Alat za transformaciju računalno generiranih alarma u statičku web stranicu

Spajić, Dominik

Sveučilište Josipa Jurja Strossmayera u Osijeku

Fakultet primijenjene matematike i informatike

Sveučilišni prijediplomski studij Matematika i računarstvo

Alat za transformaciju računalno generiranih alarma u statičku web stranicu

Završni rad

Mentor:

doc. dr. sc. Domagoj Ševerdija

Student:

Dominik Spajić

Osijek, 2024.

Sadržaj

1 | Uvod

U današnjem digitalnom dobu, nadzor i upravljanje informacijskim sustavima igraju ključnu ulogu u održavanju pouzdanosti i sigurnosti organizacijskih operacija. S razvojem računalnih tehnologija, sve je veći broj sustava koji automatski generiraju alarme na temelju raznih događaja i stanja u sustavu. Ti alarmi mogu ukazivati na različite probleme, od manjih grešaka do ozbiljnih incidenata koji zahtijevaju hitnu intervenciju.

Međutim, jednostavno generiranje alarma nije dovoljno; ključno je osigurati da ti alarmi budu dostupni i lako čitljivi onima koji su odgovorni za održavanje i upravljanje sustavima. U tom kontekstu, web aplikacije se ističu kao idealno rješenje za prikaz i upravljanje alarmima, pružajući korisnicima pristup alarmnim informacijama u stvarnom vremenu, neovisno o njihovoj fizičkoj lokaciji.

Tema ovog završnog rada je razvoj alata za transformaciju računalno generiranih alarma u interaktivnu web stranicu. Cilj ovog rada je istražiti i implementirati rješenje koje omogućuje praćenje promjena u datotekama koje sadrže alarme i automatsko prikazivanje tih alarma na web stranici. U radu će se koristiti suvremene web tehnologije kao što su HTML, CSS i JavaScript za izradu korisničkog sučelja, dok će poslužitelj biti implementiran u Pythonu korištenjem Flask okvira, Watchdog biblioteke i Socket.IO za osiguranje dvosmjerne komunikacije u stvarnom vremenu.

Kroz ovaj rad, detaljno će se opisati proces izrade ovog alata, od inicijalne ideje i planiranja do tehničke implementacije i testiranja. Također, istražit će se prednosti i izazovi povezani s ovakvim pristupom te će se predložiti moguća poboljšanja za budući razvoj sličnih sustava. Na taj način, ovaj rad ne samo da pruža tehničku dokumentaciju o specifičnom rješenju, već također služi kao primjer kako se modernim tehnologijama može unaprijediti učinkovitost upravljanja informacijskim sustavima.

2 | Web razvoj

2.1 Web razvoj

Web razvoj se odnosi na proces stvaranja web stranica i web aplikacija koje se izvršavaju na web preglednicima. Od svojih početaka u 1990-ima, web razvoj se značajno razvijao, prateći rastuće potrebe za bogatijim i interaktivnijim korisničkim iskustvima. Prvi web preglednici bili su ograničeni samo na prikazivanje jednostavnih HTML stranica. Danas, web razvoj obuhvaća složene aplikacije koje mogu uključivati obradu podataka u stvarnom vremenu, bogatu grafiku, i integraciju s različitim platformama i servisima.
Tijekom godina, web razvoj je prolazio kroz nekoliko faza, počevši od statičnih stranica s jednostavnim tekstom, pa do dinamičkih aplikacija koje omogućuju interakciju s korisnicima putem formulara, baza podataka i API-ja. Svaka faza u evoluciji web razvoja donijela je nove alate i tehnologije koje su olakšale razvoj složenih i skalabilnih web aplikacija (vidjeti [1]).

2.2 Osnovne web tehnologije

2.2.1 Razvoj klijenta

Klijentski razvoj aplikacije obuhvaća sve aspekte koje korisnik izravno vidi i s kojima interaktivno komunicira na web stranici. To uključuje HTML (HyperText Markup Language), CSS (Cascading Style Sheets) i JavaScript (vidjeti [1]).

- **HTML** je temeljni jezik za izradu web stranica koji definira strukturu i sadržaj stranice. Elementi HTML-a organiziraju sadržaj u odlomke, naslove, tablice, forme i druge osnovne komponente stranice.
- **CSS** omogućuje stiliziranje HTML elemenata, omogućujući razvojnim programerima da prilagode izgled i dojam web stranice. Kroz CSS se definiraju boje, fontovi, margine, razmaci, te responzivnost stranice na različitim uređajima.
- **JavaScript** je skriptni jezik koji omogućuje dodavanje interaktivnosti na web stranice. Kroz JavaScript, elementi stranice mogu reagirati na korisničke akcije, kao što su klikovi mišem, unos podataka ili pomicanje po stranici. Osim toga, JavaScript se koristi za dinamičko ažuriranje sadržaja stranice bez potrebe za ponovnim učitavanjem cijele stranice. (vidjeti [2]).

Kombinacija ovih tehnologija omogućuje izradu modernih web sučelja koja su responzivna, intuitivna i prilagođena korisničkom iskustvu.

2.2.2 Razvoj poslužitelja

Poslužiteljski razvoj odnosi se na dio sustava koji radi iza kulisa, upravljajući logikom aplikacije, bazama podataka i serverima. Dok klijent obrađuje sve što korisnik vidi, poslužitelj osigurava da aplikacija pravilno funkcionira i da su svi podaci sigurno pohranjeni i dostupni kada su potrebni (vidjeti [1]).

- **Server-side skriptni jezici**, kao što su Python, PHP, Ruby, i Node.js, omogućuju upravljanje aplikacijskom logikom, autentifikacijom korisnika, obradom podataka i komunikacijom s bazama podataka.
- **Baze podataka**, kao što su MySQL, PostgreSQL, MongoDB, i druge, koriste se za pohranu i upravljanje podacima koje aplikacija koristi i generira. Baze podataka čine srž mnogih web aplikacija, omogućujući spremanje korisničkih podataka, sadržaja, transakcija i drugih ključnih informacija.
- **API (Application Programming Interface)** služi kao most između različitih dijelova aplikacije, omogućujući komunikaciju između klijenta i poslužitelja te između aplikacije i vanjskih usluga. API-jevi su ključni za integraciju aplikacija s vanjskim sustavima i servisima, omogućujući razmjenu podataka i funkcionalnosti na siguran i standardiziran način (vidjeti [4]).

2.2.3 Razvoj punog spektra (Full-Stack Development)

Full-stack development obuhvaća oba aspekta, klijentski i poslužiteljski razvoj. Full-stack developeri su svestrani stručnjaci koji razumiju i rade na svim slojevima web aplikacije, od izrade vizualnog korisničkog sučelja do dizajna i implementacije aplikacijske logike na serveru. Ova vrsta razvoja omogućuje programerima da imaju cjelovito razumijevanje i kontrolu nad cijelim razvojnim procesom, što može biti posebno korisno u manjim timovima ili startupima gdje je potreban širok spektar vještina.

2.2.4 Važnost web razvoja

Web razvoj je postao nezamjenjiv dio modernog poslovanja, obrazovanja, zabave i mnogih drugih sektora. Danas, gotovo svaka organizacija posjeduje web stranicu ili aplikaciju koja omogućuje pristup informacijama, proizvodima i uslugama. Web razvoj omogućuje digitalnu prisutnost organizacija i omogućava korisnicima diljem svijeta pristup raznim uslugama putem interneta.

S obzirom na stalni napredak tehnologije, web razvoj je dinamično područje koje zahtijeva kontinuirano učenje i prilagodbu novim alatima, tehnikama i sigurnosnim standardima. Razvojne platforme, okviri i biblioteke stalno se ažuriraju i unapređuju, što omogućuje izradu sve složenijih i interaktivnijih web aplikacija koje zadovoljavaju potrebe korisnika i tržišta.

3 | Opis projekta

Ovaj projekt ima za cilj izraditi alat za transformaciju računalno generiranih alarma u interaktivnu web stranicu. Sustav je dizajniran tako da automatski prati promjene u datotekama s alarmima i prikazuje ih na web stranici u stvarnom vremenu. Projekt se sastoji od klijenta, koji korisnicima omogućuje pregled i upravljanje alarmima putem web sučelja, te poslužitelja, koji upravlja logikom aplikacije i sinkronizacijom podataka. U nastavku su detaljno opisani ključni aspekti implementacije.
Potpuna implementacija je dostupna na repozitoriju: Moj GitHub repozitorij

3.1 Opća funkcionalnost sustava

Glavna funkcionalnost sustava je automatsko praćenje promjena u datotekama s alarmima, generiranje odgovarajućih prikaza na web stranici i omogućavanje korisnicima da pregledavaju, pretražuju i filtriraju alarme prema različitim kriterijima. Sustav koristi Python za implementaciju poslužitelja, dok su HTML, CSS i JavaScript korišteni za klijentski dio aplikacije.

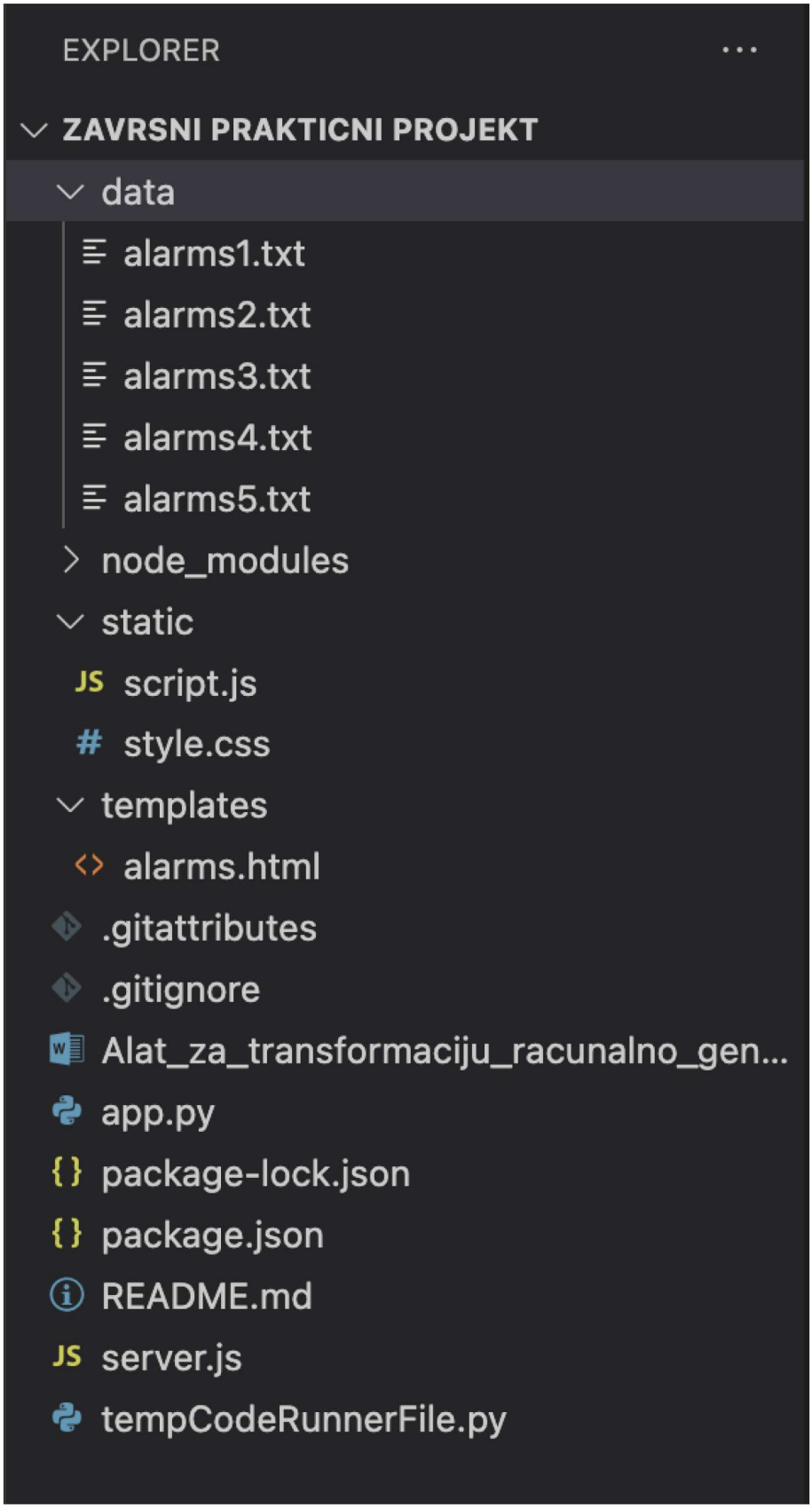
3.2 Implementacija poslužiteljske strane aplikacije

Poslužitelj sustava implementiran je u Pythonu koristeći Flask okvir. Flask omogućava jednostavno postavljanje web servera, dok Watchdog biblioteka prati promjene u direktoriju s alarmima. Socket.IO je korišten za dvosmjernu komunikaciju između servera i klijenta, što omogućuje ažuriranje podataka na web stranici u stvarnom vremenu.

3.2.1 Struktura poslužitelja

Poslužitelj je podijeljen na nekoliko ključnih komponenti:

- Flask server koji služi za upravljanje HTTP zahtjevima i posluživanje klijenta.
- Watchdog biblioteka koja kontinuirano prati promjene u direktoriju s alarmima.
- Socket.IO za dvosmjernu komunikaciju između poslužitelja i klijenta.



Slika 3.1: Arhitektura sustava za praćenje i prikaz alarma

3.2.2 Učitavanje i ažuriranje podataka o alarmima

Funkcije `load_alarms` i `update_alarms_data` ključne su za učitavanje i ažuriranje podataka o alarmima iz tekstualnih datoteka u stvarnom vremenu. Datoteke koje sadrže alarme, poput `alarms1.txt`, `alarms2.txt`, itd., sadrže informacije o alarmima u tekstualnom formatu.

Format podataka o alarmima

Alarmi dolaze u tekstualnom obliku, pri čemu su različite kategorije alarma odvojene specifičnim separatorima, kao što su zarezi (‘,‘) ili točke-zarezi (‘;‘). Svaki alarm sadrži nekoliko informacija, uključujući ID alarma, opis alarma, razinu ozbiljnosti, te vremensku oznaku. Evo primjera kako alarm može izgledati u datoteci:

```
1, "High Temperature", Critical, 2024-08-21 12:34:56
2, "Low Pressure", Warning, 2024-08-21 12:36:12
```

Parsiranje tekstualnih datoteka

Funkcija `load_alarms` učitava sadržaj ovih datoteka, razdvaja podatke prema separatorima, i pohranjuje ih u strukturu podataka koju program koristi za daljnju obradu. Funkcija `update_alarms_data` se koristi za ažuriranje postojećih podataka kada dođe do promjena u datotekama. Ove funkcije osiguravaju da svi podaci o alarmima budu uvijek ažurirani i dostupni na web stranici.

```
1   def load_alarms(file_path):
2       alarms = []
3       with open(file_path, 'r') as file:
4           for line in file:
5               parts = line.strip().split(',')
6               alarm = {
7                   'id': int(parts[0]),
8                   'description': parts[1].strip('"'),
9                   'severity': parts[2],
10                  'timestamp': parts[3]
11              }
12              alarms.append(alarm)
13      return alarms
14
15  def update_alarms_data():
16      alarm_files = ['alarms1.txt', 'alarms2.txt', 'alarms3.txt']
17      all_alarms = []
18      for file in alarm_files:
19          all_alarms.extend(load_alarms(file))
20      return all_alarms
21
```

Listing 3.1: Kod za učitavanje i ažuriranje podataka o alarmima

Objašnjenje
- `load_alarms` funkcija: Učitava sadržaj datoteke s alarmima, razdvaja podatke prema zarezu i pohranjuje ih u listu kao pojedinačne zapise.
- `update_alarms_data` funkcija: Poziva `load_alarms` za svaku datoteku s alarmima, te sve učitane alarme spaja u jednu listu koju vraća.

3.2.3 Real-time ažuriranje podataka na stranici

Komunikacija između servera i klijenta ostvarena je korištenjem Socket.IO, koji omogućava dvosmjernu komunikaciju u stvarnom vremenu. Kada Watchdog detektira promjenu u direktoriju, novi podaci se šalju klijentu pomoću Socket.IO-a, čime se automatski ažurira prikaz na web stranici (vidjeti [3]).

```
1   from flask import Flask, render_template
2   from flask_socketio import SocketIO, send
3
4   app = Flask(__name__)
5   socketio = SocketIO(app)
6
7   @app.route('/')
8       def index():
9       return render_template('index.html')
10
11  @socketio.on('message')
12  def handleMessage(msg):
13      print('Message: ' + msg)
14      send(msg, broadcast=True)
15
16  def emit_new_data(data):
17      socketio.emit('update', data)
18
19  if __name__ == '__main__':
20      socketio.run(app, debug=True)
```

Listing 3.2: Kod za emitiranje događaja s novim podacima

Objašnjenje:
- 'index' ruta: Poslužuje glavnu HTML stranicu kada klijent posjeti glavnu URL adresu (vidjeti [5]).
- 'handleMessage' funkcija: Prima poruke od klijenata i prosljeđuje ih svim povezanim korisnicima.

3.2.4 Uloga datoteke `server.js`

Datoteka 'server.js' služi kao middleware komponenta koja upravlja vezom između klijenta i servera, posebno u kontekstu implementacije web sučelja. Ova datoteka omogućuje pokretanje Node.js servera koji može obrađivati zahtjeve, upravljati sesijama korisnika, i poslužiti statičke datoteke. U kontekstu ovog projekta, 'server.js' služi kao posrednik koji povezuje podatke iz poslužiteljskog sustava (koji su implementirani u Pythonu) sa klijentom (vidjeti [2]).

```
const express = require('express');
const http = require('http');
const socketIo = require('socket.io');

const app = express();
const server = http.createServer(app);
const io = socketIo(server);

app.get('/', (req, res) => {
    res.sendFile(__dirname + '/index.html');
});

io.on('connection', (socket) => {
    console.log('a user connected');
    socket.on('disconnect', () => {
        console.log('user disconnected');
    });
});

server.listen(3000, () => {
    console.log('listening on *:3000');
});

```

Listing 3.3: Osnovna struktura datoteke server.js

Objašnjenje:
- Express.js: Koristi se za upravljanje HTTP zahtjevima i poslužuje statičke datoteke.
- Socket.IO: Omogućava dvosmjernu komunikaciju u stvarnom vremenu između servera i klijenata.
- Server slušatelj: Pokreće server i omogućava mu da sluša na portu 3000.

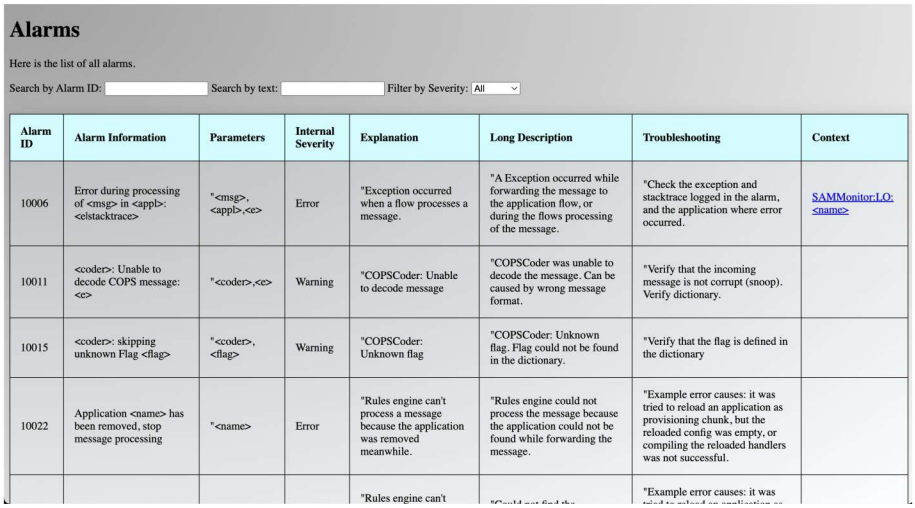
3.3 Implementacija klijentske strane aplikacije

Klijentski dio aplikacije izrađen je korištenjem HTML-a, CSS-a i JavaScript-a. Stranica omogućuje korisnicima pregled i filtriranje alarma putem jednostavnog i intuitivnog sučelja. Prikaz podataka organiziran je u tablicama, a uz pomoć JavaScript-a i CSS-a osigurana je responzivnost i interaktivnost stranice.

3.3.1 Struktura klijenta

Klijent se sastoji od sljedećih komponenti:

- HTML datoteka koja definira strukturu stranice.
- CSS datoteka koja osigurava stilizaciju i izgled stranice.
- JavaScript datoteka koja upravlja dinamičkim aspektima stranice, poput ažuriranja tablice alarma u stvarnom vremenu i omogućavanja filtriranja alarma.



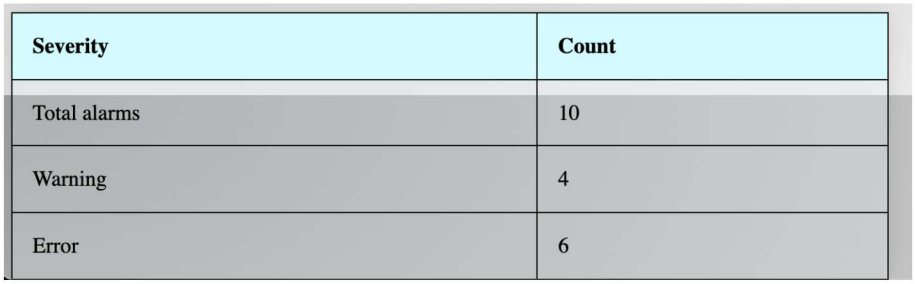
Alarms

Here is the list of all alarms.

Search by Alarm ID: Search by text: Filter by Severity: All

Alarm ID	Alarm Information	Parameters	Internal Severity	Explanation	Long Description	Troubleshooting	Context
10006	Error during processing of <msg> in <appl>: <elstacktrace>	"<msg>, <appl>,<e>	Error	"Exception occurred when a flow processes a message.	"A Exception occurred while forwarding the message to the application flow, or during the flows processing of the message.	"Check the exception and stacktrace logged in the alarm, and the application where error occurred.	SAMMonitor:LO: <name>
10011	<coder>: Unable to decode COPS message: <e>	"<coder>,<e>	Warning	"COPSCoder: Unable to decode message	"COPSCoder was unable to decode the message. Can be caused by wrong message format.	"Verify that the incoming message is not corrupt (snoop). Verify dictionary.	
10015	<coder>: skipping unknown Flag <flag>	"<coder>, <flag>	Warning	"COPSCoder: Unknown flag	"COPSCoder: Unknown flag. Flag could not be found in the dictionary.	"Verify that the flag is defined in the dictionary	
10022	Application <name> has been removed, stop message processing	"<name>	Error	"Rules engine can't process a message because the application was removed meanwhile.	"Rules engine could not process the message because the application could not be found while forwarding the message.	"Example error causes: it was tried to reload an application as provisioning chunk, but the reloaded config was empty, or compiling the reloaded handlers was not successful.	
				"Rules engine can't		"Example error causes: it was	

Slika 3.2: Korisničko sučelje za pregled alarma



Severity	Count
Total alarms	10
Warning	4
Error	6

Slika 3.3: Statistika broja i vrste alarma

3.3.2 HTML i CSS struktura

HTML i CSS definiraju strukturu i izgled web stranice. Primjer osnovne HTML strukture za prikaz alarma prikazan je u nastavku:

```
1     <!DOCTYPE html>
2     <html lang="en">
3     <head>
4         <meta charset="UTF-8">
5         <meta name="viewport" content="width=device-width, initial-
    scale=1.0">
6         <title>Alarm Monitoring</title>
7         <link rel="stylesheet" href="style.css">  <!-- Povezivanje
    CSS datoteke -->
8     </head>
9     <body>
10        <div class="container">
11            <h1>Alarm Monitoring System</h1>
12            <input type="text" id="filterInput" placeholder="Filter
     alarms..."> <!-- Polje za filtriranje -->
13            <table id="alarmsTable">
14                <thead>
```

```
                    <tr>
                        <th>ID</th>
                        <th>Alarm</th>
                        <th>Severity</th>
                        <th>Timestamp</th>
                    </tr>
                </thead>
                <tbody>
                    <!-- Alarmi  -->
                </tbody>
            </table>
        </div>
        <script src="/socket.io/socket.io.js"></script>  <!-- Povezivanje Socket.IO -->
        <script src="script.js"></script>  <!-- Povezivanje JavaScript datoteke -->
    </body>
    </html>

```

Listing 3.4: Osnovna HTML struktura za prikaz alarma

Objašnjenje:
- HTML struktura: Definira izgled i strukturu web stranice s tablicom za prikaz alarma i poljem za filtriranje.
- CSS: Koristi se za stilizaciju i prilagodbu izgleda stranice.

3.3.3 JavaScript za dinamičko ažuriranje i filtriranje podataka

JavaScript se koristi za dinamičko ažuriranje podataka na stranici. Kada server pošalje novi podatak putem Socket.IO-a, JavaScript skripta ažurira tablicu s alarmima na stranici. Također, omogućeno je filtriranje alarma prema unosu korisnika.

```
    const socket = io.connect('http://' + document.domain + ':' + location.port);

    socket.on('update', function(data) {
        const table = document.getElementById('alarmsTable').getElementsByTagName('tbody')[0];
        const newRow = table.insertRow();

        const cell1 = newRow.insertCell(0);
        const cell2 = newRow.insertCell(1);
        const cell3 = newRow.insertCell(2);
        const cell4 = newRow.insertCell(3);

        cell1.innerHTML = data.id;
        cell2.innerHTML = data.alarm;
        cell3.innerHTML = data.severity;
        cell4.innerHTML = data.timestamp;
    });

```

```
18     document.getElementById('filterInput').addEventListener('keyup'
    , function() {
19         const filterValue = this.value.toLowerCase();
20         const rows = document.getElementById('alarmsTable').
    getElementsByTagName('tr');
21 
22         for (let i = 1; i < rows.length; i++) {
23             const cells = rows[i].getElementsByTagName('td');
24             let match = false;
25 
26             for (let j = 0; j < cells.length; j++) {
27                 if (cells[j].innerHTML.toLowerCase().indexOf(
    filterValue) > -1) {
28                     match = true;
29                     break;
30                 }
31             }
32 
33             rows[i].style.display = match ? '' : 'none';
34         }
35     });
36 
```

Listing 3.5: JavaScript za ažuriranje i filtriranje tablice alarma

Objašnjenje:
-Socket.IO povezivanje: Omogućava dvosmjernu komunikaciju između klijenta i servera.
-Ažuriranje tablice alarma: Kada server pošalje novi podatak, tablica se automatski ažurira s novim alarmom.
-Filtriranje alarma: Korisnik može filtrirati prikazane alarme prema unosu u polje za pretragu.

3.4 Testiranje i performanse

Sustav je testiran s različitim skupovima podataka kako bi se osigurala točnost prikaza i performanse sustava. Testovi su pokazali da je sustav sposoban brzo i točno ažurirati podatke na stranici u stvarnom vremenu, čak i s velikim brojem alarma.

- Sustav je testiran s različitim veličinama datoteka i učestalostima promjena kako bi se osiguralo da Watchdog ispravno detektira sve promjene.
- Socket.IO je testiran za brzinu prijenosa podataka između servera i klijenta, a rezultati su pokazali minimalno kašnjenje u prikazu podataka.
- Klijentski dio aplikacije testiran je na različitim uređajima kako bi se osigurala responzivnost i ispravnost prikaza na svim ekranima.

3.5 Zaključak

Ovaj projekt je uspješno implementirao alat za transformaciju računalno generiranih alarma u interaktivnu web stranicu koja omogućuje pregled i upravljanje alarmima u stvarnom vremenu. U budućnosti, projekt se može proširiti dodatnim funkcionalnostima kao što su naprednije filtriranje alarma, integracija s vanjskim sustavima za upravljanje incidentima, te optimizacija performansi za rad s još većim skupovima podataka.

Literatura

[1] J. Robbins, *Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics*, O'Reilly Media, Sebastopol, 2012.

[2] *JavaScript*, dostupno na
`https://developer.mozilla.org/en-US/docs/Web/JavaScript`

[3] *Socket.IO Documentation*, dostupno na
`https://socket.io/docs/v4/`

[4] *What is an API?*, dostupno na
`https://www.postman.com/what-is-an-api/`

[5] *Flask Documentation*, dostupno na
`https://flask.palletsprojects.com/en/latest/`

Sažetak

Ovaj završni rad opisuje razvoj alata za transformaciju računalno generiranih alarma u interaktivnu web stranicu. Korištenjem tehnologija kao što su HTML, CSS, JavaScript i Pythonov Flask okvir, sustav automatski prati promjene u datotekama s alarmima i prikazuje ih u stvarnom vremenu. Implementacija također uključuje Watchdog za praćenje promjena i Socket.IO za dvosmjernu komunikaciju. Rezultati pokazuju da alat učinkovito omogućava praćenje i upravljanje alarmima, čime se poboljšava pouzdanost informacijskih sustava.

Ključne riječi

HTML, CSS, JavaScript, Socket.IO, Flask, Watchdog, web razvoj, alarmi, real-time sustavi, razvoj klijenta, razvoj poslužitelja

Tool for Transforming Computer-Generated Alarms into a Static Web Page

Summary

This thesis describes the development of a tool for transforming computer-generated alarms into an interactive web page. Utilizing technologies such as HTML, CSS, JavaScript, and the Python Flask framework, the system automatically monitors changes in alarm files and displays them in real-time on a web page. The implementation also includes Watchdog for file change monitoring and Socket.IO for bidirectional communication. The results demonstrate that the tool effectively facilitates the tracking and management of alarms, thereby improving the reliability of information systems.

Keywords

HTML, CSS, JavaScript, Socket.IO, Flask, Watchdog, web development, alarms, real-time systems, frontend development, backend development

Životopis

Zovem se Dominik Spajić i rođen sam 30.07.2001. u Osijeku. Obrazovanje sam započeo u Osnovnoj školi "Vladimir Nazor" Čepin, a zatim sam završio III. Gimnaziju u Osijeku 2020. godine. Te iste godine upisujem prijediplomski sveučilišni studij Matematike i računarstva na Fakultetu primijenjene matematike i informatike Sveučilišta J.J. Strossmayera u Osijeku (tada naziva Odjel za matematiku). Tečno govorim engleski jezik, obzirom da sam imao izvrsne ocjene u osnovnoj i srednjoj školi. Od dodatnog obrazovanja završio sam Osnovnu glazbenu školu Franje Kuhača u Osijeku, modul klavir, 2016. godine. Također sam predstavljao grad Osijek na najvišem rangu natjecanja u državi i Europi kroz rukomet i atletiku.