) != 2 && len(input) != 1 {
123           sendMsgln("Usage: rm <remote_path>")
124           continue
125         }
126
127         remotePath := ""
128         if len(input) == 1 {
129           remotePath = resolvePath("")
130         } else {
131           remotePath = resolvePath(input[1])
132         }
133
134         err := client.Delete(remotePath)
135         if err != nil {
136           sendMsg(fmt.Sprintf("Failed to delete file: %v\n", err))
137         } else {
138           sendMsgln("File deleted successfully.")
139         }
140
141       case "ls":
142         if len(input) != 2 && len(input) != 1 {
143           sendMsgln("Usage: ls <remote_path>")
144           continue
145         }
146
147         remotePath := ""
148         if len(input) == 1 {
149           remotePath = resolvePath("")
150         } else {
151           remotePath = resolvePath(input[1])
152         }
153
154         entries, err := client.ReadDir(remotePath)
```

```
155         if err != nil {
156           sendMsg(fmt.Sprintf("Failed to list directory: %v\n", err))
157         } else {
158           out := "Directory contents:\n"
159           tab := [][]string{}
160 
161           tab = append(tab, []string{"[Count]", "[Type]", "[Name]", "[Size
    ]"})
162           for i, entry := range entries {
163             row := []string{}
164             row = append(row, fmt.Sprint(i)+")")
165             etype := ""
166             if entry.IsDir() {
167               etype = "d"
168             } else {
169               etype = "-"
170             }
171             row = append(row, etype)
172             row = append(row, entry.Name())
173             row = append(row, fmt.Sprint(entry.Size()))
174             tab = append(tab, row)
175           }
176           out += formatTable(tab)
177           sendMsg(out)
178         }
179 
180       case "cd":
181         if len(input) != 2 {
182           sendMsgln("Usage: cd <remote_path>")
183           continue
184         }
185         newPath := input[1]
186         if err := changeDirectory(client, newPath); err != nil {
187           sendMsg(fmt.Sprintf("Failed to change directory: %v\n", err))
188         } else {
189           sendMsg(fmt.Sprintf("Changed directory to: %s\n", ftpCurrentDir)
    )
190         }
191 
192       case "rmdir":
193         if len(input) != 2 && len(input) != 1 {
194           sendMsgln("Usage: rmdir <remote_path>")
195           continue
196         }
197 
198         remotePath := ""
```

```
199         if len(input) == 1 {
200           remotePath = resolvePath("")
201         } else {
202           remotePath = resolvePath(input[1])
203         }
204 
205         err := client.Rmdir(remotePath)
206         if err != nil {
207           sendMsg(fmt.Sprintf("Failed to remove directory: %v\n", err))
208         } else {
209           sendMsgln("Directory removed successfully.")
210         }
211 
212       case "rmdirall":
213         if len(input) != 2 && len(input) != 1 {
214           sendMsgln("Usage: rmdirall <remote_path>")
215           continue
216         }
217 
218         remotePath := ""
219         if len(input) == 1 {
220           remotePath = resolvePath("")
221         } else {
222           remotePath = resolvePath(input[1])
223         }
224 
225         err := removeDirectoryRecursive(client, remotePath)
226         if err != nil {
227           sendMsg(fmt.Sprintf("Failed to recursively delete directory: %v\
      n", err))
228         } else {
229           sendMsgln("Directory recursively deleted successfully.")
230         }
231 
232       case "exit":
233         sendMsgln("Exiting...")
234         wsMessageChan <- Message{Type: "loginError", Body: "Error"}
235         return
236 
237       default:
238         sendMsgln("Unknown command.")
239       }
240       updatePrompt()
241 
242     }
243 }
```

```
244
245 func formatTable(data [][]string) string {
246   if len(data) == 0 {
247     return ""
248   }
249
250   // Calculate the maximum width for each column
251   colWidths := calculateColumnWidths(data)
252
253   // Use a bytes.Buffer to build the formatted string
254   var buf bytes.Buffer
255   writer := tabwriter.NewWriter(&buf, 0, 0, 1, '_', 0)
256
257   // Build each row
258   for _, row := range data {
259     for i, col := range row {
260       // Pad each column to its calculated width
261       formattedCol := fmt.Sprintf("%-*s", colWidths[i], col)
262       // Replace spaces with underscores
263       writer.Write([]byte(strings.ReplaceAll(formattedCol, " ", "_")))
264       // Add a tab to separate columns
265       writer.Write([]byte("\t"))
266     }
267     writer.Write([]byte("\n")) // End the row
268   }
269
270   writer.Flush()
271   return buf.String()
272 }
273
274 // calculateColumnWidths determines the maximum width for each column in
      the table
275 func calculateColumnWidths(data [][]string) []int {
276   // Find the number of columns
277   colCount := len(data[0])
278   colWidths := make([]int, colCount)
279
280   // Iterate over the data to calculate the max width for each column
281   for _, row := range data {
282     for i, col := range row {
283       if len(col) > colWidths[i] {
284         colWidths[i] = len(col)
285       }
286     }
287   }
288
```

```
289     return colWidths
290 }
291
292 // Change directory function that updates globalRemotePath
293 func changeDirectory(client *goftp.Client, newPath string) error {
294     // Resolve the path relative to current directory
295     newFullPath := resolvePath(newPath)
296
297     // Check if the directory exists on the server by attempting to read
        it
298     _, err := client.ReadDir(newFullPath)
299     if err != nil {
300         return fmt.Errorf("directory does not exist or cannot be accessed")
301     }
302
303     // Update the global path
304     ftpCurrentDir = newFullPath
305
306     if ftpCurrentDir[len(ftpCurrentDir)-1] != '/' {
307         ftpCurrentDir += "/"
308     }
309
310     return nil
311 }
312
313 // Resolve path based on the current globalRemotePath
314 func resolvePath(path string) string {
315
316     if path == "" {
317         return ftpCurrentDir
318     }
319
320     if filepath.IsAbs(path) {
321         return path
322     }
323     return filepath.Join(ftpCurrentDir, path)
324 }
325
326 // Other helper functions for FTP operations
327 func uploadFile(client *goftp.Client, localPath, remotePath string)
        error {
328     file, err := os.Open(localPath)
329     if err != nil {
330         return err
331     }
332     defer file.Close()
```

```
333     return client.Store(remotePath, file)
334 }
335
336 func downloadFile(client *goftp.Client, remotePath, localPath string)
        error {
337     file, err := os.Create(localPath)
338     if err != nil {
339         return err
340     }
341     defer file.Close()
342     return client.Retrieve(remotePath, file)
343 }
344
345 func removeDirectoryRecursive(client *goftp.Client, remotePath string)
        error {
346     entries, err := client.ReadDir(remotePath)
347     if err != nil {
348         return err
349     }
350     for _, entry := range entries {
351         fullPath := filepath.Join(remotePath, entry.Name())
352         if entry.IsDir() {
353             err := removeDirectoryRecursive(client, fullPath)
354             if err != nil {
355                 return err
356             }
357
358         } else {
359             err := client.Delete(fullPath)
360             if err != nil {
361                 return err
362             }
363         }
364     }
365     return client.Rmdir(remotePath)
366 }
```

Листинг 3: Файл getIp.go

```
1 package main
2
3 import (
4     "encoding/json"
5     "io/ioutil"
6     "net/http"
7 )
8
```

```
type IP struct {
  Query string
}

func getip2() string {
  req, err := http.Get("http://ip-api.com/json/")
  if err != nil {
    return err.Error()
  }
  defer req.Body.Close()

  body, err := ioutil.ReadAll(req.Body)
  if err != nil {
    return err.Error()
  }

  var ip IP
  json.Unmarshal(body, &ip)

  return ip.Query
}
```

Листинг 4: Файл socket.go

```
package main

import (
  "fmt"
  "log"
  "net/http"
  "strings"
  "sync"

  "github.com/gorilla/websocket"
)

type Message struct {
  Type string `json:"type"`
  Body string `json:"body"`
}

var (
  clients  = make(map[*websocket.Conn]bool) // Connected clients
  upgrader = websocket.Upgrader{
    CheckOrigin: func(r *http.Request) bool {
      return true
    },
```

```
24    }
25    mu sync.Mutex
26  )
27
28  // Initisalize connection
29  func handleConnections(w http.ResponseWriter, r *http.Request) {
30    log.Println("New ws connection")
31    ws, err := upgrader.Upgrade(w, r, nil)
32    if err != nil {
33      log.Printf("Error upgrading connection: %v\n", err)
34      return
35    }
36    defer ws.Close()
37
38    mu.Lock()
39    clients[ws] = true
40    mu.Unlock()
41
42    // Read from socket
43    for {
44      var msg Message
45      err = ws.ReadJSON(&msg)
46      log.Println("New ws message: ", msg)
47      if err != nil {
48        log.Printf("Error reading JSON: %v\n", err)
49        mu.Lock()
50        delete(clients, ws)
51        mu.Unlock()
52        break
53      }
54
55      if msg.Type == "ftpCommand" {
56        go func() {
57          ftpCommandChan <- msg.Body
58        }()
59      } else if msg.Type == "ftpLogin" {
60        log.Println(msg.Body)
61        logindata := strings.Split(msg.Body, " ")
62        ftpServer = ""
63        ftpLogin = ""
64        ftpPassword = ""
65        if len(logindata) == 3 {
66          ftpServer = logindata[0]
67          ftpLogin = logindata[1]
68          ftpPassword = logindata[2]
69        }
```

```
70
71          if client, err := connectFTP(); err != nil {
72            wsMessageChan <- Message{Type: "loginError", Body: "Error"}
73            log.Printf("Failed to login: server|%s|, user|%s|, password|%s|"
      , ftpServer, ftpLogin, ftpPassword)
74          } else {
75            wsMessageChan <- Message{Type: "loginSuccess", Body: "<3"}
76            log.Println("FTP login")
77            go runFTP(client)
78          }
79        } else {
80          fmt.Printf("Unknown ws comman: %s", msg.Type)
81        }
82
83    }
84 }
85
86 func handleMessages() {
87    // Send to socket
88    for {
89      msg := <-wsMessageChan
90
91      mu.Lock()
92      for client := range clients {
93        err := client.WriteJSON(msg)
94        if err != nil {
95          log.Printf("Error writing JSON: %v\n", err)
96          client.Close()
97          delete(clients, client)
98        }
99      }
100     mu.Unlock()
101   }
102 }
103
104 func startSocket() {
105   wsMessageChan = make(chan Message)
106   clients = make(map[*websocket.Conn]bool)
107   http.HandleFunc("/ws", handleConnections)
108
109   go handleMessages()
110
111   fmt.Println("WebSocket server started on " + myIP + wsPort)
112   log.Fatal(http.ListenAndServe(myIP+wsPort, nil))
113 }
```

Код вебсайта

Листинг 5: Файл main.go

```go
package main

import (
  "log"
  "net/http"
)

func main() {
  // Serve static files (CSS, JS, images)
  fs := http.FileServer(http.Dir("./static"))
  http.Handle("/static/", http.StripPrefix("/static/", fs))

  http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
    log.Printf("Requested URL: %s", r.URL.Path)

    http.ServeFile(w, r, "./static/index.html")
  })

  // http.HandleFunc("/work", func(w http.ResponseWriter, r *http.
    Request) {
  //  log.Printf("Requested URL: %s", r.URL.Path)
  //  http.ServeFile(w, r, "./static/console.html")
  // })
  log.Println("JS client started on localhost:1932")
  log.Fatal(http.ListenAndServe(":1932", nil))
}
```

Листинг 6: Файл index.html

```html
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0
    ">
    <title>Console</title>
    <link rel="stylesheet" href="/static/style.css">
</head>

<body>
    <div class="main" id="login">
        <h1>Login to FTP</h1>
        <div class="login-form">
```

```
14              <div class="login-form__field">
15                  <label for="server">Enter server:</label>
16                  <input type="text" id="server" name="server" value="" />
17              </div>
18              <div class="login-form__field">
19                  <label for="username">Enter your login:</label>
20                  <input type="text" id="username" name="username" value="
    " />
21              </div>
22              <div class="login-form__field">
23                  <label for="password">Enter your password:</label>
24                  <input type="password" id="password" name="password"
    value="" />
25              </div>
26              <div class="login-form__field">
27                  <span>Status:</span>
28                  <span id="connectionStatus">Disconnected</span>
29                  <button onclick="sendLogin()" id="submitBtn">Login</
    button>
30              </div>
31          </div>
32      </div>
33
34      <div class="main" id="console">
35          <h1>FTP server console</h1>
36          <div class="output" id="output" tabindex="0">
37              <template id="command-template">
38                  <div class="output__line">
39                      <span class="output__prompt"></span>
40                      <span class="output__commands"></span>
41                      <span class="cursor"> </span>
42                  </div>
43              </template>
44              <template id="output-template">
45                  <div class="output__line">
46                      <span class="output__text">Text</span>
47                  </div>
48              </template>
49          </div>
50
51      </div>
52      <script src="/static/app.js"></script>
53  </body>
54
55  </html>
```

Листинг 7: Файл app.js

```
1
2  const ws = new WebSocket("ws://185.104.251.226:1429/ws");
3  var ws_opened = false
4  ws.onopen = () => {
5      console.log("Connected to server");
6      ws_opened = true
7      document.getElementById("submitBtn").disabled = false
8  };
9
10 ws.onmessage = (event) => {
11     const msg = JSON.parse(event.data);
12     if (msg.type == "display") {
13         displayMessage(msg.body)
14     } else if (msg.type == "setPrompt") {
15         setPrompt(msg.body)
16         addLine()
17
18         // if (document.querySelectorAll(".output__commands").length ==
       0) {
19         //     addLine()
20         // }
21     } else if (msg.type == "loginError") {
22         // setPrompt(msg.body)
23         document.getElementById("connectionStatus").textContent = "Wrong
       login!"
24
25     } else if (msg.type == "loginSuccess") {
26         document.getElementById("connectionStatus").textContent = "
       Connected"
27         document.getElementById("submitBtn").disabled = true
28         document.getElementById("console").style.visibility = "visible"
29     } else {
30         console.error("Unknown command from socket")
31     }
32 };
33
34 ws.onclose = () => {
35     console.log("Disconnected from server");
36     ws_opened = false
37     document.getElementById("submitBtn").disabled = true
38     document.getElementById("connectionStatus").textContent = "Closed"
39     document.getElementById("console").style.visibility = "hidden"
40 };
41
42 function sendLogin() {
```

```
43      const serv = document.getElementById("server").value
44      const user = document.getElementById("username").value
45      const pswd = document.getElementById("password").value
46
47      document.getElementById("connectionStatus").textContent = "Waiting
        ..."
48
49      const message = {
50          type: "ftpLogin",
51          body: [serv, user, pswd].join(" "),
52      };
53      ws.send(JSON.stringify(message));
54  }
55
56  function sendMessage(bodyr) {
57      const message = {
58          type: "ftpCommand",
59          body: bodyr,
60      };
61      ws.send(JSON.stringify(message));
62  }
63
64  function displayMessage(msg, type) {
65      const template = document.querySelector("#output-template");
66      const outputDiv = document.querySelector(".output");
67      if (msg[msg.length - 1] == "\n") {
68          msg = msg.slice(0, -1)
69      }
70      msg.split("\n").forEach(splMsg => {
71
72          const newCommand = template.content.cloneNode(true);
73          newCommand.querySelector(".output__text").textContent = splMsg
74
75          if (type) {
76              newCommand.querySelector(".output__text").classList.add(type
        )
77          }
78
79          outputDiv.appendChild(newCommand);
80      });
81  }
82
83
84  function clearCMD() {
85      // const cmds = document.getElementsByClassName("output__line")
86      const cmds = document.querySelectorAll(".output__line")
```

```
87
88      for (const element of cmds) {
89          element.remove()
90      }
91
92      addLine()
93  }
94
95  function setPrompt(prompt) {
96      const template = document.querySelector("#command-template");
97      template.content.querySelector(".output__prompt").textContent =
        prompt + " "
98  }
99
100 function addLine() {
101     const outputDiv = document.querySelector(".output");
102     const curs = document.querySelector(".cursor")
103     if (curs) {
104         curs.remove()
105     }
106     const template = document.querySelector("#command-template");
107     const newCommand = template.content.cloneNode(true);
108     outputDiv.appendChild(newCommand);
109 }
110
111
112 document.addEventListener("DOMContentLoaded", () => {
113     const outputDiv = document.querySelector(".output");
114     let commands = document.querySelectorAll(".output__commands");
115
116
117
118     // Ensure the `output` div is focusable and listens for key events
119     outputDiv.addEventListener("keydown", (event) => {
120         event.preventDefault();
121
122         // Ensure the main div is focused
123         if (document.activeElement === outputDiv) {
124             commands = document.querySelectorAll(".output__commands");
125             if (commands.length != 0) {
126                 const lastCommand = commands[commands.length - 1];
127                 // Check for special keys
128                 if (event.key === "Backspace") {
129                     lastCommand.textContent = lastCommand.textContent.
        slice(0, -1); // Remove last character
130                 } else if (event.key === "Enter") {
```

```
131                     if (lastCommand.textContent == "clear") {
132                         clearCMD()
133                     } else if (lastCommand.textContent == "") {
134                         addLine()
135                     } else if (["upload", "download"].indexOf(
    lastCommand.textContent.split(" ")[0]) != -1) {
136                         // sendMessage(lastCommand.textContent)
137                         // Make popup
138                     } else if (["ls", "rm", "cd", "mkdir", "exit", "
    rmdirall", "help", "rmdir"].indexOf(lastCommand.textContent.split("
    ")[0]) != -1) {
139                         displayMessage("Waiting...")
140                         sendMessage(lastCommand.textContent)
141                     } else {
142                         displayMessage("Unknown command", "
    output__text_error")
143                         addLine()
144                     }
145 
146                 } else if (event.key.length == 1) {
147                     lastCommand.textContent += event.key; // Append
    typed character
148                 }
149             }
150 
151 
152         }
153     });
154 });
```

Листинг 8: Файл style.css

```
1 .main {
2     display: grid;
3     justify-items: center;
4     width: 80%;
5     margin-left: auto;
6     margin-right: auto;
7 }
8 
9 .login-form {
10     display: block;
11 }
12 
13 .login-form button {
14     /* width: 100%; */
15     width: 150px;
```

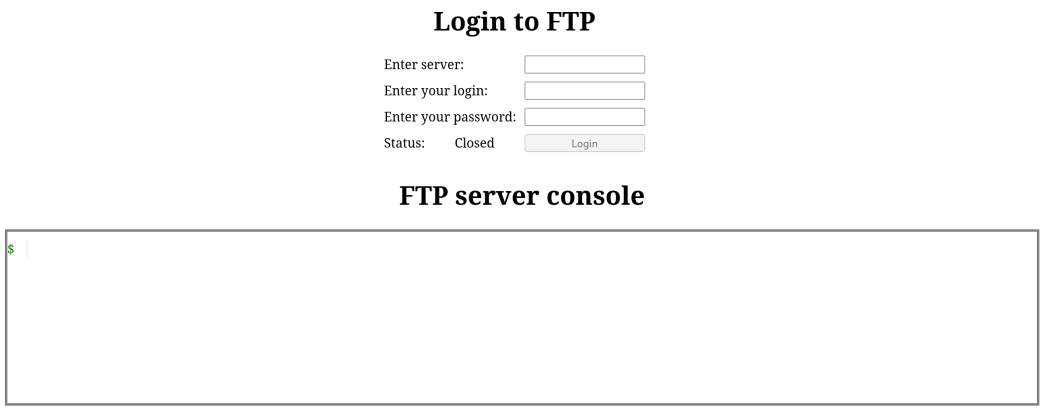
```
16 }
17
18 .login-form__field {
19     margin-bottom: 10px;
20     display: flex;
21     justify-content: space-between;
22 }
23
24 .login-form__field input {
25     margin-left: 10px;
26     min-width: 150px;
27     max-width: 150px;
28     /* max-width: max-content; */
29     /* align-items: right; */
30     /* margin-left: auto; */
31     /* border: 1px solid red; */
32 }
33
34 #console {
35     visibility: hidden;
36 }
37
38 hr {
39     width: 100%;
40
41 }
42
43 .output {
44     border: 3px solid gray;
45     outline: none;
46     padding-top: 10px;
47
48     min-width: calc(1vw * 80);
49     max-width: 50%;
50     min-height: calc(1vh * 24);
51 }
52
53 .output:focus {
54     border: 3px solid darkblue;
55 }
56
57 .output__line {
58     margin: 0;
59     margin-left: 5%;
60     margin-top: 5px;
61     text-align: left;
```

```
62      font-size: 20px;
63      text-decoration: none;
64      display: block;
65  }
66
67  .output__prompt {
68      color: green;
69      text-decoration: none;
70  }
71
72  .output__prompt::after {
73      content: "$";
74      text-decoration: none;
75  }
76
77  .output__line:last-of-type .output__prompt::after {
78      text-decoration: underline;
79  }
80
81  .output__commands {
82      margin-left: 10px;
83  }
84
85  .output__text {
86      margin-left: 5px;
87      font-size: 15px;
88      color: darkslategray;
89  }
90
91  .output__text_error {
92      color: darkred;
93  }
94
95  .cursor {
96      animation-name: blink;
97      animation-duration: 500ms;
98      animation-iteration-count: infinite;
99      /* animation-timing-function: linear; */
100     animation-direction: alternate;
101     background-color: gray;
102     opacity: 1;
103
104 }
105
106
107 @keyframes blink {
```

```
108     from {
109         opacity: 1;
110     }
111
112     to {
113         opacity: 0;
114     }
115 }
```

Вывод программы

После открытия вебсайта происходит соединение по WebSocket. Таким образом, веб интерфейс обращается к серверу, который в свою очередь выполняет команды при помощи FTP



Вывод

В этот лабораторной я научился переписывать старый код, потому что к нему иначе нельзя добавить новый функционал. Помимо этого, я вспомнил WebSocket, а также CSS.