INFORMAȚII ÎN TIMP REAL ALE UNUI SERVER

Diogen Babuc, Cristian Pal, Sergiu Ardelean
Universitatea de Vest, Timișoara
Facultatea de Matematică și Informatică
Informatică
Anul II
Proiect Colectiv

email-uri:
diogen.babuc00@e-uvt.ro
cristian.pal93@e-uvt.ro
sergiu.ardelean98@e-uvt.ro

23 mai 2020

I Introducere

Peisajul IT continuând să se îndrepte către medii cu platformă mixtă, administratorii IT și furnizorii de servicii gestionate (MSP) se luptă cu modul de gestionare a sistemelor Windows, macOS și Linux pe care le găsesc în mediile lor.

Simplitatea este că nu există mai multe medii IT care să fie omogene, doar pentru Windows. Zilele acelea au trecut. Acum, practic, fiecare organizație de pe planetă conține un amestec de mașini Windows, Mac și Linux. Fiecare sistem de operare are apelul său către diferiți utilizatori, departamente și cazuri de utilizare, astfel încât este nechibzuit să încercați să eliminați oricare dintre ele din singurul motiv de a nu le putea gestiona. De exemplu, odată cu creșterea platformei macOS Apple și a serverelor Linux la Amazon Web Services (AWS).

Această aplicație exprimă un suport de monitorizare a stării sistemului în timp real pentru client. Un client poate avea mai multe servere, respectiv mai mulți clienți se pot conecta simultan la același sistem. Pentru ca utilizatorul să se poată conecta la sistemul pe care îl deține, va trebui să urmeze anumite instrucțiuni. Pentru început, este important să știe că această aplicație este accesibilă de pe orice dispozitiv și orice sistem de operare, care are în dotare un browser web și conexiune la internet, printr-un *URL* specificat. După accesare, utilizatorul va fi bine venit pe pagina de logare. Dacă nu are cont, are posibilitatea de a se înregistra, apoi, va avea oportunitatea de a-și introduce datele de conectare pentru serverul pe care îl deține, prin completarea câmpurilor cum sunt *IP*-ul, port-ul și numele. După crearea serverelor, obiectele se vor adăuga într-o listă, de unde utilizatorul selectează serverul dorit, urmând conexiunea propriu-zisă și nu doar atât, ci mai are opțiunea de a primi notificări în cazul în care este offline și există probleme cu serverul respectiv. După acceptarea conexiunii, utilizatorul o să observe datele serverelui selectat, reprezentate grafic print-un dashboard. La finalizarea monitorizării, utilizatorul va putea să se deconecteze de la server, iar ulterior va putea ieși și din aplicație.

Scopul creării acestei aplicații oferă posibilitatea de monitorizare a serverelor la distanță a unui utilizator.

II Descriere formală

În etapa de descriere a arhitecturii proiectului, care conține funționalitățile necesare pentru crearea proiectului și desfășurarea asamblărilor, membrii echipei au efectuat o descriere detaliată a sistemului software, a bazei de date și a componentelor securității, precum și conexiunea acestor proprietăți între ele. Aplicația este una compusă din două părți. Anume, prima parte rulează pe fiecare server al clientului, iar cealaltă parte ne oferă un *web server* ce poate fi accesat prin intermediului unui web browser, care conține flexibilitatea de rulare și pe dispozitivele mobile, fără alte necesități speciale.

Aplicația din perspectiva fluxului de control presupune lansarea serverului web, care furnizează *link*-ul URL către client, iar la accesarea acestui link, apare pagina de logare, unde și începe fluxul de control. Fluxul de control are ca și precondiție existența contului de utilizator în baza de date. În caz contrar, ca și flux alternant, utilizatorul are oportunitatea de a-și crea un cont de utilizator. Procesul de înregistrare presupune introducerea unei adrese de e-mail valide. Procesul de înregistrare se desfășoară printr-un formular, care necesită să fie completat corect. Anume, este de așteptat ca, în cazul unei completări incorecte (numele utilizatorului sau adresa de e-mail deja ocupată), pe pagina de înregistrare aceste excepții să fie tratate cu mare atenție. Completând corect procesul de înregistrare, clientul este transmis pe pagina de logare, în care se poate conecta la aplicație, prin introducerea validă a datelor înscrise în câmpurile de nume de utilizator și parolă (care este criptată - inevitabil - la înregistrare și logare). Pentru a-și introducere serverele proprii – care se adaugă și în lista de servere, stocată pe pagina cu servere înregistrate – Clientul poate să apese pe butonul de adăugare a serverelor în cadrul profilului său. Va fi prefigurat că trebuie să completeze câmpul de nume de server, IP și port, unde – iarăși inevitabil – datele vor fi unice. În caz contrar, se indică o excepție pentru a se evita conflictul de date sau, eventual, suprascrierea lor, care poate să perturbe întregul sistem. În cazul în care un utilizator, accesând pagina cu lista de servere și selectând un server din listă, nu se poate conecta la server, înseamnă că serverul respectiv nu a fost pornit, astfel încât aplicația de client nu s-a putut lega de server. Totuși, pentru cazul în care serverul nu este online, se aruncă o excepție, „Server Not Found”. Datele despre server vor fi reprezentate grafic, în timp real, pentru utilizator, sub forma unor diagrame dreptunghi (bar charts) și diagrame cerc. Utilizatorul va putea, totodată, să mărească diagrama dorită, cu scop de a observa mai bine datele despre server. Pentru a se deconecta de la un server anumit, utilizatorul va fi întors – prin butonul de deconectare de la server – la pagina web care conține lista de servere, cu posibilitatea de accesare a datelor despre un alt server. De asemenea, utilizatorul poate sterge un server expirat (nefuncțional) din listă.

Apăsând pe butonul de ieșire din aplicație (Logout), care se află pe meniul de navigare al aplicației, utilizatorul o va părăsi pe aceasta, iar datele lui vor rămâne stocate în bază astfel încât se va putea loga oricând ar dori.

În procesul de conectare la server, serverul web se conecteaza la opțiunea selectată de utilizator, prin intermediul de IP adresă și port. Conexiunea este sigură, deoarece ea se efectuează în mod prefixat, utilizând un mesaj dinamic. Se tratează și excepția de securitate, în care calea alternantă este o indicare de tip mesaj, semnalizat în pagina *html*, care avertizează utilizatorul că nu există o conexiune la server. Dacă a reușit conexiunea, serverul web execută o cerere cu mesaj prefixat pentru a obține date reale, într-un interval de timp. Datele extrase sunt afișate pe un bord (dashboard).

Pentru a accesa datele, serverul aplicației detectează sistemul de operare pe care este rulat serverul. În functie de selecție, pentru sistemul *Mac* se folosește librăria SMC produsă de Apple. Pe Mac excepția apare când valoarea nu este găsită și, pentru semnalare, se returnează valoarea zero. Pe sistemul de operare Linux se utilizează librăria „psutil”, iar pe Windows librăria „psutil” și anumite scripturi (ex. *PowerShell*). Datele sunt salvate într-un dicționar cu chei și valori. Valoarea

din dicționar este obținuta datorită acestor librării. După ce dicționarul s-a finalizat, acesta este serializat și trimis pe fir. Serverul web recepționează aceste pachete, deserializând mesajele cutare în obiecte de tip dicționar, în care cheile sunt axa *Ox* și *Oy*, iar valorile dicționarului sunt numele componentelor serverului, respectiv valorile acestora. Din dicționarele menționate, datele sunt extrase pentru a fi reprezentate grafic.

Serverul aplicației conține o clasa de notificare, care dispune de mesaje predefinite pentru fiecare alertă în parte (temperatura ridicată, viteza ventilatorului peste limitele permise, etc). Bineînțeles, aceste alerte pot fi dezactivate de utilizator doar atunci când el este online. Notificarea se execută prin adresa de e-mail și este compusă din atâtea alerte câte apar pentru acel server. Procesul de creare a notificărilor presupune definirea a două tipuri de valori pentru fiecare componentă. În clasa de notificări s-au definit și câteva valori constante pentru a reprezenta frontiera superioară a componentelor. În cazul în care valoarea citită depășește pragul de sus al constantei, se crează alerta și afisează un mesaj. Valoarea – se presupune – este optimă atunci când nu depășește valoarea constantei propuse. Deoarece aplicația permite notificarea pentru un server comun, notificarea se trimite utilizatorilor care dețin acel server, introducându-se automat în lista de utilizatori pentru serverul accesat. Inainte ca adresele de e-mail să fie adăugate în lista de utilizatori, ele sunt verificate. Condiția suficientă este aceea de a fi conform standardelor de compunere a unei adrese de e-mail. După pornirea notificării, aplicația lucrează independent; anume, se crează un fir de execuție. Atunci când postcondiția este indeplinită, clasa de notificări rulează independent de aplicația serverului web. Postcondiția constă în a avea măcar o adresa de e-mail valida în lista de utilizatori, iar acel utilizator să aibă notificarea pornită.

III Tehnologii utilizate

Procesul de înregistrare la aplicație presupune introducerea validă a tuturor datelor, cum ar fi de pildă adresa de email, sau introducerea a datelor unice pentru utilizator. Adică, adresa de email și numele de utilizator să fie unice. Pagina de înregistrare și cea de logare au fost realizate folosind tehnologia HTML, CSS și Javascript, cu menționarea faptului că formele pentru înregistrare și logare (care au permis inserarea datelor în baza de date) au fost construite în Python (tehnologia Flask pe care o deține limbajul Python), cu ajutorul clasei predefinite din pachetul importat, numit *FlaskForm*. Pentru inserarea datelor în baza s-au folosit metodele de *add()* și *commit()*, predefinite în librăria *flask_sqlalchemy* și apelate în funcția definită de programator, numită *save().* După întregistrarea cu succes, utilizatorul va fi redirectat la pagina de logare (prin apelul funcției predefinite de render_template și apelul parametrului predefinit al funcției, numit *content)*, unde va trebui sa-și introducă parola și numele de utilizator aferente celor create anterior. Se va verifica dacă parola este corectă, accesând parola din baza de date folosind metoda *first()* din pachetul *query* acordat clasei User definită de programator. Considerând partea de securitate, ea a fost tratată cu mare atenție, deoarece s-au folosit anumite funcții predefinite din pachetul *wtf.validators* (Email(), DataRequired(), InputRequired(), Length()). După logarea corectă, utilizatorul are acces la pagina de profil al utilizatorului, deoarece a fost redirectat la URL-ul pentru această pagină, apelând funcția *redirect(url_for(...))*, predefinită în pachetul importat *flask*. Pentru a descoperi datele despre utilizatorul curent, și a prelucra cu datele pe care el le solicită, au fost introduse funcțiile *login_user, logout_user, current_user, login_required* din pachetul *flask_login*.

Utilizatorul va introduce servere în aplicație apăsând pe un buton creat în HTML, care conține funcția *addServerInLista()* în Javascript, aferentă acțiunii necesare. Poate, de asemenea, să șteargă serverele pe care nu le mai dorește. Aplicația automat detectează care server a dorit să fie șters, deoarece fiecare server la creare are și un buton *Delete.* Prin forma accesată în fișierul *views.py* și transferată prin content în pagina HTML, folosind tehnologia *Jinja2*, se detectează serverul care este dorit să fie șters prin apelul funcției form.name_serv(). Datele din bază (despre server sau utilizator) sunt afișate pe ecran folosind tehnologia Jinja2 ({{...}}).

Procesul de preluare a datelor în timp real presupune accesarea unui mecanism de declanșări a serverului implementat în aplicația pentru clientul rețelei de calculatoare, care solicita datele de la aplicația implementată pentru serverul rețelei. Cererea se execută prin IP și port, iar datele returnate se stochează în funcții definite de programator. Clientul aplicației va fi un mediu de transport al datelor, până ele nu vor ajunge pe pagina web, pentru a fi reprezentate vizual. După încheierea procesului de preluare a datelor, se termină conexiunea între interfața grafică și clientul aplicației, pe care utilizatorul a solicitat-o. Lucrul respectiv s-a desfășurat prin crearea unei instanțe de tip Informație, prin intermediul căreia s-au extras datele, apelându-se funcțiile *get()* pentru componentele extrase. S-au extras componente care au valori constante, precum și cele cu valori variabile (*getDataOnce()* și *getDataRepeat()*). Pe conținutul datelor apoi s-a apelat funcția render_template(), în care s-au introdus componentele, pentru a fi transportate în fișierul *HTML,* unde au fost stocate în variabile folosind *Jinja2.* În fișierul HTML, datele au fost introduse în diagrame și reprezentate prin pachetul Plotly și metoda newPlot() în Javascript, sau pur și simplu afișate pe ecran folosind acolade duble (Jinja2). Dacă o valoare a componentei sistemului depășește limita superioară a intervalului cu valori aferente unei componente, lucrul respectiv se semnalează sub forma unei alerte, care se trimite prin notificări la adresa de email a utilizatorului. În cod s-a citit din baza de date adresa de email a utilizatorului curent, prin variabila *email = request.form.get('email', '', type=str)* și de asemenea, s-a verificat dacă utilizatorul are notificările pornite (daca a apăsat pe butonul CheckBox). În cazul în care nu are notificările pornite, se va returna 0, iar dacă are, se va returna valoarea 1, în funcția *check_selected()*. Aceste două variabile vor deveni parametri pentru funcția request() din clasa Informatie, definită de programator. În cazul

în care utilizatorul a accesat o pagină (URL) eronată, atunci va fi apelată funcția handle_bad_request(), care ne îndrumă (prin werkzeug.exceptions.NotFound din Flask) spre o pagină 404.

Când vine vorba de baza de date, datele au fost stocate în SQLite. Baza de date a fost creată și modificată din Flask, folosind instrucțiunile din pachetul *flask_sqlalchemy*. Pentru a conecta baza de date cu datele despre utilizatorul curent din bază, s-a folosit clasa User_Mixin din librăria flask_login. Pentru crearea coloanelor și definirea specificațiilor și a constrângerilor acestora, s-a folosit clasa db din librăria flask_sqlalchemy. În clasa respectivă se găsesc metodele precum *Column(), String(), Integer()*, etc, și parametrii precum *primary_key, foreign_key, unique, nullable*, etc. Pentru a accesa o tabelă anume din bază trebuia mai întâi apelată în Flask funcția *query_all()* la clasa asociată tabelei. Operațiile de adăugare, ștergere, comitere și undo au fost menționate mai sus.

IV Codul sursă

Link GitHub:
https://github.com/fortunab/Proiect_PC_Stud_Dumas

V Concluzii și direcții viitoare

În săptămâna a cincea a apărut primul bug la baza de date, datorită unor greșeli de logică, astfel că datele nu s-au alocat în mod corect în bază. Tabelul care a conținut date despre utilizator și server, s-a scizionat în două tabele, având scopul ca datele despre utilizator și cele despre registrul serverului să fie reprezentate separat. Tabelele sunt conectate între ele prin relația de alăturare echivalentă (equi join), având coloana de ID a utilizatorului comună. În săptămâna a șaptea s-a rezolvat eroarea de trimitere consecutivă și neîncetată a notificărilor, prin crearea unei clase de notificări, adică a unui tip de date abstract. Această eroare a fost complet reparată prin crearea unei funcții care instanțiază un obiect de tip Notificare, în același fișier. Folosind această abordare, membrii echipei au împiedicat resetarea valorilor și, deci, transmiterea continuă a unei singure alerte. În săptămâna a opta și a noua a fost rezolvată și problema de relație între notificare și server. Totuși, alerta a fost corect creată pentru un host local, însă trebuia să se apeleze în clasa de notificare și serverul selectat. Prin acestă înfăptuire, riscul de incompatibilitate a fost eliminat.

Când vine vorba de partea de interfață grafică (site web), primele probleme au apărut la partea de stocare a datelor despre utilizator în baza de date, după înregistrare, în săptămana a șaptea. Bug-ul a fost rezolvat în așa fel încât s-a creat o clasa pentru definirea formei de înregistrare, iar ulterior prin instanțierea unui obiect de tip Utilizator cu atributele declarate în clasa de utilizator, în fișierul de vederi (views.py). Pentru ca datele personale să fie stocate în bază, în fișierul html de înregistrare (register.html), s-a folosit comanda de cerere a datelor din forma declarată anterior, folosind șablonul Jinja. Un alt bug, care a fost reparat și a apărut în săptămâna a zecea, este acela de a stoca *M* servere, propuse de un singur utilizator, într-o tabelă cu *N* servere. Ulterior, aceste servere vor fi transferate pe pagina cu servere.

Alte bug-uri care au apărut, aveau legătură cu compatibilitatea; anume, compatibilitatea dintre pagina web și server din aspectul transmiterii de notificări, sau dintre pagina web și datele în timp real pentru reprezentarea grafică.

De asemenea, de așteptat este să mai customizăm aplicația în așa fel încât ea să atragă mai mulți clienți. Am dori să o facem publică prin intermediul pachetului *pythonanywhere* și să facem un WebView în Android Studio care să creeze aplicația mobilă (disponibilă și pentru dispozitivele IOS și pentru cele Android). Lucrul respectiv ne-ar da un motiv să o postăm pe *Google Play*. Pentru început, descrierea aplicației și codul aferent ei vor fi postate pe platforma *github*.