Departamentul Automatică şi Informatică Industrială
Facultatea Automatică şi Calculatoare
Universitatea POLITEHNICA din Bucureşti

AUTOMATICĂ ŞI INFORMATICĂ INDUSTRIALĂ

RAPORT ȘTIINȚIFIC NR. 2

RAPORT DE CERCETARE PENTRU SEMESTRUL II

Eduard-Ștefan POPA

Conducător științific de doctorat: Conf. dr. ing. Mihnea Alexandru MOISESCU

Tema: Stress Testing pentru platforme web

Program de masterat: Managementul și Protecția Informației

Cuprins

1 Rezumat

În cadrul raportului de cercetare pentru primul semestru a fost prezentată o introducere în domeniul testării de tipul Stress Testing, enunțând obiectivele sale principale alături de impactul pe care îl poate avea asupra calității unui produs software. De asemenea, a fost prezentat un studiu de caz dintr-un articol ce aparține Universității Texas din Arlington în care se face referire la structura generală a unui framework de testare automată și tratează un exemplu concret de utilizare.

În continuarea acestei activități se vor prezenta primele contribuții de cercetare aplicativă. În prima etapă se vor preciza primele funcționalități care au fost dezvoltate în cadrul modulului de testare al serverelor ce operează pe protocolul HyperText Transfer Protocol (prescurtat HTTP). De asemenea, fiecare dintre funcționalitățile enumerate sunt însoțite de o descriere sumară.

Mai departe s-a realizat o introducere în cele două framework-uri care au stat la baza implementării acestei aplicații - Angular și, respectiv, Spring Boot. O scurtă argumentare a alegerii acestor instrumente a fost realizată prin prezentarea avantajelor pe care le dețin.

Pentru înțelegerea funcționării aplicației de testare s-a prezentat schema arhitecturală cu descrierea fluxului de date între nivelurile existente. S-a prezentat structura celor două proiecte - unul pentru frontend și celălalt pentru backend - pentru evidențierea caracterului modular al programului software început.

În ultima parte al raportului s-a făcut o paralelă între prezentarea interfeței și a modului de utilizare și interpretare cu detalierea aspectelor ce țin de codul dezvoltat. Mai mult s-a pus accent pe programul dezvoltat pentru implementarea serverului de backend pentru a detalia modul în care se execută testarea după parametri transmiși printr-o cerere HTTP de la client.

2 Proiectarea soluției tehnice

2.1 Stabilirea cerințelor aplicației

În tabelul de mai jos sunt evidențiate funcționalitățile aplicației dezvoltate în cadrul primei etape de cercetare aplicative împreună cu o scurtă descriere a acestora.

Tabelul 1. Prezentare a funcționalităților aplicației de testare

Nr. crt.	Funcționalitate	Descriere funcționalitate
1.	Configurarea aplicației	Conectarea interfeței cu clientul la serverul de backend ce realizează rularea testelor se face printr-un fișier de configurare în format JSON stocat în folderul dedicat interfeței web. Acest fișier conține portul pe care rulează serverul.
2.	Crearea planului de testare pentru servere HTTP	Utilizatorul poate crea un plan de testare în cadrul modului HTTP (singurul creat în cadrul acestei lucrări de cercetare) pe care să îl execute ulterior. Planul de testare presupune specificarea următoarelor atribute: ● numărul de fire de execuție (sau utilizatorii virtuali care vor simula traficul); ● valoarea timpului de "ramp up" care înseamnă perioada de timp dintre trimiterea a două cereri consecutive (pentru a simula un scenariu real de testare); ● numărul de bucle (de câte ori se repetă testul); ● URL-ul către serverul HTTP ce urmează să fie testat; ● portul serverului; ● metoda HTTP din header-ul cererilor; ● calea către resursă; ● adăugarea de parametri la header-ul cererii prin specificarea numelui și a valorii parametrului (acest pas este opțional);
3.	Rularea testelor de Stress Testing conform planului creat de utilizator	Toate valorile menționate în formularul din partea de creare a planului de testare sunt trimise serverului de backend printr-o cerere HTTP cu metoda GET. Rezultatele rulării sunt trimise înapoi către client în format JSON și vor fi parsate pentru a putea fi reprezentate corespunzător.

4.	Afișarea rezultatelor în interfața web	Rezultatele primite de la server sunt reprezentate în diferite moduri: ● un grafic cu evolutia latenței; ● un tabel ce conține detalii precum codul răspunsului cu un mesaj corespunzător codului, numărul firul de execuție care a produs rezultatul și resursa accesată; ● un element de tip textarea care afișează răspunsul primit de le resursă; ● detalii despre execuția testelor: 1) numărul de teste efectuate; 2) rata de răspuns la cereri; 3) media perioadei de răspuns; 4) valoarea minimă a perioadei de răspuns la cerere; 5) valoarea maximă a perioadei de răspuns la cerere;
5.	Vizualizarea istoricului	Planurile de testare și rezultatele acestora sunt stocate la finalul rulării, iar utilizatorul poate vederea conținutul planurilor de testare în format jmx (identic cu formatul xml) direct din interfață fără a fi nevoie rularea unui plan de testare.

2.2 Alegerea tehnologiilor și platformelor de dezvoltare utilizate

Angular

Angular este un framework de dezvoltare a aplicațiilor web de tip open source și care se bazează pe limbajul de programare TypeScript. Acesta a fost creat de echipa Angular din cadrul companiei Google în anul 2016, fiind extrem de utilizat de către dezvoltatorii de software în ultimii ani. Conform unui studiu condus de Stack Overflow în anul 2019 reiese faptul că Angular este al doilea cel mai utilizat framework în rândul programatorilor, în proporție de 32,4% dintre dezvoltatorii web lucrând cu Angular. [1]

Unul dintre principalele motive pentru care Angular este un framework atât de preferat în comunitatea dezvoltatorilor de aplicații web este faptul că arhitectura Angular este orientată pe componente, fapt ce asigură o calitate mai bună a codului scris. Componentele în acest caz pot fi gândite ca o bucată din interfața cu utilizatorul, aspect care face oferă o modularitate crescută a aplicației. De asemenea, această arhitectură de tipul MVC (model - view - controller) realizează o izolare a logicii aplicației față de componentele UI. Astfel, controller-ul primește acțiuni de la aplicație și, conform cu acestea, operează cu o bază de date ce constituie modelul, iar apoi va transmite datele solicitate componentei de view (interfața cu clientul) care le va afișa într-o anumită manieră.

Pe lângă capacitatea de a implementa aplicații web într-un mod eficient, Angular este totodată utilizat în cadrul platformelor mobile, fiind un framework pentru realizarea de soluții software cross-platform. Eficiența acestor soluții provine din unele avantaje pe care Angular le aduce, precum: servicii, injectarea dependențelor și rutarea rapidă. O altă proprietate specifică acestor platforme este cea numită “two-way data binding” care aduce și aceasta un cost de timp mai scăzut. În cadrul

arhitecturii MVC, această proprietate este exemplificată prin faptul că orice schimbare în model aduce imediat o modificare corespunzătoare și în view fără a fi necesară reîncărcarea paginii. [2]

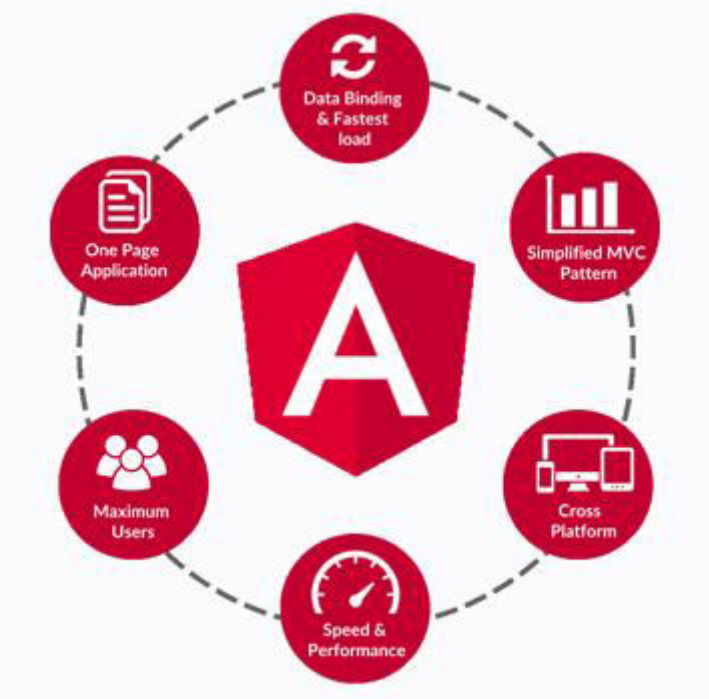

Figura 1. Avantajele framework-ului Angular [3]

Noțiunea de injectare a dependențelor în contextul Angular definește cum anumite porțiuni din cod interacționează unele cu altele și cum modificarea unei bucăți se propagă la altele. Acestea se definesc direct în cadrul componentelor și se folosesc entități numite injectori pentru a le cupla cu elemente din afara componentei. Prin acest proces se asigură reutilizarea componentei, fiind mai ușor de controlat și de testat.

Un alt aspect de care Angular face uz îl constituie serviciile. În general, componentele sunt create cu scopul de a prezenta datele primite de la server, nu și pentru salvarea directă a acestora. Aceste servicii au rolul de a distribui date între componente și, implicit, clase care nu sunt corelate una de alta. Angular are implementate deja anumite servicii printre care și injectoarele de dependențe. Toate aceste elemente cresc gradul de modularitate al aplicației și permit o implementare robustă. [4]

Spring Boot

Spring Boot reprezintă un framework pentru dezvoltatorii de cod Java care permite dezvoltarea de aplicații de tip servere pentru partea cu clientul pornind de la o configurație minimală. Spring Boot este configurat cu server de aplicație web de tip Tomcat într-o arhivă cu extensia .jar, deci pasul de configurare al serverului de backend este în general îndeplinit de acest framework. Aplicațiile de tip Spring Boot pot fi generate extrem de ușor folosind un tool de bootstrapping online numit Spring Initializr de unde se pot face toate setările proiectului. Ca și avantaje, Spring Boot aduce următoarele:

- scalabilitate înaltă
- toleranță la defecte

- capacitatea de a utiliza diferite tehnologii pentru mai multe microservicii
- implementare rapidă a fiecărui microserviciu

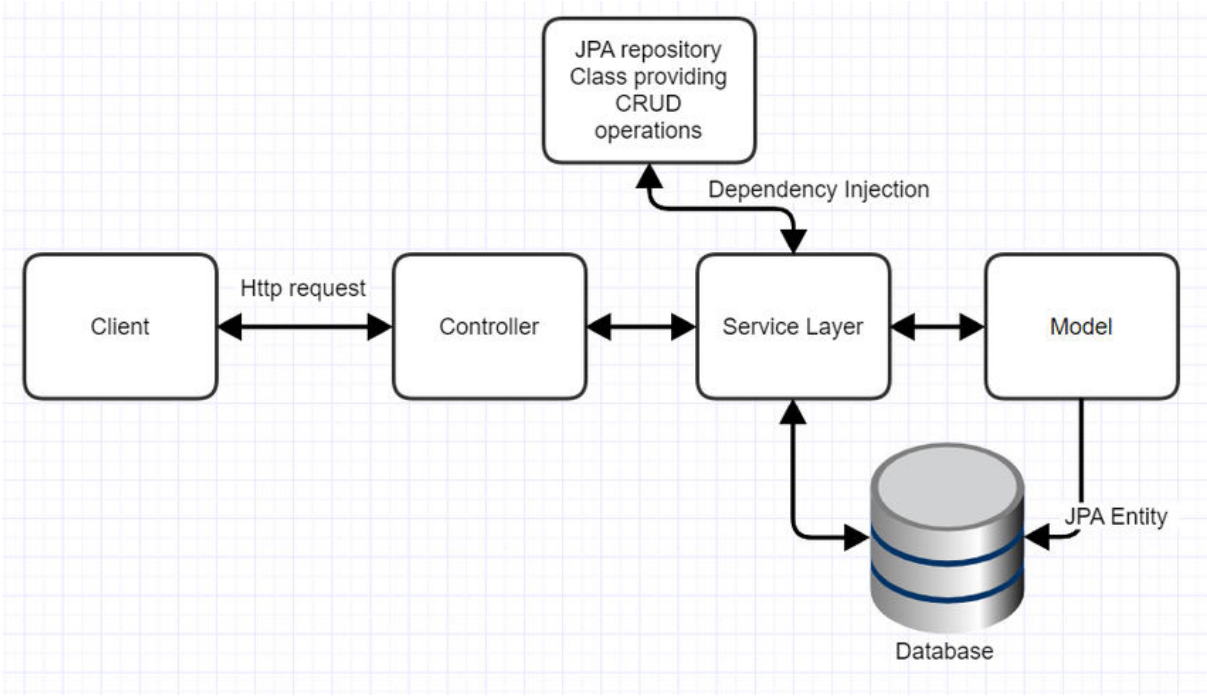

Figura 2. Arhitectura unei aplicații Spring Boot [5]

Framework-ul Spring Boot prezintă o arhitectură pe mai multe nivele, în care fiecare nivel comunică cu nivelul superior și inferior. Cele patru nivele sunt:

- nivelul de prezentare - acest nivel are rolul de a primi cererile din partea clientului, de a le procesa și de a emite un răspuns acestuia. În general, serverul este configurat astfel încât ca formatul răspunsului să fie JSON, deoarece este ușor de parsat. De asemenea, acest nivel se ocupă și cu autentificarea cererii HTTP înainte de a o transfera următorului nivel (nivelul de business).
- nivelul de business - acest nivel contine logica de funcționare a aplicației și deține clasele de servicii care interacționează cu baza de date. În privința cererilor, acest nivel are rolul de validare și autorizare a cererilor de a accesa resursele aplicației.
- nivelul de persistență - are sarcina de a converti diverse obiecte în entități (înregistrări) ce vor aparține bazei de date
- nivelul bazei de date - acesta este nivelul în care sunt stocate datele asupra cărora se execută operațiile CRUD (create, citire, modificare, ștergere). Este important de menționat faptul că Spring Boot conține un sistem propriu de management al datelor relaționale numit JPA (Java Persistence API) și poate fi integrat prin adăugarea dependenței JPA în interfața Spring Initializr.

2.3 Arhitectura aplicației

Schema arhitecturală a aplicației este prezentată în figura de mai jos.

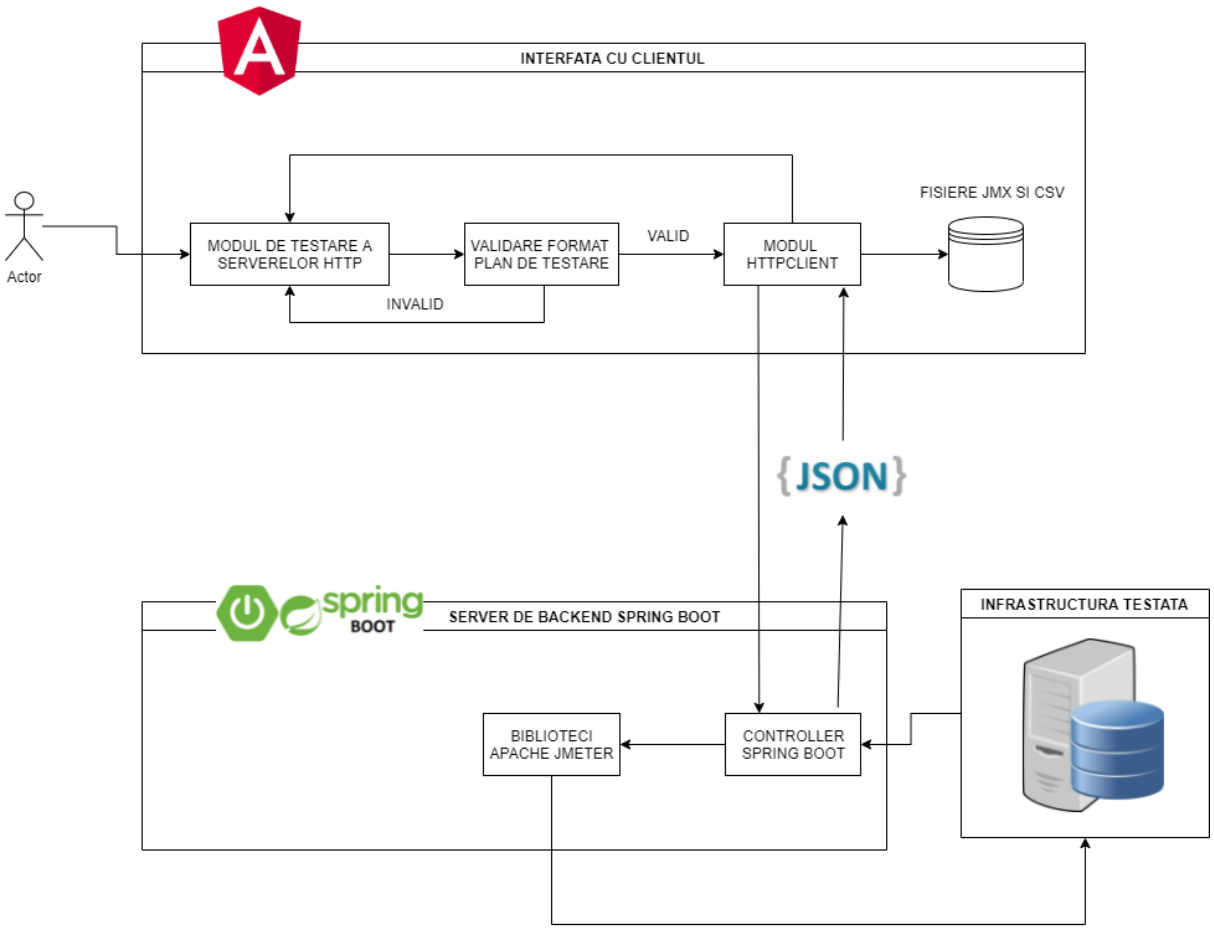

Figura 3. Arhitectura aplicației de Stress Testing

Pentru partea cu clientul, framework-ul Angular folosește în mod implicit portul 4200, iar serverul de Spring Boot rulează pe portul 8080. După completarea informațiilor necesare pentru crearea planului de testare și execuția sa, acestea sunt verificate înainte de a fi trimise de către modulul HTTP Client la serverul Spring Boot pentru a evita apariția unei erori sau excepții. Dacă datele din formular sunt validate atunci modulul HTTP Client creează un vector de parametri pe care îl pune în header-ul cererii HTTP către portul pe care rulează serverul de backend. Controllerul Spring Boot primește cererea din care extrage valorile parametrilor care vor fi folosiți pentru a inițializa diverse obiecte. Acestea vor fi preluate de mai multe funcții care sunt integrate în clase care fac parte din arhive .jar denumite biblioteci Apache JMeter [6]. Funcțiile vor executa planul de testare, iar rezultatele vor fi returnate serverului client în format JSON de către controller.

Parsarea rezultatelor se va face tot în cadrul modulului HTTP Client, deoarece răspunsul va fi prelucrat în mod asincron (rularea testelor se face pe o anumită perioadă, timp în care aplicația client poate executa alte task-uri) și nu va putea fi accesat din exteriorul acestui modul. Totuși, pentru ca rezultatul să fie stocat permanent se apelează la inițializarea unei variabile din storage-ul local al browser-ului în timpul apelului asincron.

Figura 4. Consola la deschiderea serverului de Spring Boot

La pornirea serverului de backend din proiectul Maven Spring Boot pe consola aplicației vor fi afișate diverse detalii despre configurația serverului. Astfel, se pot vizualiza detalii despre:

- versiunea de Spring Boot utilizată (v2.4.4);
- versiunea de Java (Java 8);
- PID-ul procesului generat de pornirea serverului;
- interfețele JPA;
- versiunea de server Tomcat (Apache Tomcat 9.0.44);
- portul pe care rulează serverul (8080);
- timpul de pornire al serverului;

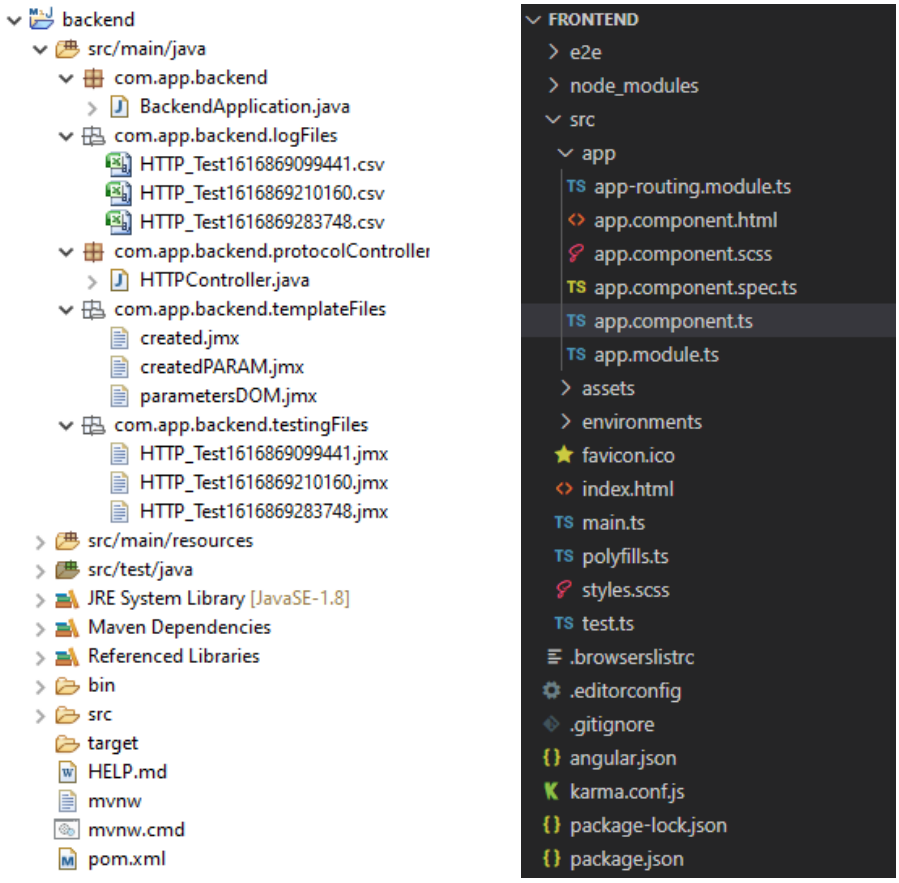

Figura 5. Structura proiectelor pentru partea de server și de client

Din punct de vedere arhitectural, partea de server de backend este compusă din mai multe directoare:

- com.app.backend - în acest folder se află clasa BackendApplication care lansează serverul de Spring Boot;
- com.app.backend.logFiles - aici sunt stocate fișierele cu extensia .csv care conțin statistici cu privire la diverse metrici colectate în timpul testelor de Stress Testing;
- com.app.backend.protocolController - momentan în acest director exista un singur fișier .java, cel corespunzător modului de testare al serverelor HTTP;
- com.app.backend.templateFiles - conține diverse fișiere .jmx necesare pentru crearea fișierelor cu planurile de testare;
- com.app.backend.testingFiles - aici se află planurile de testare menționate;

În ceea ce privește proiectul de frontend, structura fișierelor este următoarea:

- folderul src - conține fișierele principale pentru dezvoltarea aplicației Angular;
- folderul app - conține fișierele create de dezvoltator în cadrul componentelor;
- app.component.scss - este fișierul ce conține sintaxa CSS pentru stiluri;
- app.component.html - conține codul HTML pentru definirea elementelor de interfață;
- app.component.spec.ts - fișier de testare a componentei;
- app.component.ts - fișier Typescript ce conține logica de funcționare a componentei;
- app.module.ts - fișier ce conține toate importurile și dependențele necesare aplicației Angular;
- angular.json - fișier de configurare al proiectului;

3 Implementarea soluției tehnice

3.1 Interfața cu utilizatorul

Prima interfață care se deschide la accesarea url-ului http://localhost:4200 (la care se găsește resursa web) este pagina de introducere a detaliilor despre planul de testare.

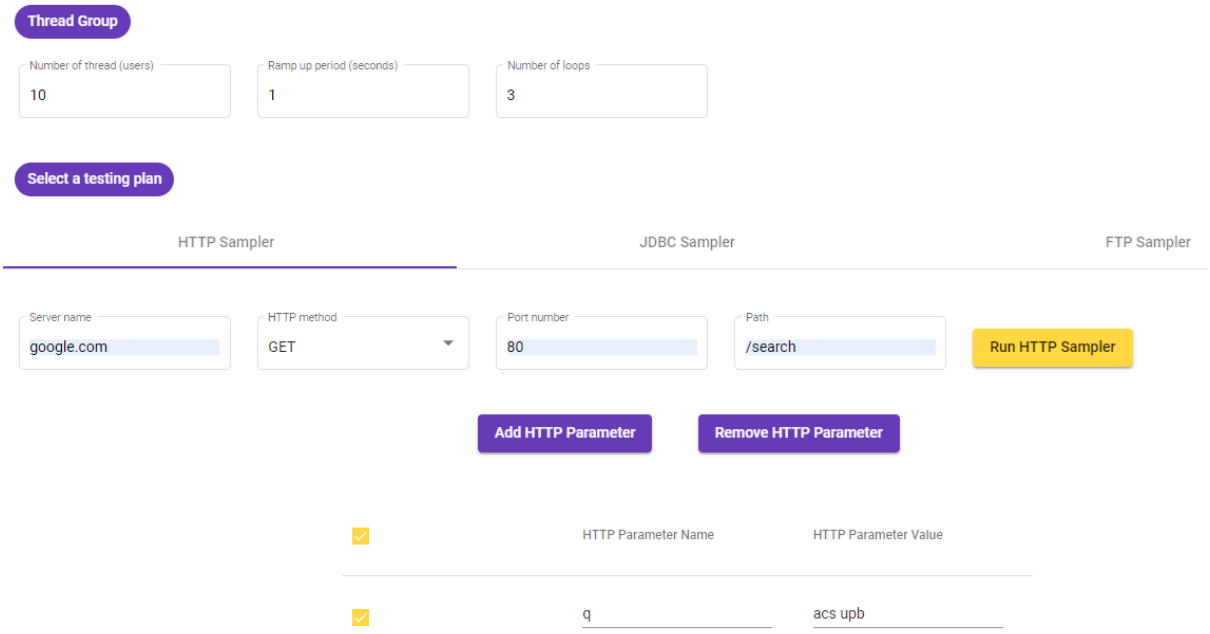

Figura 6. Interfața pentru crearea planului de testare

Așa cum a fost descris în partea de definire a funcționalităților, pagina de configurare a planului de testare conține câmpuri pentru definirea numărului de fire de execuție, a perioadei de generare între două fire de execuție consecutive și a numărului de cicluri de testare. Mai jos sunt cerute date în legătură cu serverul HTTP testat (numele de server, metoda HTTP asignată fluxului de cereri ce va fi generat, portul serverului, calea către resursa cerută și, opțional parametri de căutare). Utilizatorul are la dispoziție în josul paginii un tabel în care poate adăuga și șterge rânduri ce vor conține numele parametrului alături de valoarea acestuia. La terminarea planului, utilizatorul va apăsa butonul “Run HTTP Sampler”. Dacă cel puțin un câmp nu respectă regulile de definire atunci testul nu se va executa, iar utilizatorul va primi o alertă de tip “danger”.

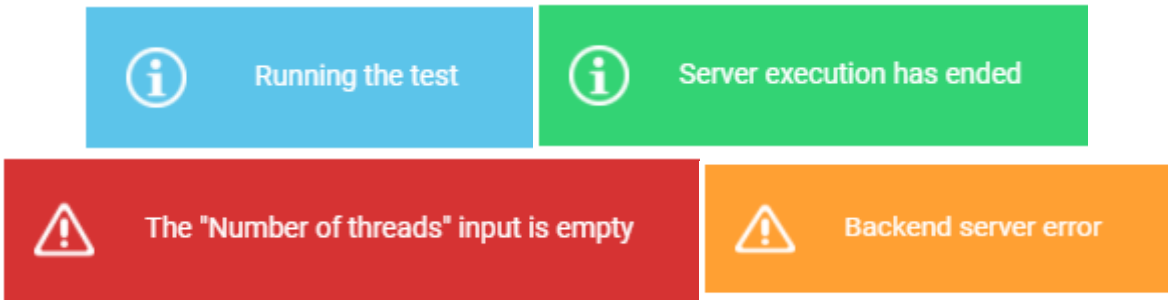

Figura 7. Diverse tipuri de alerte de tipul ngx-alerts în funcție de situație

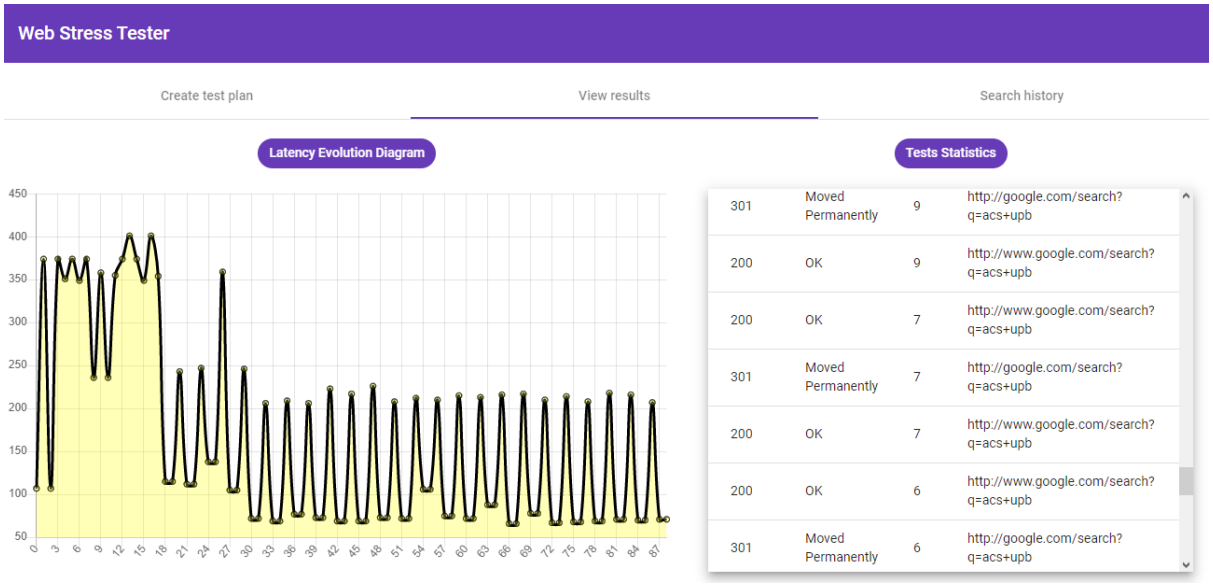

Figura 8. Graficul evoluției latenței și tabel cu informații despre răspunsurile din partea serverului HTTP testat

Pe pagina destinată vizualizării rezultatelor testului sunt amplasate mai multe elemente, astfel:

1. graficul evoluției latenței - graficul conține valoarea latenței pentru fiecare eșantion analizat (nu este în funcție de timp);
2. tabel ce conține informații despre răspunsurile din partea infrastructurii analizate - conține date precum codul răspunsului, mesajul corespunzător codului, numărul firului de execuție care a generat răspunsul și resursa testată;
3. un element de tip textarea care prezintă răspunsul primit de la serverul HTTP;
4. statistici despre testele desfășurate: numărul de teste, debitul (numărul de cereri care primesc răspuns în unitatea de timp), valorile medii, minime și maxime înregistrate la emiterea răspunsurilor pentru cererile trimise;

Pentru implementarea interfeței s-a făcut uz de componentele de UI puse la dispoziție de biblioteca Angular Material care a putut fi instalată în cadrul proiectului de frontend prin comanda:

ng add @angular/material

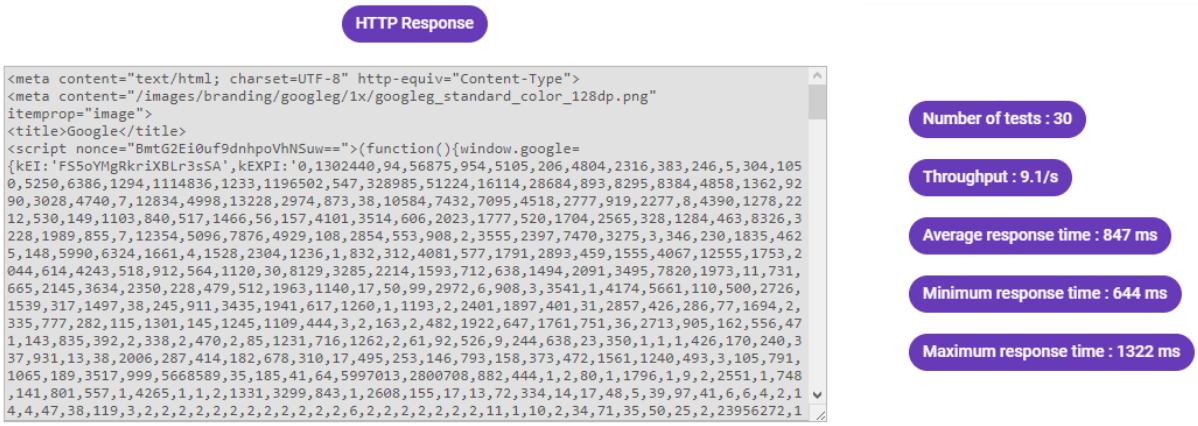

Figura 9. Răspunsul serverului HTTP și statistici legate de teste

De asemenea interfața prezinta si o rubrică în care se poate accesa conținutul fișierelor ce conțin planurile de testare. În partea stângă se află un tabel ce conține fișierele de testare și un buton pentru selectarea fișierului ce se dorește să fie parsat și afișat în elementul de tip textarea din partea dreaptă prin apăsarea butonului “Parse selected file” aflat sub tabel.

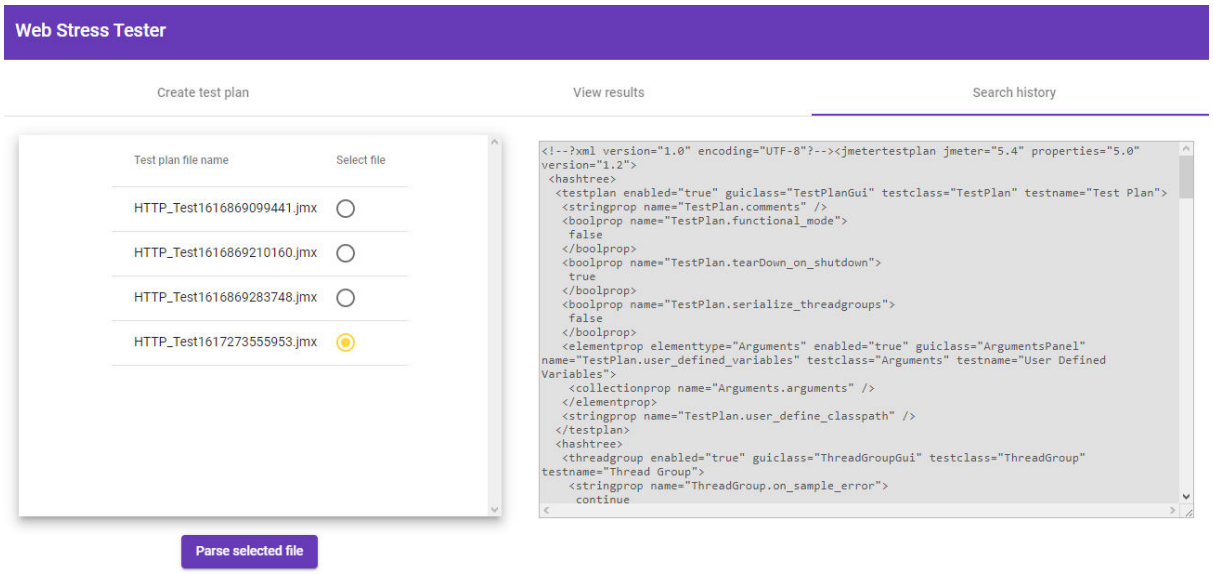

Figura 10. Vizualizare istoricului

3.2 Considerente legate de implementarea aplicației

Pentru crearea fișierului de controller pentru modulul de testare al serverelor HTTP a fost necesară importarea unor arhive .jar în cadrul proiectului de backend. Acestea sunt prezentate în figura de mai jos în partea stângă, iar lângă acestea sunt prezentate pachetele ce contine clasele a căror metode vor fi utilizate pentru lansarea în execuție a testelor în concordanță cu valorile primite de la interfața clientului.

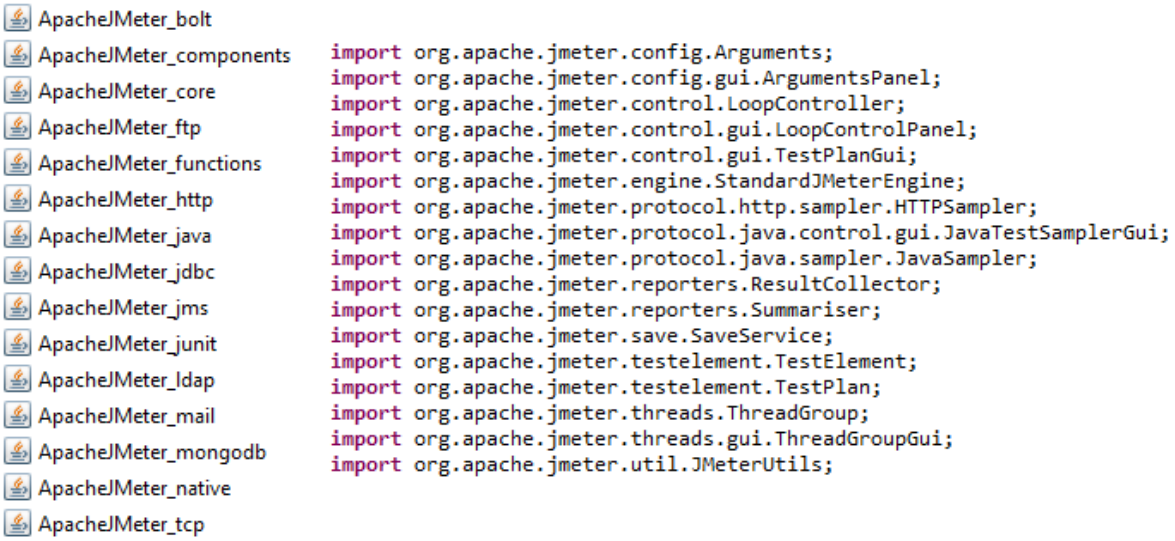

Figura 11. Arhivele .jar din biblioteca Apache JMeter și importurile în proiectul Spring Boot

În cadrul framework-ului Spring Boot există o serie de adnotări care au rolul de a oferi informații suplimentare despre program. Astfel, în cadrul programului există mai multe astfel de elemente, precum:

- @RestController - marchează o clasă ca fiind un handler pentru cererile HTTP;
- @CrossOrigin - permite cererile venite doar de la clientul specificat de atributul origins;
- @RequestMapping - este folosit pentru a mapa o funcție peste o serie de cereri HTTP. În acest caz, adnotarea @RequestMapping permite toate metodele implementat în HTTP (GET, POST, PUT, DELETE .etc);
- @RequestParam - este folosit pentru a extrage date din header-ul unei cereri HTTP și o variabila adnotată astfel echivalează cu un parametru din cerere;

```
@CrossOrigin(origins = "http://localhost:4200")
@RequestMapping("/api/http")
public List<Object> httpRequest(@RequestParam(required = false, value = "users") String users,
        @RequestParam(required = false, value = "ramp") String ramp,
        @RequestParam(required = false, value = "loops") String loops,
        @RequestParam(required = false, value = "server") String server,
        @RequestParam(required = false, value = "method") String method,
        @RequestParam(required = false, value = "port") String port,
        @RequestParam(required = false, value = "path") String path,
        @RequestParam(required = false, value = "httpparam") String httpparam)
        throws FileNotFoundException, IOException, ParserConfigurationException, SAXException, TransformerException {
```

Figura 12. Definirea funcției de rulare a testelor

Pentru început se defineste un obiect de tipul HTTPSampler, care conform documentației de la Apache este un tip de dată care este capabil să interpreteze cereri HTTP incluzând aspecte referitoare la cookie-uri și autentificare. Acest obiect are atribute, precum: numele de serverului, portul pe care acesta rulează, calea către resursa web, metoda indexată în header-ul cererii, dar și lista de parametri. Această listă este primită de la interfața Angular sub forma unui vector descris ca o succesiune “nume parametru - valoare parametru”.

```
// HTTP Sampler
HTTPSampler httpSampler = new HTTPSampler();
httpSampler.setDomain(server);
if(port != null && port != "")  httpSampler.setPort(Integer.parseInt(port));
httpSampler.setPath(path);
httpSampler.setMethod(method);
String[] paramsHttp = StringUtils.substringsBetween(httpparam , "\"", "\"");
for(int i=0; i<paramsHttp.length/2; i++) {
    httpSampler.addArgument(paramsHttp[2*i], paramsHttp[2*i+1]);
}
httpSampler.addTestElement(javaSampler);
```

Figura 13. Definirea obiectului HTTP Sampler

De asemenea, trebuie configurat și grupul de fire de execuție, librăria Apache JMeter punând la dispoziție clasele LoopController și ThreadGroup. Obiectul LoopController are ca atribut de interes numărul de cicluri de testare, iar pentru obiectul ThreadGroup trebuie setat numărul de fire de execuție și perioada de ramp-up (timpul dintre generarea a două cereri consecutive). Deoarece toți parametri din cererea de la serverul client sunt de tip String se face conversia la întreg prin metoda parseInt pentru a nu genera erori la apelul funcțiilor de setare pentru atributele menționate.

```
// Loop Controller
TestElement loopController = new LoopController();
((LoopController) loopController).setLoops(Integer.parseInt(loops));
loopController.addTestElement(javaSampler);
((LoopController) loopController).setFirst(true);
loopController.setProperty(TestElement.TEST_CLASS, LoopController.class.getName());
loopController.setProperty(TestElement.GUI_CLASS, LoopControlPanel.class.getName());
((LoopController) loopController).initialize();

// Thread Group
ThreadGroup threadGroup = new ThreadGroup();
threadGroup.setNumThreads(Integer.parseInt(users));
threadGroup.setRampUp(Integer.parseInt(ramp));
threadGroup.setName("Thread Group");
threadGroup.setSamplerController(((LoopController) loopController));
threadGroup.setProperty(TestElement.TEST_CLASS, ThreadGroup.class.getName());
threadGroup.setProperty(TestElement.GUI_CLASS, ThreadGroupGui.class.getName());
```

Figura 14. Definirea grupului de fire de execuție

Planului de testare i se adaugă un obiect de tip Summariser care are rolul de a colecta datele obținute în urma rulării testului prin instrucțiunea jmeter.run() și de a le salva într-un fișier cu extensia .csv. Numele fișierului este dat de un șir de caractere constant (HTTP_Test) la care se adaugă momentul de timp exprimat în milisecunde în raport cu ora 00:00, 1 Ianuarie 1970 (timpul UNIX) pentru a se asigura unicitatea numelor. De asemenea, biblioteca utilizată pune la dispoziție clasa SaveService care este utilizată pentru a salva datele din variabila "testPlanTree" ce conține toate variabilele planului de testare într-un fișier cu extensia .jmx la o locație specificată.

```
// Test Plan
TestPlan testPlan = new TestPlan("Create JMeter Script From Java Code");
testPlan.setProperty(TestElement.TEST_CLASS, TestPlan.class.getName());
testPlan.setProperty(TestElement.GUI_CLASS, TestPlanGui.class.getName());
testPlan.setUserDefinedVariables((Arguments) new ArgumentsPanel().createTestElement());

// Construct Test Plan from previously initialized elements
testPlanTree.add(testPlan);
HashTree threadGroupHashTree = testPlanTree.add(testPlan, threadGroup);
threadGroupHashTree.add(javaSampler);

String fileName = String.valueOf(System.currentTimeMillis());

// save generated test plan to JMeter's .jmx file format
SaveService.saveTree(testPlanTree, new FileOutputStream(
        "D:\\facultate\\Anul 5\\cercetare\\backend\\backend\\src\\main\\java\\com\\app\\backend\\testingFiles\\HTTP_Test"
                + fileName + ".jmx"));

// add Summarizer output to get test progress
Summariser summer = null;
String summariserName = JMeterUtils.getPropDefault("summariser.name", "summary");
if (summariserName.length() > 0) {
    summer = new Summariser(summariserName);
}

// Store execution results into a .csv file
String logFile = "D:\\facultate\\Anul 5\\cercetare\\backend\\backend\\src\\main\\java\\com\\app\\backend\\logFiles\\HTTP_Test"
        + fileName + ".csv";
ResultCollector logger = new ResultCollector(summer);
logger.setFilename(logFile);
testPlanTree.add(testPlanTree.getArray()[0], logger);

jmeter.configure(testPlanTree);
jmeter.run();
```

Figura 15. Execuția planului de testare și colectarea rezultatelor

Structura unui fișier .jmx ce conține un plan de testare este prezentat în figura următoare. Acesta conține toate elementele specificate în interfața web în format XML, fiind foarte ușor de parsat și manipulat. Limbajul Java utilizat pentru crearea serverului de backend permite utilizarea claselor DocumentBuilder și DocumentBuilderFactory pentru parsarea rapidă a documentelor cu structuri de tip XML (printre care și fișierele cu extensia .jmx).

```
    <stringProp name="ThreadGroup.on_sample_error">continue</stringProp>
    <elementProp elementType="LoopController"
                 enabled="true"
                 guiclass="LoopControlPanel"
                 name="ThreadGroup.main_controller"
                 testclass="LoopController"
                 testname="Loop Controller">
       <boolProp name="LoopController.continue_forever">false</boolProp>
       <stringProp name="LoopController.loops">3</stringProp>
    </elementProp>
    <stringProp name="ThreadGroup.num_threads">10</stringProp>
    <stringProp name="ThreadGroup.ramp_time">1</stringProp>
    <boolProp name="ThreadGroup.scheduler">false</boolProp>
    <stringProp name="ThreadGroup.duration"/>
    <stringProp name="ThreadGroup.delay"/>
    <boolProp name="ThreadGroup.same_user_on_next_iteration">true</boolProp>
 </ThreadGroup>
 <hashTree>
    <HTTPSamplerProxy enabled="true"
                      guiclass="HttpTestSampleGui"
                      testclass="HTTPSamplerProxy"
                      testname="HTTP Request">
       <elementProp elementType="Arguments"
                    enabled="true"
                    guiclass="HTTPArgumentsPanel"
                    name="HTTPsampler.Arguments"
                    testclass="Arguments"
                    testname="User Defined Variables">
          <collectionProp name="Arguments.arguments">
             <elementProp elementType="HTTPArgument" name="q">
                <boolProp name="HTTPArgument.always_encode">true</boolProp>
                <stringProp name="Argument.value">acs upb</stringProp>
                <stringProp name="Argument.metadata">=</stringProp>
                <boolProp name="HTTPArgument.use_equals">true</boolProp>
                <stringProp name="Argument.name">q</stringProp>
             </elementProp>
          </collectionProp>
       </elementProp>
```

Figura 16. Porțiune din structura unui plan de testare

La partea de Angular a aplicației de testare, apelul către o resursă din serverul Spring Boot se face printr-o variabilă de tipul HttpClient declarată în constructorul clasei. Aceasta poartă denumirea de “http” și i se poate atașa în cadrul apelului un atribut “param” care constituie o serie de parametri ai cererii. De asemenea, apelului asincron i se aplică metoda subscribe() care are rolul de a colecta temporar rezultatele venite din partea serverului apelat. Pentru a asigura persistența răspunsului se stochează răspunsul într-un local storage. Avantajul framework-ului Angular este acela că la apelul funcției subscribe() se poate alege ca eventualele erori să fie afișate în consola browserului, aspect ce face mai ușoară identificarea bug-urilor si rezolvarea lor.

```
onParseJMX() {
  if (this.primaryContact.fileName.length > 0) {
    let params = new HttpParams().set('test', this.primaryContact.fileName);
    this.http
      .get('http://localhost:8080/api/getPlansDetails', { params: params })
      .subscribe(
        (result) => {
          this.plan = JSON.parse(JSON.stringify(result));
        },
        (error) => {
          console.log(error);
          this.alertService.warning('Backend server error');
        }
      );
  } else {
    this.alertService.warning("You haven't selected a plan file");
  }
}
```

Figura 17. Apelul modulului HTTPClient către endpoint-ul "/api/getPlansDetails" care mapează o funcție din controller-ul HTTP

La inițializarea procedeului de testare de tipul Stress Testing pe consola mediului de dezvoltare Java utilizat (Eclipse versiunea 2018-09) sunt afișate detalii despre evoluția testelor, precum:

- numărul de fire de execuție activate;
- perioada de suspendare a acestora (5000 nanosecunde);
- mesaj ce atestă pornirea fiecărui fir de execuție;
- un sumar (pot fi mai multe) ce conține elemente despre conectarea la serverul testat: debit, timpul de răspund mediu, minim și maxim, numărul de teste picate, numărul de fire de execuție active și terminate;
- mesaj ce arată terminarea execuției thread-urilor;

```
2021-04-02 19:13:00.926  INFO 2164 --- [nio-8080-exec-3] o.a.jmeter.engine.StandardJMeterEngine   : All thread groups have been started
2021-04-02 19:13:00.961  INFO 2164 --- [hread Group 1-1] org.apache.jmeter.samplers.SampleResult  : Note: Sample TimeStamps are START times
2021-04-02 19:13:00.961  INFO 2164 --- [hread Group 1-1] org.apache.jmeter.samplers.SampleResult  : sampleresult.default.encoding is set to ISO-8859-1
2021-04-02 19:13:00.961  INFO 2164 --- [hread Group 1-1] org.apache.jmeter.samplers.SampleResult  : sampleresult.useNanoTime=true
2021-04-02 19:13:00.961  INFO 2164 --- [hread Group 1-1] org.apache.jmeter.samplers.SampleResult  : sampleresult.nanoThreadSleep=5000
2021-04-02 19:13:01.019  INFO 2164 --- [hread Group 1-2] org.apache.jmeter.threads.JMeterThread   : Thread started: Thread Group 1-2
2021-04-02 19:13:01.118  INFO 2164 --- [hread Group 1-3] org.apache.jmeter.threads.JMeterThread   : Thread started: Thread Group 1-3
2021-04-02 19:13:01.216  INFO 2164 --- [hread Group 1-4] org.apache.jmeter.threads.JMeterThread   : Thread started: Thread Group 1-4
2021-04-02 19:13:01.316  INFO 2164 --- [hread Group 1-5] org.apache.jmeter.threads.JMeterThread   : Thread started: Thread Group 1-5
2021-04-02 19:13:01.416  INFO 2164 --- [hread Group 1-6] org.apache.jmeter.threads.JMeterThread   : Thread started: Thread Group 1-6
2021-04-02 19:13:01.516  INFO 2164 --- [hread Group 1-7] org.apache.jmeter.threads.JMeterThread   : Thread started: Thread Group 1-7
2021-04-02 19:13:01.618  INFO 2164 --- [hread Group 1-8] org.apache.jmeter.threads.JMeterThread   : Thread started: Thread Group 1-8
2021-04-02 19:13:01.707  INFO 2164 --- [hread Group 1-9] org.apache.jmeter.threads.JMeterThread   : Thread started: Thread Group 1-9
2021-04-02 19:13:01.809  INFO 2164 --- [read Group 1-10] org.apache.jmeter.threads.JMeterThread   : Thread started: Thread Group 1-10
2021-04-02 19:13:02.283  INFO 2164 --- [hread Group 1-3] org.apache.jmeter.reporters.Summariser   : summary +      1 in 00:00:02 =    0.6/s Avg:  1096 Min:  1096 Max:  1096 Err:
summary +      1 in 00:00:02 =    0.6/s Avg:  1096 Min:  1096 Max:  1096 Err:     0 (0.00%) Active: 10 Started: 10 Finished: 0
2021-04-02 19:13:04.969  INFO 2164 --- [hread Group 1-3] org.apache.jmeter.threads.JMeterThread   : Thread is done: Thread Group 1-3
2021-04-02 19:13:04.969  INFO 2164 --- [hread Group 1-3] org.apache.jmeter.threads.JMeterThread   : Thread finished: Thread Group 1-3
2021-04-02 19:13:05.046  INFO 2164 --- [hread Group 1-1] org.apache.jmeter.threads.JMeterThread   : Thread is done: Thread Group 1-1
2021-04-02 19:13:05.047  INFO 2164 --- [hread Group 1-1] org.apache.jmeter.threads.JMeterThread   : Thread finished: Thread Group 1-1
2021-04-02 19:13:05.355  INFO 2164 --- [hread Group 1-6] org.apache.jmeter.threads.JMeterThread   : Thread is done: Thread Group 1-6
2021-04-02 19:13:05.355  INFO 2164 --- [hread Group 1-6] org.apache.jmeter.threads.JMeterThread   : Thread finished: Thread Group 1-6
2021-04-02 19:13:05.387  INFO 2164 --- [hread Group 1-5] org.apache.jmeter.threads.JMeterThread   : Thread is done: Thread Group 1-5
2021-04-02 19:13:05.388  INFO 2164 --- [hread Group 1-5] org.apache.jmeter.threads.JMeterThread   : Thread finished: Thread Group 1-5
2021-04-02 19:13:05.420  INFO 2164 --- [hread Group 1-4] org.apache.jmeter.threads.JMeterThread   : Thread is done: Thread Group 1-4
2021-04-02 19:13:05.420  INFO 2164 --- [hread Group 1-4] org.apache.jmeter.threads.JMeterThread   : Thread finished: Thread Group 1-4
2021-04-02 19:13:05.503  INFO 2164 --- [hread Group 1-2] org.apache.jmeter.threads.JMeterThread   : Thread is done: Thread Group 1-2
2021-04-02 19:13:05.504  INFO 2164 --- [hread Group 1-2] org.apache.jmeter.threads.JMeterThread   : Thread finished: Thread Group 1-2
2021-04-02 19:13:06.006  INFO 2164 --- [read Group 1-10] org.apache.jmeter.threads.JMeterThread   : Thread is done: Thread Group 1-10
2021-04-02 19:13:06.006  INFO 2164 --- [read Group 1-10] org.apache.jmeter.threads.JMeterThread   : Thread finished: Thread Group 1-10
2021-04-02 19:13:06.151  INFO 2164 --- [hread Group 1-7] org.apache.jmeter.threads.JMeterThread   : Thread is done: Thread Group 1-7
2021-04-02 19:13:06.151  INFO 2164 --- [hread Group 1-8] org.apache.jmeter.threads.JMeterThread   : Thread is done: Thread Group 1-8
2021-04-02 19:13:06.151  INFO 2164 --- [hread Group 1-7] org.apache.jmeter.threads.JMeterThread   : Thread finished: Thread Group 1-7
2021-04-02 19:13:06.151  INFO 2164 --- [hread Group 1-8] org.apache.jmeter.threads.JMeterThread   : Thread finished: Thread Group 1-8
2021-04-02 19:13:06.855  INFO 2164 --- [hread Group 1-9] org.apache.jmeter.threads.JMeterThread   : Thread is done: Thread Group 1-9
2021-04-02 19:13:06.856  INFO 2164 --- [hread Group 1-9] org.apache.jmeter.threads.JMeterThread   : Thread finished: Thread Group 1-9
2021-04-02 19:13:06.857  INFO 2164 --- [nio-8080-exec-3] o.a.jmeter.engine.StandardJMeterEngine   : Notifying test listeners of end of test
```

Figura 18. Evoluția testelor în cadrul consolei platformei Eclipse Java

4 Concluzii

În cadrul acestui raport de cercetare aplicativă au fost prezentate aspecte legate de implementarea primei părți ale aplicației de testare. Aceasta permite analiza serverelor pe bază de protocol HTTP prin urmărirea anumitor parametri: latență, rata de răspuns a cererilor în unitatea de timp și, respectiv, timpii de răspuns.

Conform capturii de ecran de la figura cu numărul 9 se observă un timp de răspuns mediu de 644 de milisecunde. Pentru a realiza o comparație se execută în linie de comandă executabilul ping către serverul google.com. Rezultatul este prezentat în figura de mai jos.

```
Pinging google.com [172.217.19.110] with 32 bytes of data:
Reply from 172.217.19.110: bytes=32 time=32ms TTL=115
Reply from 172.217.19.110: bytes=32 time=32ms TTL=115
Reply from 172.217.19.110: bytes=32 time=31ms TTL=115
Reply from 172.217.19.110: bytes=32 time=31ms TTL=115

Ping statistics for 172.217.19.110:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 31ms, Maximum = 32ms, Average = 31ms

C:\Users\popae>ping -l 266 google.com

Pinging google.com [172.217.19.110] with 266 bytes of data:
Reply from 172.217.19.110: bytes=68 (sent 266) time=31ms TTL=115
Reply from 172.217.19.110: bytes=68 (sent 266) time=31ms TTL=115
Reply from 172.217.19.110: bytes=68 (sent 266) time=32ms TTL=115
Reply from 172.217.19.110: bytes=68 (sent 266) time=31ms TTL=115

Ping statistics for 172.217.19.110:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 31ms, Maximum = 32ms, Average = 31ms
```

Figura 19. Rezultatul comenzii ping cu dimensiunea predefinită a pachetelor de 32 bytes / 266 bytes

În mod predefinit dimensiunea pachetelor ping este de 32 de bytes, iar pentru testul de Stress Testing prezentat în raport, dimensiunea maximă a pachetelor este de aproximativ 266 de bytes. La execuția aceleași comenzi ping dar cu opțiunea "packetsize" setată la 266, rezultatele sunt asemănătoare. Un argument pentru această inconsistență este faptul că se lansează 10 fire de execuție, dar sistemul host care suportă inițierea operației de testare deține 4 core-uri. Din asta rezultă că la un moment dat se pot executa maxim 4 fire de execuție din cele 10 ceea ce duce la un paralelism parțial, existând fire de execuție ce se execută secvențial.

Măsurând timpul de execuție al instrucțiunii jmeter.run() se constată că timpul necesar este aproximativ de 10.000 de milisecunde (10 secunde), interval în care s-au realizat mai multe operații decât înregistrarea valorilor latenței, precum partea de management a thread-urilor care se execută parțial paralel, colectarea datelor de la server. În acest fel se poate justifica diferența dintre valorile latențelor înregistrate de utilitarul ping și cele din fișierul de log.

5 Bibliografie

[1] Franciszek Stodulski (2020) - 7 Benefits of Angular You Should Know if You Want to Build a Digital Product - disponibil online la adresa https://www.netguru.com/blog/benefits-of-angular

[2] Anastasiia Shybeko - 8 ADVANTAGES OF ANGULAR FOR BUSINESSES AND DEVELOPERS - disponibil online la adresa https://light-it.net/blog/8-advantages-of-angular-for-businesses-and-developers/

[3] Schemă Angular - https://www.minddigital.com/angular-js-development/

[4] Documentație oficială Angular - https://angular.io/tutorial/toh-pt4

[5] Rakshith Rao (Mai 2020) - Review on Spring Boot and Spring Webflux for Reactive Web Development - disponibil online la adresa

https://www.researchgate.net/figure/Fig-2-Architecture-flow-of-spring-boot-Applications-Spring-boot-uses-all-the-features_fig2_341151097

[6] Documentație biblioteca Apache JMeter
https://jmeter.apache.org/api/org/apache/jmeter/JMeter.html