Documentație WebServer VVS

Cotișel Radu - Bogdan

Decembrie 2020

Cuprins

Capitolul 1

Introducere

Aplicația implementată de mine reprezintă un WebServer descris în Java, care deservește requesturi pagini HTML deținute de user în directoarele locale, și în cazul în care acestea sunt găsite și serverul este configurat corect, el le afișează pe socketul pe care are o conexiune activă.

Aplicația dispune de o interfață grafică utilizată pentru configurarea serverului, totodată poate rula în mod standalone, dar fără posibilități de configurare similare.

Capitolul 2

Implementare

2.1 Implementarea Serverului

Serverul Web este implementat în clasa de mai jos, această clasă ocupându-se cu pornirea/oprirea serverului și acceptarea conexiunilor pe socketul dorit, precum și cu trecerea serverului în mentenanță.

```
public class WebServer extends Thread {
    //instanta serverului expusa spre exterior
    private static WebServer instance;

    //socketul pe care se trimite requestul si pe care se face afisarea paginilor
    protected Socket clientSocket;

    //numarul de port dorit
    private int portNumber;

    //socketul care accepta conexiuni la pornirea serverului
    private ServerSocket serverSocket;

    //acest camp ne expune starea actuala a serverului spre exterior
    private String serverState;

    //Avem doi constructori in clasa WebServer, acestia deservind pentru crearea unei instante
         de server atunci cand se doreste asta si totodata constructorul parametrizat este
        folosit pentru acceptarea conexiunilor pe socketul dat ca parametru.

    private WebServer(Socket clientSoc) {
        clientSocket = clientSoc;
    }

    private WebServer() {
        serverState = "Stopped";
    }

    //Aceasta clasa reprezinta hibridul unui Singleton, deoarece inspre exterior ea expune
        doar o instanta unica, insa in interiorul ei, pentru acceptarea conexiunilor, mai
        multe instante sunt create, fiecare ruland pe un Thread separat. Instanta expusa spre
        exterior este folosita pentru schimbarea starii serverului, pornirea, oprirea sau
        trecerea lui in mentenanta.

    public static WebServer getInstance() {
        if (instance == null) instance = new WebServer();

        return instance;
    }

    //Urmatoarele metode se ocupa de tranzitiile serverului dintr-o stare in alta

    //Metoda startServer() porneste un ciclu infinit care porneste Thread-uri care accepta
        conexiuni pe socketul corespunzator portului dorit.
```

```
public void startServer() {
    serverState = "Running";

    try {
        serverSocket = new ServerSocket(portNumber);
        System.out.println("Connection Socket Created");
        try {
            while (true) {
                System.out.println("Waiting for Connection");
                new WebServer(serverSocket.accept()).start();
            }
        } catch (SocketException e) {
            if (serverSocket.isClosed())
                System.out.println("Connection Closed.");
        } catch (IOException e) {
            System.err.println("Accept failed.");
            System.exit(1);
        }
    } catch (IOException e) {
        System.err.println("Could not listen on port: 10008.");
        System.exit(1);
    } finally {
        try {
            if (serverSocket != null) {
                serverSocket.close();
            }
        } catch (IOException e) {
            System.err.println("Could not close port: 10008.");
            System.exit(1);
        }
    }
}

//Metoda stopServer() apeleaza close pe socketul initializat de metoda anterioara. Aceasta
     va face metoda accept de pe socket sa arunce o exceptie, astfel incheind ciclul
    infinit in care ruleaza si oprind serverul.
public void stopServer() throws IOException {
    serverState = "Stopped";
    serverSocket.close();
}

//Cele doua metode de mai jos trec serverul din 'Running' in 'Maintenance' si viceversa.
public void startServerMaintenance() {
    serverState = "Maintenance";
}

public void endServerMaintenance() {
    serverState = "Running";
}

//Aceasta metoda reprezinta override-ul methodei run() din clasa de baza Thread extinsa de
     WebServer.
public void run() {
    //Anuntam existenta unei noi conexiuni
    System.out.println("New Communication Thread Started");

    try {
        BufferedReader in = new BufferedReader(new InputStreamReader(
                clientSocket.getInputStream()));

        //Citim prima linie din InputStreamul socketului client, linie care contine
            requestul si il deservim prin clasa HTMLRenderer
        HtmlRenderer.getInstance().renderHtmlPage(in.readLine(), clientSocket.
            getOutputStream());

        //Afisam in consola datele conexiunii
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            System.out.println("Server: " + inputLine);

            if (inputLine.trim().equals(""))
                break;
        }
```

```
            in.close();
            clientSocket.close();
        } catch (IOException e) {
            System.err.println("Problem with Communication Server");
            System.exit(1);
        }
    }

    //Getters and Setters
}
```

Secvența de cod 2.1: Web Server

2.2 Afișarea paginilor HTML

De afișarea paginilor HTML pe server se ocupă clasa HtmlRenderer. Aceasta primește requestul trimis către server, citește adresa pagini care se dorește a fi accesată și, în cazul în care aceasta există, o afișează, în caz contrar, afișează '404 File not found'.

```
public class HtmlRenderer {
    //Fields

    //Aceasta clasa este singleton
    public static HtmlRenderer getInstance() {
        if (instance == null) instance = new HtmlRenderer();

        return instance;
    }

    //Metoda aceasta se ocupa de afisarea paginilor pe server
    public String renderHtmlPage(String request, OutputStream clientOutputStream) {
        if (request != null) {
            //se printeaza pe output stream-ul socketului serverului faptul ca urmeaza sa se
                afiseze continut de tip html
            PrintWriter out = new PrintWriter(clientOutputStream, true);
            out.println("HTTP/1.1 200 OK");
            out.println("Content-Type: text/html");
            out.println("\r\n");

            //in cazul in care starea serverului este cea de mentenanta se va afisa mereu
                pagina care anunta userul de aceasta stare
            if (WebServer.getInstance().getServerState().equals("Maintenance")) {
                String htmlFilePath = maintenancePageLocation + "/index.html";

                try {
                    BufferedReader reader = new BufferedReader(new FileReader(htmlFilePath));
                    String inputLine;
                    while ((inputLine = reader.readLine()) != null)
                        out.println(inputLine);

                    reader.close();
                } catch (Exception e) {
                    out.println("<h1>404 File not found</h1>");
                }

                out.close();

                return "maintenance";
            } else {
                //daca serverul nu este in mentenanta se extrage adresa fisierului cerut
                String[] requestParameters = request.split(" ");
                String htmlFilePath = requestParameters[1].equals("/") ? "/index.html" :
                    requestParameters[1];

                //daca acest fisier este gasit se va afisa continutul sau
```

```
                try {
                    BufferedReader reader = new BufferedReader(new FileReader(rootPageLocation
                        + htmlFilePath));
                    String inputLine;
                    while ((inputLine = reader.readLine()) != null)
                        out.println(inputLine);

                    reader.close();
                } catch (Exception e) {
                    //daca nu este gasit se va afisa eroarea in continutul paginii
                    out.println("<h1>404 File not found</h1>");
                    out.close();
                    return "404";
                }

                out.close();

                //returnam calea catre fisier pentru o testare mai usoara in UnitTesting
                return htmlFilePath;
            }
        }

        return null;
    }

    //Setters
}
```

Secvența de cod 2.2: HTML Renderer

2.3 Salvarea Setărilor

Ultimele setări cu care a rulat serverul sunt salvate la închiderea acestuia prin intermediul clasei Settings, care scrie aceste setări într-un fisier serializat și le citește la startup.

```
//Clasa Settings implementeaza interfata serializable
public class Settings implements Serializable {
    protected String rootDirectory;
    protected String maintenanceDirectory;
    protected int portNumber;
    private static final String filename = "settings.ser";

    public Settings(String rootDirectory, String maintenanceDirectory, int portNumber) {
        this.rootDirectory = rootDirectory;
        this.maintenanceDirectory = maintenanceDirectory;
        this.portNumber = portNumber;
    }

    //Metoda de mai jos salveaza setarile actuale in fisierul 'settings.ser'
    public void serializeSettings() {
        try {
            FileOutputStream file = new FileOutputStream(filename);
            ObjectOutputStream out = new ObjectOutputStream(file);

            out.writeObject(this);

            out.close();
            file.close();
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }

    //Metoda de mai jos citeste uiltimele setari salvate in fisierul 'settings.ser' si creeaza
        si returneaza o instanta a unui obiect Settings populat cu acele setari.
    public static Settings deserializeSettings() {
```

```
        try {
            FileInputStream file = new FileInputStream(filename);
            ObjectInputStream in = new ObjectInputStream(file);

            Settings settings = (Settings) in.readObject();
            in.close();
            file.close();

            return settings;
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }

        return null;
    }
}
```

Secvența de cod 2.3: Settings

Capitolul 3

Interfața grafică

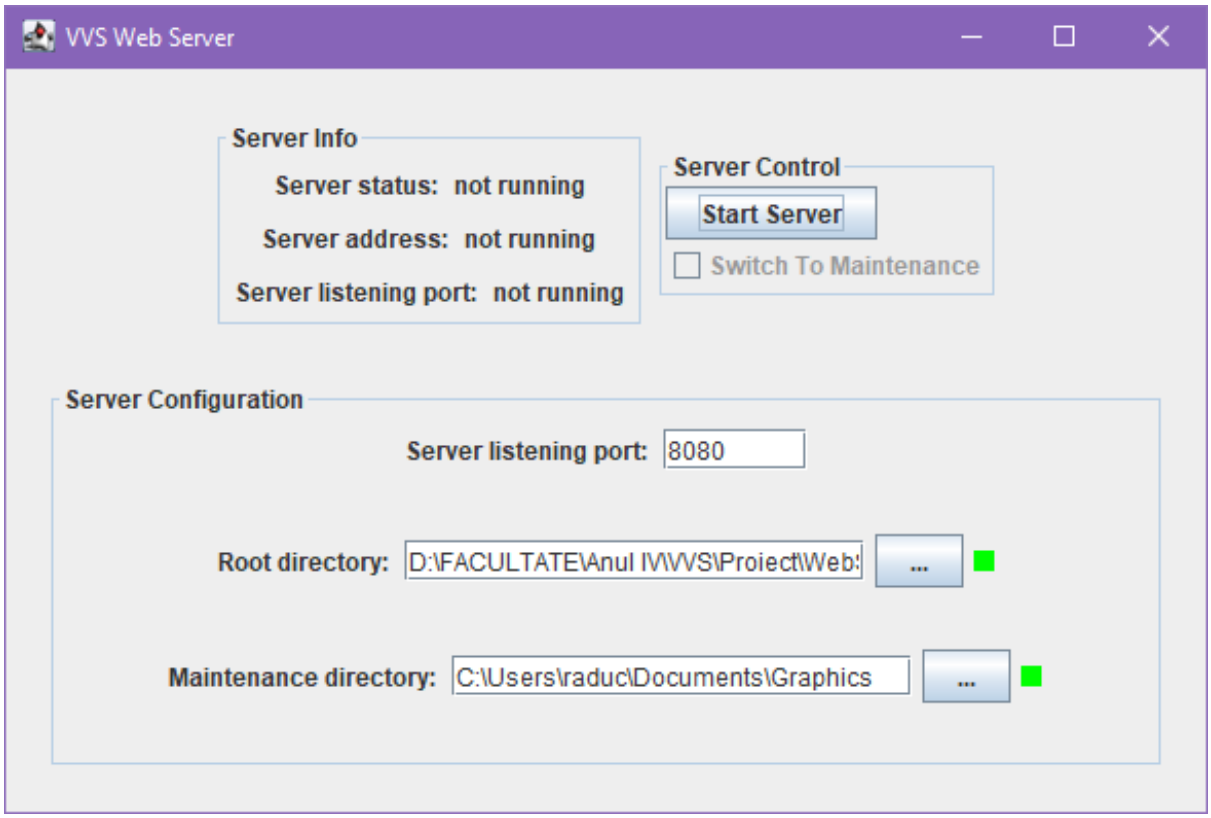

Figura 3.1: Captură de ecran cu interfața grafică a aplicației

3.1 Porțiunea Server Info

Porțiunea server info din interfață va arăta starea actuală a serverului, precum și portul și adresa pe care acesta rulează, dacă este cazul. Cele trei etichete se vor schimba dinamic în funcție de stările prin care userul va trece serverul.

3.2 Porțiunea Server Configuration

Porțiunea de server configuration este porțiunea în care se vor stabili înainte de pornirea serverului portul pe care va rula acesta și directorul de bază în care va căuta fișierele html.

În cazul în care se dorește trecerea serverului în mentenanță, înainte de aceasta, userul va putea stabili, similar directorului de bază, și directorul unde se află pagina html de mentenanță care urmează să fie afișată indiferent de request.

În dreptul celor două adrese de directoare se află un pătrat colorat care va fi verde pentru cazul în care calea către un director este corectă și roșie în caz contrar.

3.3 Porțiunea Server Control

Porțiunea de server control se ocupă de trecerea serverului dintr-o stare în alta. Se ocupă de pornirea serverului, trecerea lui în mentenanță sau de oprirea lui. Aceste schimbări vor fi vizibile și în zona de info.