Universitatea “Politehnica” din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Aplicație Web E-community pentru Developeri

Lucrare de disertație

Prezentată ca cerință parțială pentru obținerea titlului de Master în domeniul Electronică și Telecomunicații

programul de studii de masterat Electronică și Informatică Aplicată (EIA)

Conducător științific
Dr. Ing. Dragoş Ioan Săcăleanu

Absolvent
Anton DOLETE

2023

Anexa 2

Universitatea “Politehnica” din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul ELECTRONICA APLICATA SI INGINERIA INFORMATIEI

Aprobat Coordonator program master: prof. dr. ing. Adriana FLORESCU

PROPUNERE
TEMA LUCRARE DE DISERTAŢIE LA MASTERUL EIA
a masterandului Dolete Gh. Anton 411-EIA

1. Titlul temei: **Aplicație Web E-community pentru Developeri**

2. Contribuţia practică, originală a studentului va consta în (*în afara* părţii de documentare):

Aplicația presupune realizarea unei comunități virtuale pentru Developeri, care facilitează comunicarea între utilizatorii platformei, în legătură cu orice topic adus în discuție, de la împărtășirea cunoștințelor în rezolvarea diverselor erori, până la ultimele noutăți în domeniul tehnologiei. Platforma va permite utilizatorilor să creeze mai multe discuții în legătură cu diferite topicuri și asignarea de tag-uri subiectelor, iar alți utilizatori pot răspunde și să aprecieze conținutul. De asemenea atașarea codului sursă în corpul postării pentru îmbunătățirea citirii codului. Platforma va avea un mediu interactiv integrat pentru scrierea și execuția codului.
Pentru implementarea back-end se va folosi NodeJs, se vor implementa operațiunile de tip CRUD, cât și de abstractizarea și descrierea modelelor logice care se regăsesc în structura funcțională a aplicației. Aplicația va folosi o baza de date MongoDB iar această va fi hostata virtual în Mongo Atlas.
Pentru implementarea front-end se va folosi HTML, CSS și Javascript, prin intermediul React.js, limbaje care vor face posibilă o interfață coerentă a aplicației pentru utilizator. În momentul înregistrării, parolă aleasă de către utilizator va fi transformată folosind funcții hash având implementat algoritmi de criptare, codul hash fiind cel înregistrat în baza de date și folosit pentru autentificarea utilizatorilor înregistrați. În aplicație va fi implementat un sistem de notificări pentru mesajele necitite, fiind responsabil de trimiterea de notificări în număr masiv.

3. Proprietatea intelectuală asupra proiectului aparţine: *studentului*

4. Locul de desfăşurare a activităţii: *UPB*

5. Realizarea practică rămîne în proprietatea: *studentului*

6. Data eliberării temei:

CONDUCĂTOR LUCRARE:
Dr. Ing. Dragoș-Ioan Săcăleanu

STUDENT:
Dolete Anton

Anexa 5

Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul " Aplicație Web E-community pentru Developeri", prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de Master în domeniul domeniul Electronică și Telecomunicații, programul de studii program Electronică și Informatică Aplicată este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurătorilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, 22.12.2023

Absolvent Dolete Anton

Cuprins

Lista figuri

Introducere

Aplicația presupune realizarea unei comunități virtuale pentru Developeri, care facilitează comunicarea între utilizatorii platformei, în legătură cu orice topic adus în discuție, de la împărtășirea cunoștințelor în rezolvarea diverselor erori, până la ultimele noutăți în domeniul tehnologiei.

Scopul aplicației web bazate pe Flask „FDEV" este de a ajuta dezvoltatorii să aibă conversații. Utilizatorii pot crea subiecte în cadrul unor categorii prestabilite, pot răspunde și pot participa la discuții cu privire la o serie de probleme legate de programare și tehnologie pe acest site bine organizat.

Programul este organizat ca un forum convențional, complet cu postări, categorii, subiecte și comentarii. Identificarea utilizatorilor, gestionarea profilului și instrumentele de administrare pentru moderarea forumului sunt toate incluse.

De asemenea atașarea codului sursă în corpul postării pentru îmbunătățirea citirii codului. Platforma va avea un mediu interactiv integrat pentru scrierea și execuția codului.

Pentru implementarea back-end se va folosi Flask, se vor implementa operațiunile de tip CRUD, cât și de abstractizarea și descrierea modelelor logice care se regăsesc în structura funcțională a aplicației.

Aplicația va folosi o baza de date SQLite iar această va fi hostata local. Pentru implementarea front-end se va folosi HTML, CSS și Javascript, prin intermediul React.js, limbaje care vor face posibilă o interfață coerentă a aplicației pentru utilizator.

În momentul înregistrării, parolă aleasă de către utilizator va fi transformată folosind funcții hash având implementat algoritmi de criptare, codul hash fiind cel înregistrat în baza de date și folosit pentru autentificarea utilizatorilor înregistrați. În aplicație va fi implementat un sistem de notificări pentru mesajele necitite, fiind responsabil de trimiterea de notificări în număr masiv.

Capitol 1. INTRODUCERE

1.1 Aplicabilitatea aplicației și motivația lucrării

1.1.1 Motivația lucrării

Motivația acestei lucrări este de a oferi o imagine de ansamblu de baza a tehnologiilor software și a principiilor de codare necesare de a crea o aplicație online receptivă care poate fi ușor accesibilă de către utilizatori care folosesc orice navigator web.

Vom aminti tehnologiile utilizate pentru implementarea aplicației în prima parte a proiectului înainte de a ne concentra pe descrierea procesului de dezvoltare și a produsului software rezultat.

Proiectul va fi prezentat și dezvoltat în articol din punct de vedere academic. Este nevoie de modificări pentru a facilita conformitatea cu normele legislației (inclusiv normele GDPR1) și standardele de securitate cibernetică înainte de deploy-ul site-ului in WWW2.

1.1.2 Motivația FDEV

FDEV facilitează comunicarea între utilizatorii platformei despre tot ceea ce iese în discuție, de la cele mai noi dezvoltări tehnologice până la schimbul de sfaturi despre cum să remediați diferite tipuri de greșeli.

Utilizatorii vor putea să eticheteze subiecte și să înceapă mai multe conversații pe diverse subiecte de pe platformă. Alți utilizatori vor putea comenta și nota materialul. Pentru a face codul mai ușor de citit, am atașat și codul sursă la corpul postării. Un mediu interactiv integrat pentru crearea și rularea codului va fi disponibil pe platformă.

1.2 Structura lucrării

Aceasta lucrare este secționată în 4 capitole:

- Scopul lucrărilor și aplicației sunt descrise în introducere. Această parte oferă clarificări privind subiectul ce va fi discutat și, de asemenea, explică pe scurt selectarea acestei teme.
- Explicarea teoretică a subiectelor și instrumentelor software este tratată în secțiunea următoare. Începem cu metodologiile, ideile abordate în dezvoltarea Frontend-ului , apoi trecem la metodologiile, ideile abordate în dezvoltarea Backend-ului . Subliniem instrumentele și ideile din spatele integrării continue, automatizării aplicațiilor și punerii în funcțiune pentru a încheia această secțiune.
- Descrierea aplicației, proiectarea și implementarea acesteia sunt subiectele principale ale penultimei secțiuni.
- În partea finală , intitulată „Concluzii”, sunt discutate aspecte ale folosirii și dezvoltării lucrării practice , împreună cu potențialele beneficii și eventuale viitoare îmbunătățiri ale aplicației.

Capitol 2. PREZENTAREA TEORETICĂ A METODOLOGIILOR SI A COMPONENTELOR SOFTWARE

Scopul acestei secțiuni este de a introduce instrumentele software necesare pentru punerea in funcțiune aplicației, precum și cele care servesc drept fundație pentru dezvoltarea sa.

Dualitatea care este fundamentală pentru arhitectura unei aplicații full-stack are o influență directă asupra organizării capitolului (aplicație client-server). Ca rezultat, vom discuta mai întâi despre tehnologiile utilizate pentru a crea partea client (Front-end), apoi ne vom îndrepta atenția asupra tehnologiilor Back- end.
Utilizarea instrumentelor de integrare, automatizare, implementare și exploatarea aplicației va fi tratată în ultima parte a acestui capitol.

2.1. Modulele FRONT-END

JavaScript (ES6+) – Evoluția unui Limbaj Universal în Arhitectura Aplicațiilor Web Moderne

Utilizare în Dev Forum

Aplicația folosește JavaScript modern (ES6+) pentru toate funcționalitățile la nivel client, inclusiv gestionarea formularelor, solicitările AJAX, interacțiunile cu interfața de utilizare și comutarea modului întunecat.

Introducere

Evoluția și consolidarea unor limbaje de programare au dus la o revoluție în dezvoltarea web modernă, deoarece au permis aplicațiilor să se extindă dincolo de afișarea conținutului static. În acest context, JavaScript a devenit fundamentul interacțiunii web, deoarece este singura limbaj de programare care funcționează nativ în toate browserele moderne. JavaScript a evoluat foarte mult de la un limbaj de scripting simplificat pentru manipularea documentelor HTML. Acum este un limbaj multiparadigmă care suportă programarea orientată pe obiecte, funcțională și asincronă.

Deoarece JavaScript a fost standardizat sub umbrela ECMAScript, limbajul a evoluat rapid. De la versiunea ECMAScript 2015, sau ES6, sintactica și structurile limbajului au suferit schimbări semnificative.

Evoluția istorică și standardizarea limbajului JavaScript

JavaScript a fost dezvoltat de Brendan Eich în cadrul Netscape în 1995 și a fost conceput ca o modalitate rapidă de a adăuga interacțiune paginilor web. În ciuda perioadei scurte de dezvoltare, limbajul s-a impus rapid ca o parte esențială a platformei web, fiind integrat de majoritatea browserelor majore. A existat nevoie de un standard comun pentru JavaScript, așa că ECMA International a publicat standardele ECMAScript.

JavaScript a fost văzut mai degrabă ca un limbaj suplimentar pentru gestionarea evenimentelor de bază și manipularea documentelor DOM în primele sale versiuni. Dar cerințele crescute ale aplicațiilor web, împreună cu apariția framework-urilor puternice precum Angular, React și Vue, au dus la o reevaluare a bazelor limbajului. Odată cu lansarea sa din 2009, ES5 a pus bazele cu funcționalități precum mode strict și JSON, dar schimbarea reală a venit cu publicarea ECMAScript 2015, sau ES6.

ES6 și restructurarea sintactică a limbajului

JavaScript a fost văzut mai degrabă ca un limbaj suplimentar pentru gestionarea evenimentelor de bază și manipularea documentelor DOM în primele sale versiuni. Dar cerințele crescute ale aplicațiilor web, împreună cu apariția framework-urilor puternice precum Angular, React și Vue, au dus la o reevaluare a bazelor limbajului. Odată cu lansarea sa din 2009, ES5 a pus bazele cu funcționalități precum mode strict și JSON, dar schimbarea reală a venit cu publicarea ECMAScript 2015, sau ES6.

Introducerea funcțiilor arrow a fost una dintre cele mai importante progrese sintactice. Aceste funcții oferă o sintaxă succintă și mențin contextul lexical al cuvântului cheie this, rezolvând astfel una dintre cele mai

frecvente probleme care provoacă confuzie în versiunile anterioare. În plus, ES6 a adăugat suportul nativ pentru clase prin introducerea cuvântului cheie class. Acest lucru a permis o abordare mai orientată pe obiecte a programării, fără a elimina flexibilitatea prototipurilor pe care limbajul le susține în mod nativ.

Posibilitatea de a destructura obiecte și array-uri, o funcționalitate care simplifică semnificativ manipularea datelor complexe, a fost o altă schimbare semnificativă. De asemenea, au fost introduse șabloane de stringuri numite template literale, care permit interpolarea variabilelor și formatarea textului într-un mod mult mai expresiv.

M Controlul fluxului, modularitatea și asincronismul

Prin introducerea cuvintelor cheie import și export, ES6 a adăugat suport nativ JavaScript pentru module. Această funcție a ajutat la organizarea codului în fișiere distincte, fiecare fișier îndeplinind o funcție specifică. De asemenea, a permis încărcarea dinamică a componentelor în funcție de cerințele aplicației. Modularitatea nativă a devenit o parte esențială a arhitecturilor web moderne în contexte în care scalabilitatea și mentenabilitatea sunt esențiale.

Rafinarea modelului de execuție asincronă a fost un alt pas important în evoluția limbajului. JavaScript nu blochează execuția codului principal prin operații asincrone, deoarece folosește un model bazat pe ciclu de evenimente. Până la ES6, acest lucru a fost gestionat aproape în întregime prin callback-uri, ceea ce a dus frecvent la ceea ce se numește „infernul callback-urilor”.

Prin introducerea sintaxei async și await, versiunile ulterioare, în special ES2017, au consolidat acest model. Aceasta permite scrierea codului asincron într-o manieră liniară și ușor de urmărit. Această sintaxă a devenit rapid standardul de facto în dezvoltarea aplicațiilor web moderne, permițând integrarea ușoară a operațiilor asincrone, cum ar fi apelurile către API-uri, operațiile cu baze de date sau interacțiunile cu resurse externe.

Paradigme de programare și expresivitate modernă

Suportul pentru mai multe paradigme de programare este o caracteristică definitorie a JavaScript. Prin introducerea unor metode precum map, filter și reduce, precum și prin încurajarea utilizării funcțiilor pure și a imutabilității, limbajul a ajutat la adoptarea paradigmei funcționale din ES6. Aceste concepte provin din limbaje funcționale precum Haskell și Lisp și îmbunătățesc predictibilitatea codului și reduc probabilitatea apariției efectelor secundare nedorite.

În același timp, JavaScript a menținut și a îmbunătățit suportul pentru programarea orientată pe obiecte și a introdus class syntax și metode statice sau private, ceea ce a permis o structurare mai bună a codului în aplicații de dimensiuni mai mari. Deși moștenirea prototipului a rămas accesibilă, sintaxa a fost schimbată pentru a facilita dezvoltatorilor utilizarea tehnicilor complexe de compoziție și reutilizarea codului.

Utilizarea operatorilor avansați de evaluare a scurtcircuitelor, a funcțiilor de ordin superior și a capacității de a gestiona închideri, care sunt concepte esențiale pentru controlul fluxului logic, au sporit expresia limbajului. Scrierea unui cod mai clar, mai simplu și mai robust, în special în combinație, a fost posibilă datorită acestui nivel mai ridicat de expresivitate.

Aplicații practice, ecosistem și direcții de evoluție

Astăzi, aproape toate sectoarele industriei software folosesc JavaScript (ES6+), de la aplicații front-end create cu React, Angular sau Vue până la aplicații server-side dezvoltate cu Node. JS Sintaxa și funcționalitățile care au fost introduse începând cu ES6 constituie baza pentru framework-urile moderne, iar crearea de aplicații moderne este aproape imposibilă fără o înțelegere profundă a acestor concepte.

Platforme precum npm, care oferă acces la milioane de biblioteci și pachete reutilizabile, precum și o comunitate mare și activă susțin ecosistemul JavaScript. Instrumentele de construcție moderne, cum ar fi Webpack, Rollup și Vite, integrează modulele ES6 perfect și permit optimizarea codului pentru performanță și compatibilitate cu toate browserele majore.

În ceea ce privește progresele, standardul ECMAScript este actualizat anual. Cele mai recente idei vizează îmbunătățirea performanței, adăugarea de tipuri primitive noi și îmbunătățirea suportului pentru funcționalități reactive și funcționale. O nouă generație de aplicații web inteligente, interactive și scalabile este prevăzută de integrarea mai profundă a funcțiilor asincrone, apropierea de paradigme reactive și crearea de noi modele de concurrency, cum ar fi Atomics și SharedArrayBuffer.

Scopul JavaScript

JavaScript a fost introdus în `95, unul dintre obiectivele sale principale a fost să preia unele dintre sarcinile de validare a intrărilor care fuseseră anterior gestionate de limbaje de pe partea serverului. La acea vreme era nevoie de o „călătorie dus-întors" la back-end de a stabili dacă un anume „field", are o valoare nul sau dacă valoarea este incorectă. JavaScript folosește o caracteristică nouă pe care Netscape Navigator a introdus-o pentru a încerca să remodeleze acest lucru. Când modem-urile dial-up erau încă utilizate pe scară largă, abilitatea de a gestiona o anumită validare simplă pentru client era o nouă capacitate interesantă. Fiecare solicitare adresată serverului de atunci necesita răbdare din cauza vitezei lente asociate.[2]

JavaScript s-a dezvoltat într-o componentă crucială a fiecărui navigator disponibil astăzi. JavaScript interacționează acum cu practic fiecare element al ferestre a navigatorului și conținutul acestuia, nefolosit doar pentru validarea simplă a datelor((închiderile (closures), funcțiile anonime (lambda) și chiar meta-programarea)) sunt toate incluse în JavaScript ca sa arate capabilitatea acestuia să gestioneze calcule si decizii complexe. JavaScript s-a dezvoltat într-o componentă crucială pentru internet, acum este acceptat de alte navigatoare, inclusiv cele pentru dispozitive mobile și persoanele cu deficiențe.

Chiar și Microsoft a inclus în cele din urmă un JavaScript făcut în întregime de ei în IE începând cu prima ediție, în ciuda faptului că avea propriul limbaj de scripting numit VBScript. Evoluția JavaScript de la un limbaj bază la un limbaj mult mai evoluat nu a putut sa fie prevăzută. E necesar doar de câteva minute pentru a familiarizarea cu JavaScript, dar durează ani pentru a înțelege bine acest limbaj extrem de simplu, dar foarte complex. Este esențial să înțelegi natura și restricțiile JavaScript înainte de a începe călătoria spre utilizarea acestuia la maximul său potențial.[2]

Scurtă istorie

Pentru lansarea Netscape Navigator 2, Brendan Eich, un inginer Netscape la acea vreme, a demarat creerea un limbaj denumit LiveScript, asta în `95. Acesta a intenționat să fie folosit pe server, unde va fi cunoscut sub numele de LiveWire, precum și în navigator.

Pentru a termina LiveScript la timp, Netscape a format un parteneriat de colaborare cu Sun Microsystems. Valorificarea popularității limbajului de programare Java la acea vreme, Netscape a redenumit LiveScript în JavaScript apoi a fost lansat publicului.Datorită popularității JavaScript 1.0, a fost lansat JavaScript 1.1 Într-o perioadă în care utilizarea online era la un nivel maxim, Netscape s-a poziționat drept lider în industrie. În acest moment, Microsoft a luat decizia de a investi mai multe resurse în Internet Explorer, un browser rival. Microsoft a dezvăluit IE3 cu o dezvoltare numita JScript la scurt timp.

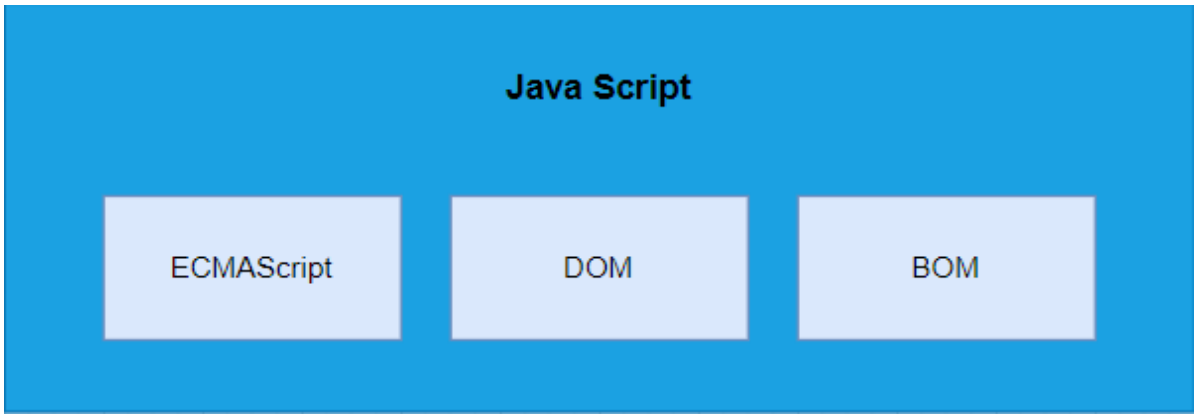
Adoptarea Microsoft a JavaScript a decurs la crearea a unei noi versiuni JavaScript si JavaScript în IE.[2]

JavaScript 1.1 a fost adus la cunoștința Asociației Europene a Producătorilor de Calculatoare (ECMA) în 1997. Pentru a „standardiza sintaxa și semantica unui limbaj de scripting neutru de tip multiplatformă", a fost înființat comitetul tehnic numărul 39 (TC39). Standardul ECMA-262, care specifică noul limbaj cunoscut sub numele de ECMAScript.

În anul următor, ECMAScript a fost acceptat și ca standard de către(ISO/IEC-16262). Din acel moment, navigatoarele web au încercat folosirea ECMAScript ca fundație de a dezvolta JavaScript.

Implementarea JavaScript

JavaScript este substanțial mai mare decât se descrie, în ciuda faptului că termenii sunt folosiți frecvent în mod interschimbabil. Următoarele trei componente separate alcătuiesc o implementare completă JavaScript Figura 1.[2]



Figură 1- Standardul ECMAScript / (DOM) / (BOM)

Standardul definit, nu are conexiune directă cu navigatorele web. O bază pentru construirea de limbaje de scripting mai puternice este stabilită în ECMA-262. Singurul mediu gazdă posibil pentru o implementare ECMAScript, este un navigator web. Sintaxa ECMAScript sunt utilizate de extensii, cum ar fi DOM , pentru a extinde funcții specifice. NodeJS fiind folosit frecvent pentru dezvoltarea back-end, este un mediu gazdă suplimentar.

Problema devine așadar, dacă ECMA-262 nu se referă la navigatoarele web, ce specifică ? La cel mai fundamental nivel, descrie următoarele componente lingvistice: ECMAScript este doar o descriere a diverselor specificații, inclusiv sintaxă, variabile, instrucțiuni, cuvinte rezervate, operatori și obiecte globale cat și cuvinte cheie. JavaScript nu este singurul limbaj care acceptă ECMAScript. Adobe ActionScript este o ilustrare diferită. [2]

Structura JavaScript în plan lexical

Setul de bază de definiții care definesc felul în care aplicațiile trebuie să fie construite în acel limbaj . O sa o numim sintaxă la nivelul său cel mai fundamental. [1]

JavaScript face parte din categoria tehnologiilor case sensitive. Ca rezultat, este imperativ să introduceți toți identificatorii, variabilele , cuvintele rezervate în mod consecvent.

Spațiile care există între elementele programului sunt ignorate de JavaScript. Alături de caracterul spațiu standard (u\0020), JavaScript înțelege pana si tab ca si spațiu.

JavaScript Comentarii

JavaScript suportă două feluri de comentarii. JavaScript consideră totul în „//” și capătul unui rând ca fiind un comentariu și îl ignoră. Un comentariu este definit ca orice text între literele „/ *” și „* /”, care se poate extinde pe o porțiune mai mare, dar nu și imbricat.

JavaScript Identificatorii și setul de cuvinte rezervate

Simplu spus, identificatorii sunt nume. Identificatorii sunt folosiți în JavaScript pentru a da nume constantelor, variabilelor, proprietăților, funcțiilor și claselor, precum și pentru a oferi etichete codului în interiorul buclelor. Un identificator JavaScript începe cu o literă și acest lucru este strict, caracterul(_), sau($). Literele, cifrele, caracterul underscore(_), caracterul dolar($) sunt acceptate ca caractere ulterioare.

JavaScript cuvintele rezervate

Ca și orice alt limbaj, JavaScript folosește doar un anumit set de identificatori pentru propriile scopuri. Acești „termeni rezervați” nu pot fi utilizați ca identificatori obișnuiți(de exemplu identificarea variabilelor sau a constantelor).

Limbajul JavaScript include termenii de mai jos. Mulți dintre aceștia sunt termeni rezervați și nu ar trebui folosiți ca nume pentru constante, variabile, funcții sau clase (precum if, while și for) deși toți pot fi folosiți ca nume de proprietăți într-un obiect. Altele, inclusiv from, of, get și set, sunt folosite cu moderație și fără ambiguitate

gramaticală ca identificatori și sunt în întregime valide. Alte cuvinte cheie, ca si let, nu se pot rezerva în totalitate pentru păstrarea compatibilității anterioară, prin urmare există restricții complicate care dictează când pot și nu pot fi utilizate ca identificatori. (De exemplu, dacă let este definit var în exteriorul clase, atunci se poate folosii ca identificator; dar, în cazul in care let este definit ca const în interiorul clasei, nu se poate.) Toți acești termeni, exceptând of ,set, target si from, care se pot utiliza și sunt utilizate în prezent pe scară largă, nu ar trebui să fie utilizate ca identificatori. [1]

Unicode

Orice caracter Unicode poate fi folosit în șiruri și comentarii, deoarece aplicațiile JavaScript sunt create folosind setul de caractere Unicode. Este o practică obișnuită să se limiteze ID-urile la caractere și cifre ASCII pentru portabilitate și simplitate a editării. Limbajul permite doar litere, cifre și caractere Unicode ca identificatori; aceasta este pur și simplu o convenție. Aceasta implică faptul că, în timp ce declară constante și variabile, programatorii pot folosi caractere și expresii matematice provenind din alte limbaje decât engleza.
JavaScript este limbajul folosit la crearea unei pagini interactive, fiind o componentă crucială a unei aplicații. JavaScript este una dintre tehnologiile web fundamentale, împreună cu HTML si CSS [1] .

Exemplu

```
// Dark mode toggle
const darkModeToggle :HTMLElement = document.getElementById( elementId: 'dark-mode-toggle');
if (darkModeToggle) {
    darkModeToggle.addEventListener( type: 'click', listener: function() :void {
        document.body.classList.toggle( token: 'dark-mode');
        const isDarkMode :boolean = document.body.classList.contains('dark-mode');
        localStorage.setItem('darkMode', isDarkMode);

        // Update icon
        const icon :HTMLElement = darkModeToggle.querySelector( selectors: 'i');
        if (isDarkMode) {
            icon.classList.remove( tokens: 'bi-moon');
            icon.classList.add('bi-sun');
        } else {
            icon.classList.remove( tokens: 'bi-sun');
            icon.classList.add('bi-moon');
        }
    });

    // Check for saved dark mode preference
    const savedDarkMode :string = localStorage.getItem( key: 'darkMode');
    if (savedDarkMode === 'true') {
        document.body.classList.add('dark-mode');
        const icon :HTMLElement = darkModeToggle.querySelector( selectors: 'i');
        icon.classList.remove( tokens: 'bi-moon');
        icon.classList.add('bi-sun');
    }
}
```

Figură 2- JavaScript Darkmode

CSS3 – Fundament al Designului Web

Limbaje de marcare precum HTML, cum ar fi CSS, este utilizat la specificarea părții vizuale a unui document. Alături de HTML și JavaScript, CSS este trio-ul WWW.

Utilizare în Dev Forum

Aplicația folosește CSS3 pentru toate stilurile, inclusiv componente personalizate, design receptiv, animații și suport pentru modul întunecat.

Introducere

Evoluția dezvoltării web a fost marcată de creșterea complexității aplicațiilor, precum și de nevoia de a oferi experiențe vizuale cât mai plăcute și coerente pentru utilizator. Într-un domeniu digital din ce în ce mai competitiv, aspectul și funcționalitatea unei interfețe grafice sunt acum cruciale pentru percepția calității unui produs software. Prescurtarea pentru Cascading Style Sheets (CSS) este componenta care se ocupă de aspectul și prezentarea vizuală a unui document web, în timp ce HTML oferă scheletul structural. Cea mai recentă versiune stabilă a limbajului de stilizare, CSS3, a schimbat complet modul în care sunt create și animate interfețele utilizatorului.

Scopul acestei lucrări este de a examina rolul CSS3 în arhitectura web modernă, de a examina progresele tehnologice aduse de această versiune și de a analiza efectele sale asupra creării aplicațiilor responsive, dinamice și accesibile.

Context istoric și evoluția standardului CSS

Ca răspuns la limitările majore ale HTML în ceea ce privește controlul stilistic, CSS a fost introdus oficial în 1996. Pentru că nu exista un mecanism distinct pentru design, dezvoltatorii foloseau etichete HTML precum atributele inline sau pentru a modifica culoarea, dimensiunea sau stilul textului. Acest lucru a dus la documente greu de citit și vizual incoerente. Un model de stiluri în cascadă care a fost introdus în prima versiune CSS a făcut posibilă separarea logicii de prezentare și facilitează reutilizarea și centralizarea definițiilor stilistice. Suportul pentru căutările de media, poziționarea relativă, stilurile de tabele și regulile de selecție mai complexe au fost adăugate limbii odată cu lansarea versiunii CSS2. Pentru o perioadă considerabilă de timp, standardul a fost limitat în ceea ce privește tranzițiile, animațiile și designul adaptiv.

Consorțiul W3C a creat CSS3 ca un set modular de specificații, mai degrabă decât o actualizare liniară. Acest lucru a făcut posibil ca fiecare modul să fie publicat și actualizat individual. Această metodă a redus fragmentarea și a facilita adoptarea la scară largă a noilor funcționalități prin adoptarea treptată și testarea acestora în implementările browserelor. Prin urmare, CSS3 a contribuit la stabilirea practicilor moderne de design, coerență tipografică și interactivitate fluidă și a devenit standardul care guvernează astăzi crearea aspectului vizual al aplicațiilor web.

Structura modulară și inovațiile aduse de CSS3

Deoarece CSS3 are o arhitectură modulară, utilizatorii pot crea și implementa moduri personalizate pentru diferite elemente de stilizare, cum ar fi layout-ul, animațiile, culorile și fonturile. Această structură ajută la extinderea progresivă a limbajului, permițând fiecărui modul să se îmbunătățească în mod independent în funcție de evoluțiile tehnologice și de nevoile comunității. Introducerea opțiunilor avansate, cum ar fi cele de tip nth-child sau nth-of-type, care permit stilizarea contextuală a elementelor fără a modifica codul HTML, este una dintre cele mai importante modificări ale CSS3.

În plus, CSS3 a introdus suport nativ pentru tranziții și transformări. Acest lucru permite utilizatorilor să animeze proprietățile CSS fără a utiliza cod JavaScript. Acest lucru a avut un impact semnificativ asupra performanței și accesibilității, permițând dezvoltatorilor să creeze efecte vizuale coerente și fluide, care funcționează bine pe o varietate de dispozitive. Modulele de transformare 2D și 3D permit elementelor să se rotească, să se scaleze sau să se mute într-un mod elegant, fără a necesita grafică complicată sau pluginuri externe.

În același timp, adăugarea proprietății de tranziție a oferit o modalitate declarativă de a aplica efectele de animație la schimbarea stării unui element. Acest lucru a crescut interactivitatea fără a crește complexitatea codului.

Prin introducerea modulului Flexbox și a CSS Grid, flexibilitatea modului de afișare a layout-urilor a fost un alt domeniu de inovație. Aceste sisteme de layout moderne sunt mult mai simple de utilizat decât modelele tradiționale bazate pe float sau table pentru alinierea și distribuirea elementelor pe orizontală și verticală. Acest lucru reduce nevoia de hack-uri și permite un control precis asupra comportamentului elementelor în funcție de dimensiunea containerului și spațiul disponibil.

Design responsive și adaptabilitate în CSS3 CSS3

Capacitatea de a crea interfețe responsive care se adaptează automat la dimensiunea și caracteristicile dispozitivului utilizatorului este printre cele mai mari progrese aduse de CSS3. Dezvoltatorii pot crea stiluri personalizate pentru diferite dimensiuni ale ecranului folosind media queries fără a modifica structura HTML sau a crea cod duplicat. Într-o eră dominată de mobilitate, în care aplicațiile trebuie să fie atât funcționale, cât și elegante pe ecrane de dimensiuni variate, de la telefoane inteligente până la monitoare 4K, această caracteristică a devenit esențială.

În plus, CSS3 facilitează crearea de interfețe accesibile și eficiente din punct de vedere al resurselor. Indiferent de contextul de afișare, utilizarea unităților relative precum rem, em, vw și vh ajută la scalabilitatea interfeței și la menținerea proporțiilor între componentele vizuale. În plus, caracteristicile moderne precum potrivirea obiectului, aspect-ratio sau prindere permit controlul ușor al tipografiei și al conținutului multimedia fără a recurge la scripting complicat sau la soluții de tip fallback.

Principia designului mobile-first, care presupune că aplicațiile sunt concepute inițial pentru ecrane mici și apoi cresc pentru rezoluții mai mari, a fost extinsă cu CSS3, combinând aceste caracteristici cu sisteme de layout moderne. Această metodă ajută la crearea de aplicații mai rapide, mai eficiente și mai concentrate pe esențial, concentrându-se pe experiența utilizatorului.

Tipografie, culori și micro-interacțiuni CSS3

Utilizarea suportului pentru fonturi personalizate și a tehnologiei @font-face a schimbat modul în care sunt gestionate stilurile tipografice în CSS3. Acum, datorită acestei inovații, dezvoltatorii pot folosi fonturi personalizate care reflectă identitatea vizuală a brandului, în loc să folosească fonturi sigure pentru web care sunt limitate ca expresivitate. Prin urmare, site-urile web moderne oferă o experiență tipografică sofisticată și continuă, fără a sacrifica performanța sau compatibilitatea.

În același timp, gama de culori disponibilă în CSS3 a fost extinsă pentru a include suport pentru transparență, valori RGBA și HSLA. Acest lucru a crescut flexibilitatea în designul vizual. Animațiile subtile și micro-interacțiunile, cum ar fi efectele hover, feedback-ul la clicuri și evidențierea dinamică a elementelor, pot fi implementate cu ușurință prin CSS3, ceea ce contribuie la o experiență de utilizare interactivă și plăcută.

Această expresivitate vizuală îmbunătățește nu numai aspectul aplicației, ci și funcționalitatea, deoarece feedback-ul vizual face navigarea mai ușoară, reduce erorile de interacțiune și încurajează utilizatorul să aibă încredere în produs.

Provocări, compatibilitate și bune practici CSS3

Dezvoltatorii trebuie să fie conștienți de diferitele moduri de implementare ale CSS3, în special în ceea ce privește modulele opționale sau experimentale, deoarece CSS3 este popular și este susținut de majoritatea browserelor moderne. Există posibilitatea ca anumite funcționalități să necesite prefixe specifice pentru anumite browsere sau teste suplimentare pentru a se asigura că comportamentul este uniform pe toate platformele. În aceste circumstanțe, utilizarea unor instrumente precum Autoprefixer, testarea în mai multe medii și respectarea standardelor de îmbunătățire progresivă sunt esențiale pentru a produce un produs de calitate.

În plus, puterea declarativă a CSS3 îl poate face greu de menținut în proiecte mari dacă nu este structurat corect. Stilul clar, scalabil și reutilizabil poate fi asigurat prin organizarea codului în module, utilizarea

convențiilor de denumire precum BEM și utilizarea preprocesoarelor CSS precum SASS sau LESS.

Provocări, compatibilitate și bune practici CSS3

Prin faptul că a schimbat complet modul în care sunt construite, stilizate și animate interfețele aplicațiilor moderne, CSS3 a marcat o revoluție discretă în dezvoltarea web. În afară de a fi doar un limbaj de stilizare, CSS3 este o parte esențială a experienței utilizatorului, având un impact asupra performanței, accesibilitatei și eficienței aplicațiilor.

CSS3 este un instrument esențial pentru orice dezvoltator sau designer care încearcă să construiască aplicații web moderne, coerente și centrate pe utilizator, datorită arhitecturii sale modulare, suportului pentru layout-uri responsive, tipografiei avansate, animațiilor fluide și interacțiunilor sofisticate. În viitor, diferențierea digitală va depinde din ce în ce mai mult de calitatea experienței vizuale, CSS3 va fi o competență esențială și o tehnologie de bază pentru orice proiect web de succes.

Framework-uri CSS

Între anii 2006–2007, resursele CSS precum Blueprint și Yahoo s-au utilizat pe scară largă. Au adus cu ei mai multe instrumente esențiale, inclusiv butoane, grile, efecte de animație, fonturi și resetarea CSS. Adoptarea acestor resurse de către dezvoltatori are potențialul de a reduce semnificativ timpul necesar dezvoltării site-urilor web prin gestionarea multor activități repetitive și laborioase. O generație de resurse frontale „full-edge”, inclusiv Bootstrap, care a inclus JavaScript în implementarea sa, a venit după aceste resurse CSS fundamentale.[5]

Exemplu

```css
/* Custom styles for Dev Forum */
/* General styles */
body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background-color: #f8f9fa;
    color: #333;
}
/* Card customization */
.card {
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    border: none;
    margin-bottom: 20px;
}
/* Animations */
.fade-in {
    animation: fadeIn 0.5s;
}
@keyframes fadeIn {
    from { opacity: 0; }
    to { opacity: 1; }
}
/* Dark mode support */
@media (prefers-color-scheme: dark) {
    body.dark-mode {
        background-color: #222;
        color: #f8f9fa;
    }
    body.dark-mode .card {
        background-color: #333;
        color: #f8f9fa;
    }
}
```

Figură 3 - Exemplu CSS3

Bootstrap 5 – Framework-ul Front-End în Arhitectura Interfețelor Web Moderne

Utilizare în Dev Forum

Apariția structurilor front-end a marcat un punct de cotitură în modul în care sunt construite aplicațiile moderne. Acest lucru s-a întâmplat într-o eră digitală caracterizată de cerințe crescânde privind designul interfețelor web și de o presiune constantă asupra timpilor de dezvoltare. Aceste instrumente au făcut posibilă crearea de interfețe coerente, responsive și compatibile cu mai multe platforme, fără a necesita o cantitate semnificativă de abilități de programare. Bootstrap este unul dintre cele mai cunoscute și utilizate framework-uri din acest domeniu. A fost creat inițial de echipa Twitter și a fost apoi adoptat de o comunitate largă de dezvoltatori din întreaga lume.

Lansarea variantei Bootstrap 5 a marcat un pas major în evoluția acestui cadru, aducând o serie de modificări conceptuale, tehnice și funcționale care au adaptat instrumentul la cerințele pieței digitale actuale. Scopul acestei lucrări este de a examina în detaliu arhitectura, beneficiile, dezavantajele și aplicațiile Bootstrap 5, subliniind rolul său în ecosistemul actual de dezvoltare web.

Evoluția Bootstrap și contextul dezvoltării sale

Bootstrap a fost dezvoltat de Mark Otto și Jacob Thornton în 2011 pentru a oferi un cadru de lucru intern pentru dezvoltarea unitară a aplicațiilor web pentru Twitter. A standardiza instrumentele de dezvoltare front-end și a oferi un set coerent de componente reutilizabile a fost scopul acestui proiect. Bootstrap, care a fost lansat inițial sub numele de Twitter Blueprint, a fost un proiect open-source și a devenit rapid popular datorită modului în care este ușor de utilizat, a documentației clare și a faptului că funcționează cu multe dispozitive mobile.

În primele versiuni ale framework-ului, biblioteca jQuery și ideile CSS3 au fost foarte importante, oferind componente predefinite precum butoane, meniuri, formulare și alerte. Bootstrap a fost modificat pentru a satisface cerințele crescânde ale dezvoltatorilor cu fiecare versiune majoră. Acest lucru s-a întâmplat prin adăugarea de noi funcționalități, cum ar fi sistemul de grid flexibil, suportul pentru design responsive și posibilitatea de personalizare prin variabile SCSS. Bootstrap 5, care a fost lansat oficial în 2021, a făcut o diferență semnificativă de versiunea anterioară prin eliminarea dependenței de jQuery, revizuirea completă a sistemului de layout și integrarea unei cantități mai mari de clase utilitare. Acest lucru a adus Bootstrap 5 la standardele moderne de dezvoltare front-end.

Arhitectura și componentele fundamentale ale Bootstrap 5

Bootstrap 5 pune accent pe performanță, accesibilitate și claritate semantică, bazându-se pe principiile modularității, extensibilității și compatibilității cross-browser. Arhitectura framework-ului este formată din trei straturi principale: stiluri CSS, componente JavaScript și o rețea de clase utilitare. Toate acestea sunt concepute pentru a oferi utilizatorilor flexibilitate în personalizarea interfeței.

Bootstrap folosește un sistem bazat pe variabile SCSS la nivelul CSS, ceea ce permite dezvoltatorilor să modifice cu ușurință aspectele generale ale unei aplicații, cum ar fi paleta de culori, marginile, tipografia și alte caracteristici esențiale. Deoarece reduce riscul de conflicte stilistice și încurajează reutilizarea codului, această abordare modulară separă stilurile de bază de cele personalizate.

Una dintre cele mai bune caracteristici ale Bootstrap este sistemul de grid, care permite organizarea conținutului într-o structură adaptabilă și adaptabilă la diferite rezoluții. Bootstrap 5 folosește un model bazat pe coloane care se poate ajusta dinamic în funcție de dimensiunea ecranului. Acest lucru vă permite să controlați cu acuratețe modul în care elementele sunt afișate pe desktop, tabletă sau telefonul mobil. Acest sistem joacă un rol important în promovarea principiului mobile-first, care a devenit un standard în proiectarea interfețelor moderne.

Bootstrap 5 oferă componente interactive precum modals, dropdowns, tooltips și carousels, care sunt implementate nativ fără a utiliza biblioteci externe. Această schimbare a fost făcută pentru a îmbunătăți performanța și dimensiunea fișierelor încărcate, ambele esențiale pentru optimizarea aplicațiilor pentru viteza de încărcare și mobilitate.

Personalizare, utilitare și fluxuri de lucru moderne

Bootstrap 5 are flexibilitatea de a fi adaptat nevoilor specifice ale fiecărui proiect, ceea ce este un avantaj. Prin utilizarea preprocesorului SCSS, variabilele globale pot fi modificate rapid. Acest lucru permite crearea de teme personalizate care se potrivesc identității vizuale a mărcii sau cerințelor estetice ale clientului. În plus, Bootstrap 5 extinde considerabil clasele utilitare, promovând o abordare utilitară a stilizării. Aceste clase vă permit să controlați proprietăți precum spațierea, alinierea, dimensiunile și culorile imediat, fără a trebui să creați alte clase CSS.

Întrucât modificările pot fi testate și implementate direct în markup-ul HTML, acest model utilitar permite prototiparea rapidă și dezvoltarea iterativă. În plus, Bootstrap funcționează bine cu instrumente de construcție moderne precum Webpack și Vite, oferind control total asupra optimizării resurselor și modularizării codului pentru proiecte de dimensiuni mari. În plus, performanța aplicației în producție poate fi îmbunătățită prin eliminarea componentelor și claselor redundante și neutilizate.

Responsivitate, accesibilitate și bune practici

Bootstrap 5 este compatibil cu principiile designului responsive, ceea ce permite crearea de interfețe care se adaptează automat la dimensiunea și orientarea ecranului. Framework-ul oferă breakpoints predefinite, care pot fi utilizate pentru a defini diferite comportamente ale componentelor în funcție de contextul de afișare. Deoarece sunt conforme cu standardele industriei, aceste breakpoints pot fi extinse sau modificate în funcție de nevoile aplicației, oferind o experiență uniformă pe toate dispozitivele, de la telefoane mobile la monitoare ultra-wide.

Respectarea principiilor de accesibilitate este un element esențial al dezvoltării moderne. Majoritatea componentelor Bootstrap 5 conțin elemente semantice și caracteristici ARIA, ceea ce facilitează navigarea prin tastatură și facilitează interpretarea corectă de către cititorii de ecran. Documentația oficială oferă instrucțiuni precise cu privire la utilizarea acestor caracteristici și încurajează practicile care sunt incluzive și respectă standardele internaționale precum WCAG (Guidelines for Web Content Accessibility).
În plus, menținerea unui cod curat, scalabil și ușor de întreținut este ajutată de aplicarea unor bune practici de organizare a codului, evitarea abuzului de clase utility și utilizarea componentelor într-un mod semnificativ semantic. Bootstrap 5 este o filozofie de dezvoltare care susține sustenabilitatea în timp, modularitatea și coerența produselor digitale.

Studii de caz și utilizare practică

Bootstrap 5 este folosit atât în proiecte comerciale, cât și în aplicații academice sau instituționale datorită echilibrului său între simplitate și putere expresivă. Framework-ul poate fi utilizat pentru a crea rapid o interfață coerentă, cu formulare bine structurate, tabele responsive și navigație eficientă pentru aplicații de afaceri, cum ar fi un sistem intern de gestiune a proiectelor. Bootstrap 5 poate fi integrat cu biblioteci JavaScript precum Chart.js sau Axios pentru a crea un ecosistem complet care susține dezvoltarea de aplicații full-stack, fără a afecta viteza de livrare sau calitatea experienței utilizatorului.

Deoarece este ușor de învățat și are o documentație bine structurată, Bootstrap este adesea folosit ca unealtă introductivă în cursurile de front-end development. Aceasta le permite studenților să se concentreze pe conceptele fundamentale ale dezvoltării web și în același timp să învețe despre fundamentele organizării logice a unei interfețe și ale unui design responsive.

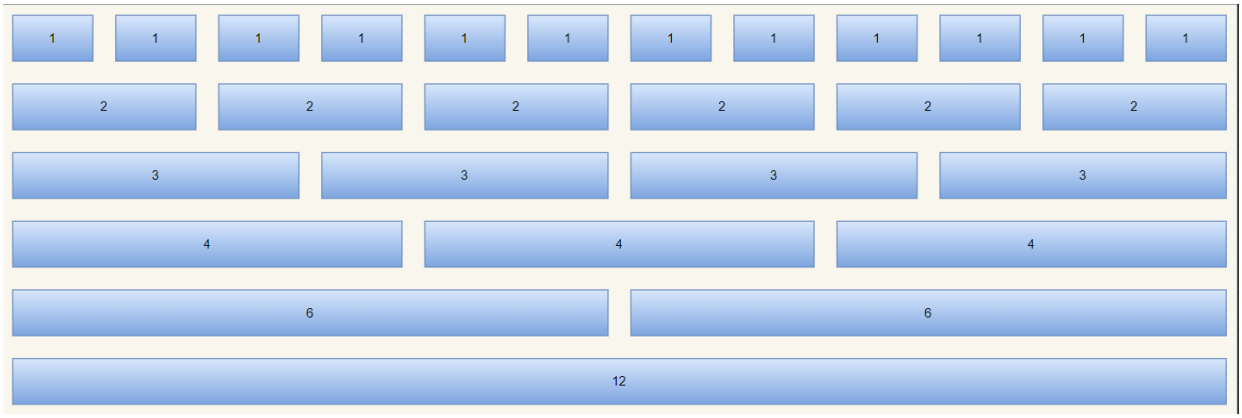
Bootstrap v1.0, care a inclus doar elemente CSS și HTML, a fost lansat în 2011. Nu există plugin-uri JavaScript în versiunea v1.3 a Bootstrap, care funcționează cu InternetExplorer7 și InternetExplorer8. Bootstrap v2.0 a fost o altă schimbare semnificativă în 2012. Biblioteca Bootstrap a fost complet rescrisă înainte de a fi transformată într-un cadru receptiv. Toate componentele sale au fost compatibile cu desktop-uri, tablete și telefoane mobile. De asemenea, au fost incluse numeroase pachete CSS și JavaScript noi. O altă lansare a

Bootstrap 3 a transformat cadrul „Mobile First and Always Responsive” după 15 ajustări. În iterațiile anterioare ale cadrului, utilizatorii au avut opțiunea de a crea un site web care a fost ușor de utilizat. În versiunea 2013, structura de directoare a proiectului și numele claselor au fost modificate.

Cu toate acestea, Bootstrap v3.o nefiind compatibil cu versiunea precedenta, în sensul că fișierele CSS și JavaScript primare nu pot fi actualizate direct la această versiune[3] . Bootstrap, ajuns la a patra ediție, continuă să fie unul dintre cele mai populare resurse în rândul dezvoltatorilor, mai ales pentru că este open source, funcționează cu o multitudine de navigatoare web, reduce timpul de dezvoltare și se poate modifica. Dar arhitectura grilă a resursei este, fără îndoială, componenta sa cea mai avantajoasă.

Cele mai receptive sisteme „grilă” create pentru dispozitive cu o rezoluție mai mica, se găsește în Bootstrap. Prin împărțirea logică a ecranului în 12 coloane, este mai ușor să scalați un document web pentru toate tipurile de rezoluție. Ca rezultat, programatorul poate alege cât de mult din suprafața afișajului ar trebui să ocupe fiecare element de design. [6]

Cele 12 coloane din sistemul standard de grilă Bootstrap (vezi Figura 4) oferă un container lățime de 940 px fără utilizarea caracteristicilor receptive. Grila se adaptează la 724/ 1170px cu includerea fișierului CSS responsive. Coloanele devin fluide și se stivuiesc vertical pentru afișaje mai mici de 767 px, cum ar fi cele de pe tablete și alte dispozitive portabile.
Figura 4 ilustrează modul în care piesele pot fi aranjate pe un layer de 12. [7]



Figură 4 - Boostrap Layer

Crearea meniurilor si a efectelor sunt doar câteva dintre nevoile fundamentale de dezvoltare care sunt îndeplinite de Bootstrap, care combină componente CSS și JavaScript utilizate pe scară largă. Pe lângă faptul că include o varietate de elemente utile care sunt ușor de utilizat în proiectarea de site-uri web, Bootstrap folosește și limbajul HTML standard. Dezvoltatorii trebuie pur și simplu să se concentreze pe generarea de marcaje HTML adecvate utilizând Bootstrap, astfel încât cadrul să îl poată înțelege și să genereze site-ul web așa cum este prevăzut.[3]

De-a lungul anilor, Bootstrap a devenit un instrument foarte popular pentru aplicațiile front-end. Este o componentă crucială pentru fiecare proiect modern datorită simplității utilizării, interoperabilității între navigatoare, suportului pentru interfețele dispozitivelor mobile și capacității de a crea aplicații web receptive.

Exemplu

```
<div class="container">
    <div class="row">
        <div class="col-md-8">
            <div class="card">
                <div class="card-header">
                    <h5 class="card-title">{{ topic.title }}</h5>
                </div>
                <div class="card-body">
                    <p class="card-text">{{ topic.description }}</p>
                    <a href="{{ url_for('topic', topic_id=topic.id) }}" class="btn btn-primary">View Topic</a>
                </div>
                <div class="card-footer text-muted">
                    Created {{ topic.created_at|time_since }}
                </div>
            </div>
        </div>
        <div class="col-md-4">
            <div class="list-group">
                <a href="#" class="list-group-item list-group-item-action active">Recent Topics</a>
                {% for topic in recent_topics() %}
                <a href="{{ url_for('topic', topic_id=topic.id) }}" class="list-group-item list-group-item-action">
                    {{ topic.title }}
                </a>
                {% endfor %}
            </div>
        </div>
    </div>
</div>
```

HTML5 – Evoluție, Structură și Impact în Dezvoltarea Aplicațiilor Web Moderne

HyperText Markup Language, sau HTML, îndeplinește două funcții cruciale: definește semantica paginilor web și specifică cum ar trebui să apară. [8]

Utilizare în Dev Forum

Aplicația folosește HTML5 pentru toate șabloanele de pagină, utilizând elemente semantice precum <header>, <nav>, <main>, <section> și <footer> pentru a crea un document bine structurat.

Introducere

Web-ul a devenit o platformă universală pentru comunicare, colaborare, comerț și educație într-un mediu tehnologic dinamic, caracterizat de interconectivitate și acces continuu la informație. Limbajul HTML, o prescurtare de la HyperText Markup Language, se află la baza acestei infrastructuri digitale globale, care oferă cadrul fundamental pentru structurarea și prezentarea conținutului pe internet. De-a lungul timpului, HTML a suferit o serie de revizuiri și standardizări pentru a satisface nevoile tot mai complexe ale utilizatorilor și dezvoltatorilor. Lansarea HTML5 a marcat un punct culminant în dezvoltarea web, deoarece a inclus elemente noi de structură și semantică, precum și mai multe capacități pentru interactivitate, multimedia și aplicații complexe.

Scopul acestei lucrări este de a analiza în profunzime arhitectura și funcționalitățile HTML5, în raport cu versiunile anterioare, și de a evalua impactul său asupra ecosistemului web contemporan.

Context istoric și necesitatea unei noi versiuni a limbajului HTML

La începutul anilor 1990, Tim Berners-Lee a dezvoltat HTML ca un limbaj simplu pentru a distribui și conecta hipertexte în rețea emergentă numită World Wide Web. O structură simplă a fost prezentă în primele versiuni ale limbajului, care s-au concentrat doar pe crearea de conexiuni între documente și formatarea textului. În timp ce numărul de utilizatori a crescut exponențial, a devenit evident că era necesar un set de specificații standardizate. Acest lucru a dus la înființarea consorțiului W3C și la crearea primelor recomandări oficiale pentru HTML.

Începând cu HTML 4.01, versiunea următoare a limbajului a adus îmbunătățiri semnificative la formele, tabelele și structurile generale ale documentelor. Cu toate acestea, HTML 4.01 nu a reușit să satisfacă cerințele crescute ale aplicațiilor web interactive, așa că dezvoltatorii au început să folosească tehnologii suplimentare precum Adobe Flash, Silverlight și Java Applets pentru a integra multimedia, animații și funcționalități complexe în paginile web. Deși eficiente pe termen scurt, aceste soluții externe au provocat probleme semnificative cu privire la securitate, performanță și compatibilitate. În acest context, a apărut nevoia unei versiuni de HTML care să poată integra funcționalitățile moderne ale nativului. În consecință, W3C a început să construiască HTML5 împreună cu WHATWG (Web Hypertext Application Technology Working Group).

Funcționalități multimedia și integrarea API-urilor native

Obiectivul principal al HTML5 a fost eliminarea dependenței de pluginuri externe pentru redarea conținutului multimedia. Elementele audio și video au fost introduse pentru a permite redarea fișierelor media direct în browser fără a instala module suplimentare. Aceste taguri sunt echipate cu atribute care controlează redarea, cum ar fi autoplay, loop, controluri sau muted. Aceste atribute pot fi modificate prin JavaScript pentru a crea playere multimedia avansate. În plus, HTML5 permite specificarea mai multor surse pentru un fișier media, asigurând compatibilitatea cu diferite formate și browsere.

În plus față de multimedia, HTML5 oferă o gamă largă de API-uri care îmbunătățesc nativ capabilitățile browserului. API-ul pentru Canvas este unul dintre cele mai importante, deoarece permite crearea de grafice bidimensionale direct pe pagină, ceea ce este util pentru instrumente vizuale, grafice interactive și jocuri. API-ul pentru geolocalizare permite aplicațiilor să identifice poziția geografică a utilizatorului, ceea ce permite aplicațiilor interactive și mobile să se adapteze contextului fizic al utilizatorului. Web Storage, care oferă o alternativă locală la cookies pentru stocarea datelor, și WebSockets, care permit comunicarea bidirecțională în timp real între client și server, sunt alte API-uri relevante.

Formulare avansate și validare nativă

Formularele web, care sunt principalul mecanism de comunicare între utilizator și aplicație, sunt un alt domeniu în care HTML5 a adus îmbunătățiri semnificative. În edițiile anterioare, gestionarea formularelor implica scrierea unui număr mare de cod JavaScript pentru a oferi o experiență interactive și pentru a valida datele. HTML5 reduce semnificativ această complexitate prin introducerea unor noi atribute și tipuri de input. De exemplu, tipurile de email, adresele URL, datele, intervalul sau culorile permit validarea automată a datelor utilizatorului fără scripturi suplimentare. În plus, atributele precum necesar, pattern sau lungime permit dezvoltatorilor să scrie reguli clare de validare în codul HTML.

Utilizând feedback imediat și scurtând timpul de încărcare a paginii, această validare nativă îmbunătățește experiența utilizatorului. În plus, datorită faptului că sunt compatibile cu tehnologii asistive și pot fi interpretate corect de browserele moderne care respectă standardele W3C, aceste funcționalități îmbunătățesc accesibilitatea aplicației.

HTML5 în contextul dezvoltării responsive și mobile-first

Deoarece utilizarea dispozitivelor mobile a depășit desktop-ul tradițional, a apărut nevoia unei abordări noi pentru crearea aplicațiilor web care se concentrează pe conceptul de design adaptabil. În tandem cu CSS3 și JavaScript, HTML5 oferă bazele necesare pentru crearea unor interfețe care se adaptează automat la dimensiunea ecranului și la capacitățile dispozitivului. Caracteristica viewport, care este descrisă în secțiunea "head" a unui document HTML5, permite conținutului să fie scalat corespunzător pe diferite dimensiuni de ecran. Acest atribut este esențial pentru crearea aplicațiilor mobile-first.

În plus, HTML5 facilitează integrarea aplicațiilor web progresive, cunoscute și sub numele de PWA. PWA combină beneficiile aplicațiilor web și native, oferind posibilitatea de a rula offline, de a primi notificări push și de a instala aplicația pe ecranul principal al dispozitivului. Aceste abilități fac din HTML5 un element esențial în ecosistemul aplicațiilor moderne, reducând diferențele tradiționale dintre aplicațiile native și cele accesate prin browser.

WWW si HTML

Înainte de anii 1990, accesul la informații prin intermediul Internetului a fost o provocare tehnică semnificativă. De fapt, chiar și cei mai inteligenți oameni și participanții la diferite activități academice au avut probleme în timp ce încercau să împărtășească date. Tim Berners-Lee a inventat legăturile hipertext pentru a accesa rapid textul pe Internet. Deși nu era o idee nouă, a reușit datorită ușurință de utilizare a HTML atunci când eforturile de hypertext mai complexe au eșuat.

Hypertext se referea inițial la text stocat electronic cu legături interne între pagini. Aproape orice articol (fișiere , text, fotografii, etc.) care poate face legătura cu alte lucruri este acum menționat cu acest nume mai mare. Organizarea și legarea de text, imagini și alte fișiere care conțin informații sunt descrise folosind limbajul de marcare hipertext. [9]

Doar aproximativ 100 de calculatoare erau capabile să deservească site-uri web HTML până în anul `93. World Wide Web (WWW), care constă din aceste pagini conectate, a inspirat crearea unui număr de navigatoare web care permit utilizatorilor să privească un document. Din cauza popularității în creștere a WWW, mai mulți dezvoltatori au creat navigatoare care ar putea afișa atât text, cât și imagini.

Concluzii

Reconfigurând complet modul în care sunt create și livrate aplicațiile web, HTML5 marchează un punct culminant în evoluția tehnologiilor web. HTML5 oferă o platformă robustă și flexibilă pentru crearea aplicațiilor web actuale și viitoare, prin introducerea unor concepte semantice clare, a capabilităților multimedia native, a unui set extins de API-uri și a suportului avansat pentru interactivitate și mobilitate. HTML5 se impune ca standard care unifică, simplifică și îmbunătățește experiența de dezvoltare și utilizare într-un mediu caracterizat de o mare diversitate tehnologică și cerințe ridicate ale utilizatorilor. Capacitatea sa de a rămâne compatibil cu versiunea anterioară asigură o tranziție fluidă între paradigmele web tradiționale și cele emergente, ceea ce îl face și mai relevant.

Exemple

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Dev Forum{% endblock %}</title>
    <!-- CSS and JavaScript includes -->
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <!-- Navigation content -->
    </nav>

    <div class="container mt-4">
        {% block content %}{% endblock %}
    </div>

    <footer class="bg-dark text-white mt-5 py-4">
        <!-- Footer content -->
    </footer>

    <!-- JavaScript includes -->
</body>
</html>
```

Figură 5 - Exemplu HTML5

2.2. Modulele BACK-END

Framework-ul Flask – Fundament pentru Dezvoltarea Aplicațiilor Web cu Python

Utilizare în Dev Forum

Aplicația folosește Flask ca cadru principal, gestionând toate solicitările HTTP și coordonând diferitele componente ale aplicației.

Introducere

În era digitală contemporană, aplicațiile web sunt esențiale pentru infrastructura tehnologică a societății. Alegeți cadrul potrivit pentru dezvoltare pentru a obține produse eficiente, scalabile și ușor de întreținut.

Python este frecvent considerat limbajul de programare interpretat și multiparadigmă preferat pentru crearea aplicațiilor web, datorită comunității sale extinse de dezvoltatori și sintaxei sale clare. În această lucrare, vom examina arhitectura framework-ului Flask, care este unul dintre cele mai populare microframework-uri Python, precum și cât de util este pentru proiecte reale.

Prezentarea generală a framework-ului Flask

În 2010, Armin Ronacher a creat Flask în timp ce lucra la proiectul Pocoo. A fost conceput ca un cadru simplu, astfel încât dezvoltatorii să aibă cel mai mare grad de autonomie atunci când aleg și integrează componentele necesare. Flask a fost construit folosind toolkit-ul Werkzeug și motorul de template Jinja2, ambele dezvoltate de echipa Pocoo.

Filosofia din spatele sticlei „micro”: Un „micro-cadru” conține componente predefinite precum ORM-ul sau sistemul de autentificare, ceea ce îi permite să funcționeze fără limitări. Această metodă oferă un grad ridicat de flexibilitate, permițând dezvoltatorului să modifice aplicația pentru a se potrivi nevoilor specifice ale proiectului.

Arhitectura Flask și fluxul de lucru

Componentele sale esențiale sunt susținute de două biblioteci principale: Werkzeug – un set de instrumente WSGI care facilitează comunicarea între aplicația Python și serverul web. Jinja2 este un motor de template-uri care permite separarea logicii aplicației de componenta de prezentare (HTML).

Routing și controlul cererilor

Routing-ul este modul în care Flask trimite cererile HTTP către funcțiile specificate de utilizator. Acest lucru este realizat cu ajutorul decoratorilor Python (@app.route), care permit o asociere simplă între URL-uri și funcțiile de procesare.

Avantaje și limitări

Avantajele includ următoarele: flexibilitate ridicată – dezvoltatorul are control total asupra structurii aplicației. Curba de învățare redusă este potrivită atât pentru începători, cât și pentru prototipuri rapide. Există multe extensii comunitare și oficiale pentru Flask, cum ar fi Flask-SQLAlchemy, Flask-Login și Flask-WTF. În plus, există câteva restricții: funcționalitățile nu sunt implicite și dezvoltatorii trebuie să integreze componentele manual. Managementul aplicațiilor mari: proiectele complexe pot fi dificile de întreținut dacă nu au o structură strictă.

Cazuri de utilizare și exemple practice

Proiecte educaționale, startup-uri, aplicații RESTful și prototipuri MVP (Minimum Viable Product) folosesc frecvent Flask. Pinterest și LinkedIn au folosit Flask pentru anumite secțiuni ale infrastructurii lor, de exemplu. Aplicația tipică care folosește Flask include crearea unui blog sau a unei interfețe CRUD conectate la o bază de date SQLite folosind Flask-SQLAlchemy.

Concluzii

Framework-ul Flask este o soluție practică și elegantă pentru dezvoltarea aplicațiilor web cu Python, care este ideală pentru cei care apreciază extensibilitatea, controlul asupra arhitecturii și ușurința. Deși nu este la fel de complet ca alte structuri de tip „full-stack”, Flask rămâne relevant prin abordarea sa minimalistă, care îl face ideal pentru proiecte flexibile, prototipuri și învățare. În cele din urmă, Flask este un punct de plecare grozav atât pentru începători, cât și pentru dezvoltatorii cu experiență.

Exemplu

```python
from flask import Flask, render_template, request, redirect, url_for
/
app = Flask(__name__)
/
@app.route('/')
def index():
    categories = Category.query.all()
    return render_template( template_name_or_list: 'index.html', categories=categories)
/topic/{topic_id}
@app.route('/topic/<int:topic_id>')
def topic(topic_id):
    topic = Topic.query.get_or_404(topic_id)
    return render_template( template_name_or_list: 'topic.html', topic=topic)
```

Figură 6 - Exemplu Flask

SQLAlchemy ORM – Managementul Bazelor de Date în Python

Utilizare în Dev Forum

Aplicația folosește SQLAlchemy pentru a defini modele de baze de date, pentru a stabili relații între ele și pentru a efectua operațiuni de bază de date.

Introducere

Baza de date reprezintă un element central în arhitectura majorității aplicațiilor software moderne. În special în dezvoltarea web și enterprise, interacțiunea cu datele persistente trebuie să fie sigură, eficientă și ușor de întreținut. Limbajul de programare Python, consacrat pentru sintaxa sa concisă și expresivă, dispune de multiple biblioteci pentru gestionarea bazelor de date, însă una dintre cele mai performante și flexibile soluții este SQLAlchemy.

Această lucrare explorează în profunzime componenta ORM (Object Relational Mapper) a SQLAlchemy, analizându-i fundamentele teoretice, caracteristicile arhitecturale, beneficiile practice și cazurile de utilizare în proiecte reale.

Definiție și scop

O tehnologie numită maparea obiect-relațională (ORM) facilitează conversia datelor între sisteme care au tipuri de date incompatibile de tip obiectual și relaționale. În special, ORM elimină necesitatea scrierii directe a interogărilor SQL, permițând manipularea datelor dintr-o bază relațională prin intermediul obiectelor într-un limbaj de programare orientat pe obiecte.

Avantaje față de accesul tradițional

Utilizarea unui ORM aduce beneficii semnificative:

- **Reducerea redundanței codului** prin generarea automată a interogărilor SQL.

- **Abstractizarea logicii de date** față de schema relațională.
- **Securitate sporită** împotriva atacurilor de tip SQL Injection.
- **Productivitate crescută** în dezvoltarea aplicațiilor complexe.

Prezentarea SQLAlchemy

Înființat de Michael Bayer, SQLAlchemy a fost lansat în 2005 pentru a oferi un cadru extensibil, eficient și non-invaziv pentru lucrul cu baze de date în Python. SQLAlchemy încurajează libertatea de a alege între utilizarea ORM-urilor de nivel înalt sau lucrul direct cu SQL („Core”). Acest lucru îl diferențiază de alte ORM-uri care cer arhitecturistricte. Două componente principale alcătuiesc SQLAlchemy: SQLAlchemy Core este un set de instrumente pentru generarea dinamică a SQL-ului, gestionarea conexiunilor și tranzacțiile. SQLAlchemy ORM conectează clase Python la tabele SQL printr-un strat obiectual deasupra Core. Cu această separare, integrarea progresivă a ORM este posibilă, oferind dezvoltatorilor control strict asupra fiecărui nivel al interacțiunii cu baza de date.

Maparea claselor și relațiilor

SQLAlchemy permite maparea claselor Python la tabele SQL prin declararea atributelor ca instanțe ale obiectelor coloane. Tipurile explicite, cum ar fi Integer, String, Boolean și altele, sunt utilizate pentru a defini tipurile de date, iar funcțiile relationship() și foreign_key sunt utilizate pentru a defini relațiile dintre tabele.

Mecanisme complexe de optimizare a performanței sunt oferite de SQLAlchemy prin controlul modului în care sunt încărcate datele relaționate. Acest lucru se numește încărcare lazy, în care datele sunt încărcate doar atunci când sunt accesate. iar încărcarea rapidă, în care sunt create întrebări pentru a include datele asociate într-o singură execuție.

Managementul sesiunilor

În SQLAlchemy ORM, sesiunea este un element esențial pentru a menține o sesiune tranzacțională între aplicație și baza de date. Adăugarea, modificarea, commit-ul și rollback-ul sunt operații care pot fi efectuate prin Session pentru a garanta consistența datelor.

Modelul „unitei de muncă” al SQLAlchemy combină toate modificările obiectelor într-o singură tranzacție atomică, ceea ce permite rollback-ul în caz de eroare și păstrarea integrității datelor.

SQLAlchemy vs Django ORM

În timp ce Django ORM este mai ușor de utilizat pentru aplicații convenționale, SQLAlchemy este o alegere preferată pentru proiecte complexe sau corporative datorită flexibilității și controlului net superior. Interogările dinamice, suportul pentru baze de date multiple și mapările avansate (cum ar fi moștenirea tabelului conectat) sunt toate posibilitățile oferite de SQLAlchemy. Alte ORM-uri cunoscute: Peewee este mai simplu, dar are puține funcții avansate. Comparativ cu SQLAlchemy, Pony ORM oferă un DSL (Domain Specific Language) propriu, dar are mai puțin suport comunitar și documentație.

Proiect real: API RESTful cu Flask și SQLAlchemy

De obicei, SQLAlchemy și Flask sunt folosite împreună pentru a crea API-uri REST. Într-un proiect tipic, Flask gestionează logica de business și rutele, iar modelele SQLAlchemy stabilesc structura bazei de date. Persistența se obține prin sesiuni și se pot aplica validări cu ajutorul unui flask-WTF sau al unui mărmeluș. SQLAlchemy este utilizat în multe proiecte diferite, cum ar fi aplicații interne pentru companii tehnologice, sisteme CRM sau ERP, platforme de analiză de date și aplicații distribuite cu scalabilitate.

Concluzii

SQLAlchemy ORM este un bun exemplu de integrare între modelul relațional de baze de date și programarea orientată pe obiecte. Pentru dezvoltatorii care au nevoie de control asupra interacțiunilor cu baza de date fără a renunța la productivitate sau abstractizare, SQLAlchemy este o soluție robustă și scalabilă datorită arhitecturii sale flexibile. Pentru a utiliza ORM cu succes, este necesară o înțelegere solidă atât a bazelor SQL, cât

și a principiilor OOP. Din acest motiv, SQLAlchemy este un instrument de instruire esențial în pregătirea profesională a oricărui inginer software.

Exemplu

```python
from sqlalchemy import Column, Integer, String, Text, DateTime, ForeignKey
from sqlalchemy.orm import relationship

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False, index=True)
    email = db.Column(db.String(120), unique=True, nullable=False, index=True)
    posts = db.relationship('Post', backref='author', lazy='dynamic')

    def set_password(self, password):  1 usage (1 dynamic)
        self.password_hash = generate_password_hash(password)
```

Figură 7 - SQLAlchemy Modelul USER

Flask-Login – Gestionarea Autentificării în Aplicații Web Python

Utilizare în Dev Forum

Aplicația folosește Flask-Login pentru a gestiona autentificarea utilizatorilor, pentru a proteja rutele care necesită autentificare și pentru a oferi acces utilizatorului curent în șabloane

Introducere

Autentificarea utilizatorilor este o parte esențială a dezvoltării aplicațiilor web pentru a garanta securitatea și personalizarea serviciilor. Procesul de autentificare include mai mult decât verificarea identității; include gestionarea întregii sesiuni, protecția datelor și controlul accesului la resursele aplicației. Framework-ul Flask se remarcă în ecosistemul Python prin adaptabilitatea și extensibilitatea, care permite integrarea unor soluții modulare pentru o gamă largă de funcționalități. În acest sens, o extensie populară este Flask-Login, o bibliotecă care permite gestionarea și autentificarea sesiunilor într-un mod sigur, logic și eficient.

Prezentarea extensiei Flask-Login și integrarea în Flask

O extensie pentru aplicațiile care folosesc Flask, Flask-Login facilitează gestionarea autentificării utilizatorilor și menținerea sesiunilor active. Această bibliotecă este proiectată pentru a facilita integrarea, menținând în același timp flexibilitatea necesară proiectelor de diferite dimensiuni. În mod automat, dezvoltatorul poate gestiona autentificarea, autentificarea, protecția paginilor, încărcarea utilizatorilor dintr-o sursă de date și verificarea autentificării. Arhitectura extensiei se bazează pe ideea de user loader, o funcție care colectează obiectele asociate pentru un utilizator autentificat specific, și pe un sistem de sesiuni care folosește cookies pentru a menține starea de conectare între cereri HTTP consecutive.

Integrarea Flask-Login într-o aplicație presupune inițializarea extensiei la nivelul instanței principale Flask și definirea unei clase care să reprezinte utilizatorul. Această clasă trebuie să implementeze câteva metode standard, printre care se numără is_authenticated, is_active, get_id și, opțional, is_anonymous. Aceste metode sunt utilizate intern de Flask-Login pentru a determina dacă un utilizator este conectat, dacă are drepturi de acces și pentru a-i

identifica unic sesiunea în derulare. Odată configurat acest model de utilizator, aplicația poate utiliza decoratorul @login_required pentru a restricționa accesul la anumite rute doar utilizatorilor autentificați, consolidând astfel securitatea logicii aplicației.

Mecanisme de gestionare a sesiunii și securitate

Utilizarea mecanismului de cookies pentru a menține continuitatea sesiunii utilizatorului autentificat este o caracteristică centrală a Flask-Login. Extensia creează un identificator unic în browser-ul clientului atunci când se conectează. Acest identificator este verificat la fiecare cerere ulterioară. Acest mecanism permite menținerea autentificării pe tot parcursul vizitei și, opțional, persistarea stării chiar și după închiderea browserului, prin activarea opțiunii „amintește-mă”.

Pentru a proteja datele de autentificare, Flask-Login nu gestionează singur parolele. În schimb, funcționează împreună cu alte biblioteci, cum ar fi Werkzeug sau Flask-Bcrypt, care permit criptarea și verificarea parolelor. Metodele de protecție împotriva atacurilor de tip Cross-Site Request Forgery (CSRF), care se pot combina cu alte extensii precum Flask-WTF, îmbunătățesc și securitatea aplicației. În plus, Flask-Login permite personalizarea comportamentului unui utilizator neautentificat în cazul în care încearcă să acceseze o resursă protejată. De exemplu, permite redirecționarea către o pagină de login sau returnarea unui mesaj de eroare clar.

Considerații arhitecturale și bune practici

Implementarea unui sistem de autentificare puternic trebuie să ia în considerare nu numai caracteristicile fundamentale ale sistemului, ci și modul în care acesta se integrează în întreaga aplicație. În acest sens, Flask-Login permite dezvoltatorilor să păstreze un echilibru între flexibilitate și simplitate, permițându-le să separe logica de autentificare de alte părți ale aplicației. O practică bună este crearea unui blueprint special pentru toate rutele de autentificare și păstrarea clasei utilizator într-un fișier separat. Acest lucru face mai ușor să organizezi codul și să reutilizeze componentele în aplicații mai mari.

Este esențial ca autentificarea să fie dublată de un sistem robust de autorizare. Deși Flask-Login nu oferă un mecanism de gestionare a permisiunilor nativ, logica de acces poate fi extinsă manual folosind alte biblioteci sau folosind roluri sau privilegii specificate în modelul utilizatorului. O atenție deosebită trebuie acordată validării datelor transmise prin formulare, prevenirii păstrării parolelor într-un format text simplu și protejării fișierelor de configurare sensibile, cum ar fi cheia secretă a aplicației.

Aplicații și studii de caz

Proiecte comerciale, aplicații educaționale, API-uri RESTful și prototipuri de produse folosesc Flask-Login. În mod normal, aplicația are o bază de date relațională care conține informații despre utilizatori și un sistem pentru înregistrare, autentificare și logout. Interfața de autentificare folosește adesea extensia Flask-WTF, care permite crearea și validarea formularelor într-un mod sigur și plăcut. Adăugarea de sesiuni paralele, verificarea tipului de utilizator și integrarea cu sisteme de autentificare externe precum OAuth 2.0 sau LDAP fac parte din funcționalitățile Flask-Login pentru aplicațiile care au mai mult de un utilizator.

Un exemplu specific de utilizare ar putea fi un sistem de management al proiectelor intern al unei organizații în care administratorii au acces la toate resursele și fiecare utilizator are acces la propriile resurse. În astfel de situații, Flask-Login demonstrează a fi o opțiune robustă și scalabilă pentru gestionarea autentificării, fără a afecta performanța sau ușurința de utilizare a aplicației.

Concluzii

Flask-Login este considerat o extensie vitală pentru dezvoltatorii care creează aplicații web sigure și bine structurate în Python. Extensia facilitează implementarea proceselor de autentificare și menținere a sesiunilor prin abordarea sa simplă, dar eficientă, fără a impune restricții arhitecturale stricte. Este ușor de integrat cu alte părți ale ecosistemului Flask datorită flexibilității sale, iar comunitatea activă și documentația extinsă îl fac popular. În cele din urmă, Flask-Login este mai mult decât un instrument de logare; este esențial pentru crearea de aplicații moderne care îndeplinesc cerințele actuale de funcționalitate și securitate.

Exemplu

```python
from flask_login import LoginManager, login_user, login_required, current_user

login_manager = LoginManager(app)
login_manager.login_view = 'login'

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

@app.route('/login', methods=['POST'])
def login():
    user = User.query.filter_by(username=request.form['username']).first()
    if user and user.check_password(request.form['password']):
        login_user(user)
```

Figură 8 - Flask-LOGIN

Optimizarea Performanței Aplicațiilor Web cu Flask-Caching

Utilizare în Dev Forum

Aplicația folosește Flask-Login pentru a gestiona autentificarea utilizatorilor, pentru a proteja rutele care necesită autentificare și pentru a oferi acces utilizatorului curent în șabloane

Introducere

Într-o epocă în care timpul de răspuns al unei aplicații web poate afecta direct rata de retenție a utilizatorilor și succesul comercial al produselor digitale, optimizarea performanței devine un obiectiv central în arhitectura sistemelor moderne. Utilizarea mecanismului de caching este una dintre cele mai bune metode de a reduce timpul de procesare și încărcarea serverului. Aceste mecanisme permit stocarea temporară a rezultatelor operațiunilor costisitoare pentru a fi reutilizate ulterior. Extensia Flask-Caching oferă o soluție flexibilă și extinsă pentru integrarea strategiilor de cache în aplicațiile dezvoltate cu Flask, un microframework scris în Python. În această lucrare, vom examina în detaliu modul în care Flask-Caching ajută la optimizarea aplicațiilor web.

Conceptul de caching și importanța sa în arhitectura web

În esență, cache-ul este procesul de stocare temporară a datelor care sunt dificile sau costisitoare de obținut în mod repetat. Acest lucru este făcut pentru a reduce timpul de răspuns și pentru a îmbunătăți eficiența aplicațiilor. Acest mecanism poate fi implementat în mediul web la nivel de client, server sau rețea și poate include păstrarea răspunsurilor HTTP, generarea de pagini HTML dinamice, rezultatele interogărilor către baza de date sau conținutul API-urilor externe.

Caching-ul reduce timpul de procesare, volumul de muncă al serverului, consumul de bandă și contribuirea la scalabilitatea întregului sistem prin eliminarea necesității de a regenera aceeași resursă la fiecare cerere. Cu toate acestea, deoarece conținutul poate deveni rapid învechit, caching-ul face gestiunea datelor mai complicată, iar sincronizarea între datele cache și sursa lor de adevăr devine esențială.

Introducere în Flask-Caching și principiile sale de funcționare

O extensie dedicată aplicațiilor construite cu cadrul Flask, Flask-Caching oferă un sistem extensibil și centralizat pentru aplicarea politicilor de cache. Flask-Caching funcționează prin decoratori care pot fi atașați la funcții costisitoare, permițând stocarea rezultatelor acestora într-un backend de cache configurabil. Este conceput pentru a fi ușor de utilizat, dar suficient de flexibil pentru aplicații complexe. Extensia poate suporta diferite sisteme, inclusiv cache-ul în memorie (SimpleCache), cache-ul sistemului de fișiere (FileSystemCache), Redis, Memcached și chiar baze de date relaționale, deoarece este menită să fie agnostică în backend.

Fiecare funcție decorată dintr-un set de argumente specific este evaluată o singură dată în procesul de caching, iar rezultatul ei este stocat împreună cu o cheie specială care a fost creată pe baza acestor argumente. În timpul cererilor ulterioare cu aceleași intrări, rezultatul este extras din cache fără a mai efectua funcția inițială. Acest lucru reduce semnificativ timpul de execuție. Folosind parametri precum timeout, Flash-Caching permite un control precis asupra duratei de viață a valorilor cache, oferind astfel un echilibru între performanță și actualitatea datelor.

Modele de caching și strategii de invalidare

Alegerea unei strategii care funcționează pentru fiecare aplicație este esențială pentru implementarea unui sistem de caching eficient. Caching-ul la nivel de funcție este unul dintre cele mai utilizate tipuri. Acest tip de caching memorează rezultatele unei operațiuni de procesare intensă, cum ar fi crearea unei pagini de dashboard sau calcularea agregărilor statistice, pentru o perioadă de timp stabilită. În mod similar, poate fi folosit caching la nivelul răspunsurilor HTTP complete sau al componentelor HTML atunci când se lucrează cu motoare de template precum Jinja2. Pe lângă aceste modele, poate fi folosit caching-ul per utilizator sau pe bază de permisiuni pentru a face personalizarea interfețelor eficiente fără a trebui să repetați procese de calcul puternice.

Invalidarea cache-ului, care înseamnă ștergerea sau reîmprospătarea datelor stocate, este o parte vitală și adesea dificilă a arhitecturii sistemelor care se bazează pe caching. Utilizând Flask-Caching, puteți curăța cache-ul fie pe scară largă, fie pe baza unei anumite chei. De asemenea, puteți configura automat un timp de expirare. Natura, volumul și nivelul de consistență dorit sunt factori importanți în alegerea unei strategii de invalidare. De exemplu, modificarea prețului unui produs într-o aplicație de e-commerce ar trebui să invalideze imediat toate cache-urile asociate cu acel produs. Pe de altă parte, reîmprospătarea articolelor într-un blog poate fi făcută frecvent, fără a afecta semnificativ coerența informațiilor.

Performanță și scalabilitate în aplicațiile Flask cu caching

Caching-ul reduce timpii de răspuns, încărcarea procesorului și a bazei de date și capacitatea aplicației de a deservi un număr mai mare de utilizatori simultan. Acest lucru se datorează în mod direct faptului că reduce numărul de operațiuni costisitoare. Testele comparative arată că utilizarea eficientă a îngropării flașcelor poate reduce timpul de răspuns cu peste 50% în aplicațiile cu un trafic mare și conținut dinamic, dar repetat. De asemenea, este posibil să scădeze semnificativ rata de erori cauzate de suprasarcină.

Pentru implementările distribuite, caching-ul trebuie proiectat pentru a ține cont de caracterul paralel al procesării și de posibilitatea fragmentării memoriei cache între noduri. Backend-uri precum Redis și Memcached permit valorilor cache să fie împărțite între mai multe instanțe ale aplicației, ceea ce asigură o performanță constantă și o coerență relativă. Deoarece oferă parametri de configurare care se aliniază cu bunele practici în infrastructura de producție, Flask-Caching facilitează integrarea acestor sisteme externe.

Considerații de securitate și limitări

Chiar dacă caching-ul are multe beneficii din punct de vedere al performanței, utilizarea necontrolată poate duce la probleme de securitate și confidențialitate. Utilizatorii neautorizați pot accesa datele sensibile dacă se stocă temporar în cache-uri partajate fără izolare adecvată. De aceea, atunci când se creează politici de caching, trebuie luate în considerare atât performanța, cât și confidențialitatea și integritatea. Ar trebui folosiți identificatori unici pentru a cifra fiecare utilizator, iar atunci când este necesar, datele ar trebui criptate sau restricționate.

Cu toate acestea, Flask-Caching are câteva dezavantaje, în special în ceea ce privește gestionarea rapidă a dependențelor dintre date și invalidarea automată a cache-ului dacă sursele externe suferă modificări. De

cele mai multe ori, în aplicațiile complexe, este necesară îmbunătățirea Flask-Caching cu instrumente suplimentare sau crearea unor metode proprii de urmărire a coerenței datelor cache.

Concluzii

Extensia Flask-Caching oferă o soluție puternică și eficientă pentru optimizarea aplicațiilor web dezvoltate în Flask, care îmbunătățește semnificativ performanța și scalabilitatea aplicațiilor. Extensia are o arhitectură simplă, dar adaptabilă, care permite dezvoltatorilor să integreze rapid politici de caching pentru a se potrivi cu diferite tipuri de date și niveluri de acces. În plus, pentru a asigura eficiența și consistența informațională, implementarea corectă a caching-ului necesită o înțelegere aprofundată a aplicației, a fluxurilor de date și a relațiilor dintre componente. Flask-Caching, care are o documentație detaliată și o comunitate activă, se impune ca o extensie vitală a arsenalului dezvoltatorului Python contemporan, ajutând la crearea unor aplicații performante, fiabile și pregătite pentru scalare.

Exemplu

```python
from flask_caching import Cache

cache = Cache(app)

@app.route('/category/<int:category_id>')
@cache.cached(timeout=60, query_string=True)
def category(category_id):
    category = Category.query.get_or_404(category_id)
    topics = Topic.query.filter_by(category_id=category_id).order_by(Topic.created_at.desc())
    return render_template('category.html', category=category, topics=topics)
```

Figură 9 - Flask-Caching

Integrarea Bazelor de Date în Aplicații Web Python cu Flask-SQLAlchemy

Utilizare în Dev Forum

Aplicația folosește Flask-SQLAlchemy pentru a defini modele, pentru a stabili conexiuni la baze de date și pentru a efectua operațiuni de bază de date în contextul aplicației Flask.

Introducere

Manipularea datelor persistente este esențială pentru arhitectura modernă a aplicațiilor web. Indiferent de amploarea aplicației, gestionarea eficientă a bazelor de date devine un obiectiv tehnologic central. Framework-ul Flask se remarcă prin minimalism și extensibilitate, oferind dezvoltatorilor un grad ridicat de control asupra arhitecturii aplicației. Acest lucru este remarcabil în contextul dezvoltării cu Python, un limbaj recunoscut pentru expresivitatea sa sintactică și ecosistemul său bogat în biblioteci. Cu toate acestea, versiunea de bază a Flask nu oferă caracteristici avansate de interacțiune cu baze de date. Astfel, extensia Flask-SQLAlchemy este un instrument esențial pentru dezvoltatorii care doresc să integreze un sistem de mapare obiect-relațională (ORM) puternic în aplicațiile lor.

Această lucrare explorează în profunzime funcționarea, structura, avantajele și limitările extensiei Flask-SQLAlchemy, punând accent pe impactul său asupra eficienței și scalabilității aplicațiilor web Python.

Fundamentele ORM și SQLAlchemy în Python

Înainte de a analiza integrarea propriu-zisă realizată de Flask-SQLAlchemy, este esențială înțelegerea conceptului de mapare obiect-relațională, abreviat ORM, și a modului în care acesta influențează interacțiunea

aplicației cu baza de date. În paradigma orientată pe obiecte, datele și operațiile asupra acestora sunt încapsulate în clase și instanțe, în timp ce bazele de date relaționale organizează informațiile în tabele, rânduri și coloane. ORM unește aceste două paradigme, permițând dezvoltatorilor să lucreze cu baze de date folosind concepte familiare programării orientate pe obiecte fără a scrie interogări SQL direct. Biblioteca de bază pe care se sprijină Flask-SQLAlchemy, SQLAlchemy, este una dintre cele mai puternice și versatile implementări ORM din ecosistemul Python. Oferă atât un grad ridicat de abstractizare, cât și capacitatea de a lucra direct cu expresii SQL atunci când este necesar. SQLAlchemy permite definirea modelelor de date ca clase Python și le mapează automat la structura bazei de date. Suportă moștenirea, validarea, relațiile complexe, tranzacțiile și integrarea multiplă cu o varietate de tipuri de baze de date, inclusiv PostgreSQL, MySQL, SQLite și Oracle.

Integrarea SQLAlchemy în Flask prin Flask-SQLAlchemy

În calitate de micro-framework, Flask nu oferă suport pentru gestionarea bazelor de date în mod nativ. Cu toate acestea, arhitectura sa modulară permite adăugarea acestei funcții prin extensii externe. Flask-SQLAlchemy oferă o interfață simplă pentru utilizarea SQLAlchemy în aplicație și este printre cele mai populare extensii ale ecosistemului Flask.

Aceasta facilitează semnificativ procesul de configurare și integrare, oferind o clasă SQLAlchemy care gestionează conexiunea la baza de date, definirea modelelor și sesiunile de interacțiune cu datele. Dezvoltatorul poate accesa o interfață unificată pentru toate operațiunile de persistare a datelor prin înregistrarea instanței SQLAlchemy în aplicația Flask. Acest lucru îi permite să gestioneze conexiunile, tranzacțiile sau configurarea dialectelor SQL manual.

ORM extinde clasa de bază oferită de Flask-SQLAlchemy pentru a defini modele de date. Fiecare model este reprezentat de o clasă Python care definește atributele tabelei corespondente. Tabela are coloane care conțin atributele clasei, iar relațiile dintre tabele sunt reprezentate prin referințe directe între modele. Instanțele acestor clase permit accesul la date, în timp ce operațiile CRUD (creare, citire, actualizare și eliminare) sunt executate folosind metodele standard oferite de SQLAlchemy. Toate acestea sunt coordonate prin sesiunea implicită gestionată de Flask-SQLAlchemy.

Arhitectura internă și ciclul de viață al unei interacțiuni cu baza de date

Arhitectura Flask-SQLAlchemy este menită să integreze logica de acces la date în ciclul de viață al unei aplicații web, menținând în același timp caracterul asincron și stateless al cererilor HTTP. Atunci când aplicația este lansată, obiectul SQLAlchemy este conectat la baza de date cu parametrii și este responsabil pentru crearea tabelelor, gestionarea tranzacțiilor și asigurarea integrității operațiilor asupra datelor. O sesiune este executată pentru fiecare cerere HTTP care implică o interacțiune cu datele. Această sesiune înregistrează toate modificările aduse obiectelor modelului și le persistă în baza de date la finalizarea cererii, prin commit. În cazul în care apare o excepție sau o eroare logică, sesiunea poate fi anulată prin rollback, astfel încât operațiile incomplete sau invalide să nu afecteze datele.

Această împărțire clară a sesiunilor pentru fiecare cerere este esențială pentru scalabilitatea aplicației și pentru a evita probleme. În plus, Flask-SQLAlchemy integrează cu extensii precum Flask-Migrate și Flask-Login, oferă mecanisme pentru migrarea bazei de date și permite utilizarea modelelor de moștenire și relațiilor avansate, care evidențiază complexitatea logicii aplicației.

Avantajele utilizării Flask-SQLAlchemy în proiecte web

În dezvoltarea aplicațiilor web, utilizarea extensiei Flask-SQLAlchemy are multe avantaje, ceea ce o face o opțiune potrivită atât pentru aplicații de dimensiuni mici, cât și pentru arhitecturi de afaceri. Deoarece majoritatea operațiunilor de bază sunt executate printr-o interfață obiectuală ușor de înțeles, simplificarea codului și eliminarea redundanței sunt printre beneficiile cele mai evidente. În plus, extensia reduce semnificativ riscul de erori logice sau inconsistențe între nivelul aplicației și cel de stocare, oferind un nivel ridicat de coerență între modele și structura bazei de date. Deoarece este ușor de integrat cu alte extensii din ecosistemul Flask, permite crearea rapidă a aplicațiilor complete care includ autentificare, migrație, cache și validare, toate bazate pe modele comune.

În plus, portabilitatea aplicației este îmbunătățită de faptul că Flask-SQLAlchemy oferă utilizatorilor

posibilitatea de a schimba între diferite sisteme de gestiune a bazelor de date prin modificarea simplă a stringului de conexiune, fără a necesita modificarea codului logic al aplicației. În plus, API-ul ORM oferă suport pentru relații complexe între modele, moștenire multiplă și interogări dinamice, ceea ce permite crearea de aplicații complexe fără a compromite lizibilitatea sau testabilitatea codului.

Limitări, riscuri și bune practici în utilizarea Flask-SQLAlchemy

Deși Flask-SQLAlchemy oferă o gamă largă de funcționalități, utilizarea necorespunzătoare poate duce la probleme pe termen lung cu performanța, securitatea sau mentenanța. Una dintre cele mai frecvente probleme este legată de gestionarea ineficientă a interogărilor în aplicațiile mari. În situații în care relațiile dintre modele nu sunt bine definite sau când se folosește lazy loading în contexte inadecvate, ORM-ul poate produce SQL-uri suboptime. În astfel de situații, interogările create pot genera întrebări N+1, care este o problemă obișnuită în ORM-uri și are un impact asupra modului în care aplicația funcționează în medii de producție.

Pentru a evita aceste probleme, este esențial ca dezvoltatorii să înțeleagă în profunzime modul în care SQLAlchemy generează și execută interogările, să folosească debug tools și să testeze performanța la scară încă din fazele inițiale ale dezvoltării.

Securitatea este o altă componentă esențială. Deși ORM-ul oferă protecție implicită împotriva atacurilor de tip SQL injection, validarea atentă a datelor introduse de utilizatori, protejarea sesiunilor și criptarea datelor sensibile sunt încă necesare. În plus, pentru a menține coerența în zonele de dezvoltare, testare și producție, orice aplicație care folosește Flask-SQLAlchemy ar trebui să folosească practici puternice de versionare a schemei bazei de date, folosind instrumente precum Flask-Migrate.

Concluzii

Flask-SQLAlchemy oferă un echilibru perfect între simplitate, flexibilitate și putere expresivă și se impune ca un instrument esențial pentru dezvoltarea aplicațiilor web cu Python. Această extensie permite dezvoltatorilor să construiască aplicații robuste, scalabile și ușor de întreținut, care se aliniază cu standardele moderne ale industriei software, prin integrarea sa elegantă cu framework-ul Flask și prin valorificarea potențialului ORM al SQLAlchemy. În ciuda complexității ascunse în spatele unei interfețe aparent simple, Flask-SQLAlchemy oferă un model de lucru care promovează practicile bune de programare și reduce semnificativ timpul de dezvoltare. Prin urmare, alegerea sa nu este doar o decizie tehnică; este, de asemenea, un angajament față de calitate, eficiență și durabilitate în ingineria aplicațiilor web.

Exemplu

```
from flask_sqlalchemy import SQLAlchemy


/
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///forum.db'
db = SQLAlchemy(app)

class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    content = db.Column(db.Text, nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
```

Figură 10 - Flask-SQLAlchemy

Motorul de Template Jinja2 – Funcționalitate, Arhitectură și Rolul Său în Aplicațiile Web Python Utilizare în Dev Forum

Aplicația folosește **Jinja2** pentru a reda toate paginile HTML, trecând date din backend pentru a fi afișate în frontend.

Introducere

În timpul perioadei de dezvoltare rapidă a aplicațiilor web, separarea logicii de afaceri de interfața utilizatorului a devenit o practică esențială în arhitectura software modernă. Separarea preocupărilor (separation of concerns) este un principiu de proiectare care permite crearea unor aplicații mai ușor de întreținut, scalabile și testabile. Deoarece Python este unul dintre cele mai populare limbaje de programare, dezvoltatorii beneficiază de o varietate de unelte și biblioteci care sprijină această abordare modulară. Acestea includ, de asemenea, motorul de template Jinja2, care este componenta principală folosită pentru generarea dinamică a conținutului HTML în cadrul framework-urilor web precum Flask.

Scopul acestei lucrări este de a examina în detaliu arhitectura și funcționalitatea motorului Jinja2, de a sublinia avantajele și dezavantajele acestuia și de a demonstra modul în care acesta contribuie la dezvoltarea eficientă a interfețelor web dinamice.

Fundamentele motoarelor de template și necesitatea lor în dezvoltarea web

Pentru a înțelege funcția pe care Jinja2 o joacă într-o aplicație web, este necesar să înțelegem conceptele de bază ale motoarelor de template. Un motor de template este un sistem software care poate combina date dinamice din aplicații cu conținut static (de obicei în format HTML sau XML) pentru a produce o ieșire completă care este furnizată utilizatorului. Această metodă facilitează distincția clară între secțiunea de procesare și cea de prezentare, deoarece elimină necesitatea inserării codului HTML direct în logica aplicației. De la sisteme simple de concatenare a șirurilor de caractere până la structuri expresive și sigure, motoarele de template au evoluat semnificativ în ultimele decenii. Aceste structuri suportă condiții, iterații, includeri și moștenirea template-urilor, precum și apeluri la funcții sau filtre specificate de utilizator.

În primul rând, în mediul Python, această necesitate a fost satisfăcută prin template-uri simple, cum ar fi cele oferite de biblioteca string standard. Deși template, a devenit necesar un instrument mai puternic odată cu complexificarea aplicațiilor web. Prin urmare, Jinja2 a apărut ca o soluție actualizată, inspirată de motorul de template Django, dar cu mai multă flexibilitate și o sintaxă ușor de înțeles care reflectă în mod natural structura HTML.

Arhitectura și principiile de funcționare ale Jinja2

Jinja2 este un motor de template puternic, dezvoltat în Python. Funcționează prin procesarea fișierelor de template care conțin expresii și cod HTML proprii limbajului Jinja, care sunt apoi interpretate și înlocuite la runtime cu valori dinamice. În primul rând, Jinja2 transformă aceste fișiere într-un arbore de sintaxă abstractă. Apoi, îl compilează într-o funcție Python internă, creând un șir de caractere HTML care arată conținutul final pe care utilizatorul îl poate vedea.

Deoarece template-urile pot fi precompilate și memorate în cache, ceea ce reduce timpul de procesare la cereri repetate, această arhitectură compusă din parser, compilator și executor asigură un nivel ridicat de performanță. Jinja2 oferă un limbaj de template expresiv care include bucle, blocuri condiționale, variabile, macrocomenzi, extensii personalizate și filtre. Similurile speciale care delimitează blocurile de cod ({{ }} pentru expresii, {% %} pentru instrucțiuni și {# # } pentru comentarii) permit o integrare elegantă cu HTML fără a afecta lizibilitatea.

Sistemul de moștenire a template-urilor este o caracteristică semnificativă a arhitecturii Jinja2. Acest lucru vă permite să creați un fișier de bază, cunoscut sub numele de „template master”, care conține secțiunile comune tuturor paginilor, cum ar fi subsolul, bara de navigare și antetul. Acest template poate fi extins prin suprascrierea doar a blocurilor relevante, ceea ce reduce cantitatea de cod duplicat și facilitează întreținerea interfeței.

Integrarea Jinja2 în Flask și fluxul de generare al paginilor

Folosind Jinja2 ca motor de template implicit, Flask facilitează generarea conținutului HTML dinamic. Pentru a interacționa cu Jinja2 în aplicațiile Flask, se utilizează funcția render_template. Această funcție primește un nume de fișier template și un set de variabile care vor fi disponibile în fișier. Aceste variabile pot fi obținute din baza de date, logica aplicației sau input-ul utilizatorului și pot fi accesate în template folosind o sintaxă Jinja2 specială.

Pentru a crea o pagină web, Flask primește o cerere HTTP. Apoi, găsește funcția de vizualizare potrivită rutei solicitate. Dacă datele care sunt necesare sunt procesate de această funcție, acesta apelează render_template. La rândul său, motorul Jinja2 este inițiat pentru a combina datele care au fost primite cu structura care este stabilită în fișierul template. O pagină HTML completă este returnată clientului ca răspuns la cerere. Un beneficiu semnificativ al acestei integrări este că codul HTML este complet separat de codul Python, ceea ce permite dezvoltatorilor front-end și back-end să lucreze împreună pe același proiect fără a interfera unul cu celălalt. structura logică. În plus, Jinja2 permite utilizarea macro-urilor și a includerilor, ceea ce permite reutilizarea elementelor vizuale în diferite părți ale aplicației.

Securitate, performanță și extensibilitate în Jinja2

În ceea ce privește securitatea, Jinja2 implementează mecanisme de protecție în mod implicit împotriva atacurilor frecvent întâlnite în aplicațiile web, cum ar fi Cross-Site Scripting (XSS). Valorile template sunt automat auto-escapate, ceea ce înseamnă că caracterele potențial periculoase sunt transformate într-un format HTML sigur. În anumite situații, acest comportament poate fi dezactivat manual, dar dezvoltatorii ar trebui să evite acest lucru dacă nu sunt complet siguri de sursa datelor lor.

În ceea ce privește performanța, Jinja2 folosește un sistem de caching care stochează template-urile compilate în memorie, ceea ce permite procesarea eficientă și rapidă chiar și în aplicații cu un volum mare de trafic. Acest comportament poate fi configurat de Flask, care oferă opțiuni pentru cache în memorie sau pe disc, în funcție de configurația infrastructurii.

Un alt avantaj al Jinja2 este extensibilitatea, care permite dezvoltatorilor să adapteze comportamentul motorului la nevoile aplicației prin definirea de filtre personalizate, funcții globale, testere și extensii. De exemplu, un proiect care gestionează date financiare poate avea un filtru personalizat care formatează automat sumele monetare, în timp ce un alt proiect poate avea o funcție care traduce conținutul în funcție de limba utilizatorului.

Aplicații practice și relevanța Jinja2 în ecosistemul Python

Motorul Jinja2 este folosit în mod obișnuit în proiecte dezvoltate cu Flask, dar și în alte framework-uri sau platforme Python care necesită generarea de conținut HTML sau XML. Dashboard-urile, sistemele de management al conținutului, aplicațiile educaționale, interfețele administrative și platformele de e-commerce sunt toate proiecte în care este folosit. În afară de web, Jinja2 poate fi utilizat pentru a crea rapoarte, template-uri de e-mail sau fișiere de configurare PDF. Acest lucru se face prin conversia HTML înainte de a o transforma în formatul dorit. Această versatilitate îl face un motor de template foarte util pentru automatizări și aplicații cross-platform.

Relevanța sa în ecosistemul Python se datorează nu doar performanței și securității, ci și comunității largi care îl susține, documentației detaliate și compatibilității cu celelalte biblioteci majore. Prin urmare, stăpânirea Jinja2 devine o competență esențială pentru orice dezvoltator care activează în domeniul dezvoltării web cu Python.

Concluzii

Jinja2 oferă dezvoltatorilor un instrument solid pentru generarea dinamică a interfețelor web, combinând simplitatea cu puterea expresivă. Jinja2 susține principiile unei dezvoltări clare, structurate și eficiente prin arhitectura sa modulară și integrarea sa nativă în structuri precum Flask. Îl face esențial pentru dezvoltarea web contemporană, datorită capacității sale de a separa logicile aplicației de aspectul vizual și a mecanismelor sale de moștenire, filtrare și protecție. În cele din urmă, Jinja2 este mai mult decât un simplu motor de template; este o parte vitală a sistemului tehnologic care permite crearea de aplicații web scalabile, sigure și ușor de întreținut.

Exemplu

```
{% extends "base.html" %}

{% block content %}
<h1>{{ topic.title }}</h1>
<div class="posts">
    {% for post in posts %}
    <div class="post">
        <div class="post-header">
            <a href="{{ url_for('profile', username=post.author.username) }}">
                {{ post.author.username }}
            </a>
            <span class="text-muted">{{ post.created_at|time_since }}</span>
        </div>
        <div class="post-content">
            {{ post.content|format_content }}
        </div>
    </div>
    {% endfor %}
</div>
{% endblock %}
```

Figură 11 - Template Jinja2

Baza de Date SQLite – Arhitectură și Utilizare

Utilizare în Dev Forum

Aplicația folosește SQLite ca backend al bazei de date, stochând toate datele într-un singur fișier (forum.db) în directorul instanței.

Introducere

Adoptarea unui sistem de gestiune a bazelor de date este un pas important în dezvoltarea aplicațiilor software moderne, deoarece influențează performanța, scalabilitatea și mentenabilitatea proiectului pe termen lung. Sistemele de baze de date relaționale continuă să fie fundamentul stocării și prelucrării datelor structurate într-un mediu tehnologic caracterizat prin diversitate și inovație rapidă.

SQLite oferă o abordare complet diferită, bazată pe simplitate, portabilitate și integrare nativă în aplicație, în ciuda faptului că soluții precum PostgreSQL, MySQL și Microsoft SQL Server domină piața bazelor de date bazate pe server. Obiectivul acestei lucrări este de a examina în detaliu caracteristicile arhitecturale ale SQLite, modul său de funcționare, beneficiile și dezavantajele sale tehnice, precum și scenariile practice în care acest motor de baze de date se dovedește a fi cea mai bună opțiune.

Contextul istoric și conceptual al dezvoltării SQLite

D. Richard Hipp a dezvoltat SQLite în 2000 pentru a oferi un motor de baze de date relaționale care poate fi integrat direct în aplicații fără a avea nevoie de un server special sau configurare externă. Proiectarea sa a fost bazată pe dorința de a oferi un instrument care să respecte cât mai fidel standardul SQL în același timp, oferind în același timp o implementare simplă, de tip embedded, care funcționează „out-of-the-box" pe orice platformă.

SQLite a fost utilizat în milioane de aplicații, de la browsere web precum Firefox și Chrome până la sisteme de operare mobile precum Android și iOS, de-a lungul timpului. Această răspândire se datorează în mare parte atât fiabilității sale demonstrate în medii importante, cum ar fi aviația comercială, mașinile și aplicațiile industriale, cât și licenței publice care permite utilizarea gratuită.

Arhitectura internă și particularitățile de implementare

SQLite este o bibliotecă C care se integrează direct în codul sursă al aplicației, spre deosebire de bazele de date convenționale care funcționează pe model client-server. Prin urmare, întreaga bază de date este stocată într-un singur fișier pe disc, iar biblioteca face apeluri directe pentru a gestiona toate operațiunile, fără a necesita utilizarea unui proces server extern. Acest model de arhitectură face ca SQLite să fie o alegere excelentă pentru aplicații standalone, mobile, desktop sau embedded care au resurse hardware limitate sau probleme de conectivitate.

Pentru stocarea datelor, SQLite folosește formatul binar propriu, care este rezistent la corupere și portabil între arhitecturi. Sistemul de tranzacții folosește un mecanism bazat pe jurnalizare care susține atomicitatea, consistența, izolarea și durabilitatea (ACID). Acest lucru garantează că datele rămâne curate chiar și în cazul unei întreruperi neprevăzute ale aplicației. SQLite suportă atât modul rollback journal, cât și varianta Write-Ahead Logging (WAL), oferind dezvoltatorilor posibilitatea de a alege între performanță îmbunătățită și fiabilitate îmbunătățită, în funcție de contextul aplicației.

Funcționalități și compatibilitate cu standardul SQL

Deși are dimensiuni reduse, SQLite oferă o gamă largă de funcționalități care se aliniază în mare parte cu cerințele SQL-92, SQL-99 și parțial SQL:2003. Motorul poate gestiona diferite tipuri de date declarative, chei primare și străine, agregări, subinterogări, funcții de fereastră, expresii comune și constrângeri de unicitate. Abordarea dinamică a SQLite asupra tipurilor de date, cunoscută sub numele de „typelessness", permite stocarea valorilor de tip diferit în coloane declarate cu un tip specific, atâta timp cât nu sunt încălcate restricțiile definite.

În plus, SQLite susține complet tranzacțiile, care sunt percepute ca un set de operații invizibile care fie se finalizează cu succes, fie se anulează integral. În plus, sunt disponibile vizualizări, indicii și triggere. Un sistem de funcții definite de utilizator (UDF) facilitează extensibilitatea și permite crearea de extensii native în C sau alte limbaje compatibile. Integrarea cu limbajele de programare moderne, în special Python, este foarte bine documentată, având biblioteci oficiale și comunitare care oferă un strat de abstractizare în spatele API-ului nativ SQLite.

Avantaje, limitări și scenarii de utilizare

Avantajul principal al SQLite este ușurința sa de utilizare, ceea ce permite dezvoltatorilor să înceapă operațiunile imediat fără a efectua configurări sau instalări suplimentare. Este o opțiune excelentă pentru aplicații mobile, dispozitive IoT și desktop, datorită fiabilității sale ridicate, care a fost demonstrată prin utilizarea sa în sisteme critice. Resursele sunt eficiente și aplicațiile sunt distribuite ușor datorită dimensiunilor reduse ale bibliotecii, compatibilității cu mai multe platforme și lipsei unui server de bază de date extern. SQLite facilitează rapid backup-ul, migrarea și replicarea bazelor de date, deoarece toate datele sunt stocate într-un singur fișier.

Dar trebuie luate în considerare și limitările. Din cauza blocării la nivel de fișier, SQLite nu este conceput pentru medii concurente cu multe scrieri simultane. Prin urmare, performanța poate scădea în aplicații multi-user care necesită multă muncă manuală și concurență ridicată. În plus, SQLite nu este potrivit pentru aplicații enterprise unde securitatea granulară este esențială, deoarece nu are un sistem avansat de autentificare și control al accesului. De asemenea, unele funcționalități sofisticate ale standardului SQL, cum ar fi procedurile stocate și suportul nativ pentru tipuri de date binare complexe, nu au fost implementate în întregime.

SQLite este o soluție de stocare comună pentru aplicații mobile, în special pe platforma Android, unde funcționează ca bază de date locală implicită. Deoarece este ușor de configurat și rapid, SQLite este folosit frecvent în fazele de dezvoltare și testare ale aplicațiilor web. Aplicațiile de tip desktop, instrumente de dezvoltare, clienții de e-mail, browserele web și chiar platformele de testare automatizată a algoritmilor de învățare automată îl folosesc.

Comparativ cu alte sisteme de gestiune a bazelor de date

SQLite oferă un model de utilizare complet diferit față de sistemele de gestiune a bazelor de date convenționale precum MySQL și PostgreSQL. SQLite este ideal pentru aplicații standalone sau embedded, deoarece se concentrează pe simplitate și integrare rapidă, în timp ce primele se concentrează pe performanță și scalabilitate în medii multi-user și multi-instanță. SQLite funcționează în cadrul procesului aplicației, eliminând latența comunicării inter-proces și permitând o manipulare directă și rapidă a datelor. Acest lucru îl diferențiază de bazele de date bazate pe server, care necesită conectarea prin protocoale de rețea și gestionarea separată a instanțelor.

SQLite are un avantaj semnificativ în ceea ce privește consumul de resurse, deoarece necesită puțin spațiu de stocare, memorie și daemon de fundal. Totuși, alte SGDB-uri pot fi mai bune pentru aplicații care necesită replicare distribuită, control strict al permisiunilor sau executarea de operațiuni complexe în paralel.

Concluzii

SQLite oferă o soluție simplă, robustă și performantă pentru gestionarea datelor structurate într-o varietate de contexte și se dovedește a fi un instrument extrem de util în arsenalul dezvoltatorului modern. SQLite îndeplinește majoritatea cerințelor fundamentale fără a adăuga complexitate suplimentară, fie că este utilizat în aplicații mobile, soluții embedded sau ca mediu de prototipare. Pentru situațiile în care eficiența, portabilitatea și simplitatea sunt importante, este o opțiune utilă.

Acest lucru se datorează designului său simplu, compatibilității sale cu standardele SQL și fiabilității demonstrate în mii de proiecte reale. Cu toate acestea, este esențial să înțelegem limitările acestei tehnologii și să alegem contextul de utilizare corect. În cele din urmă, SQLite continuă să fie o parte vitală a ecosistemului software modern și face un mare efort pentru a face tehnologia de gestiune a datelor mai accesibilă tuturor.

Exemplu

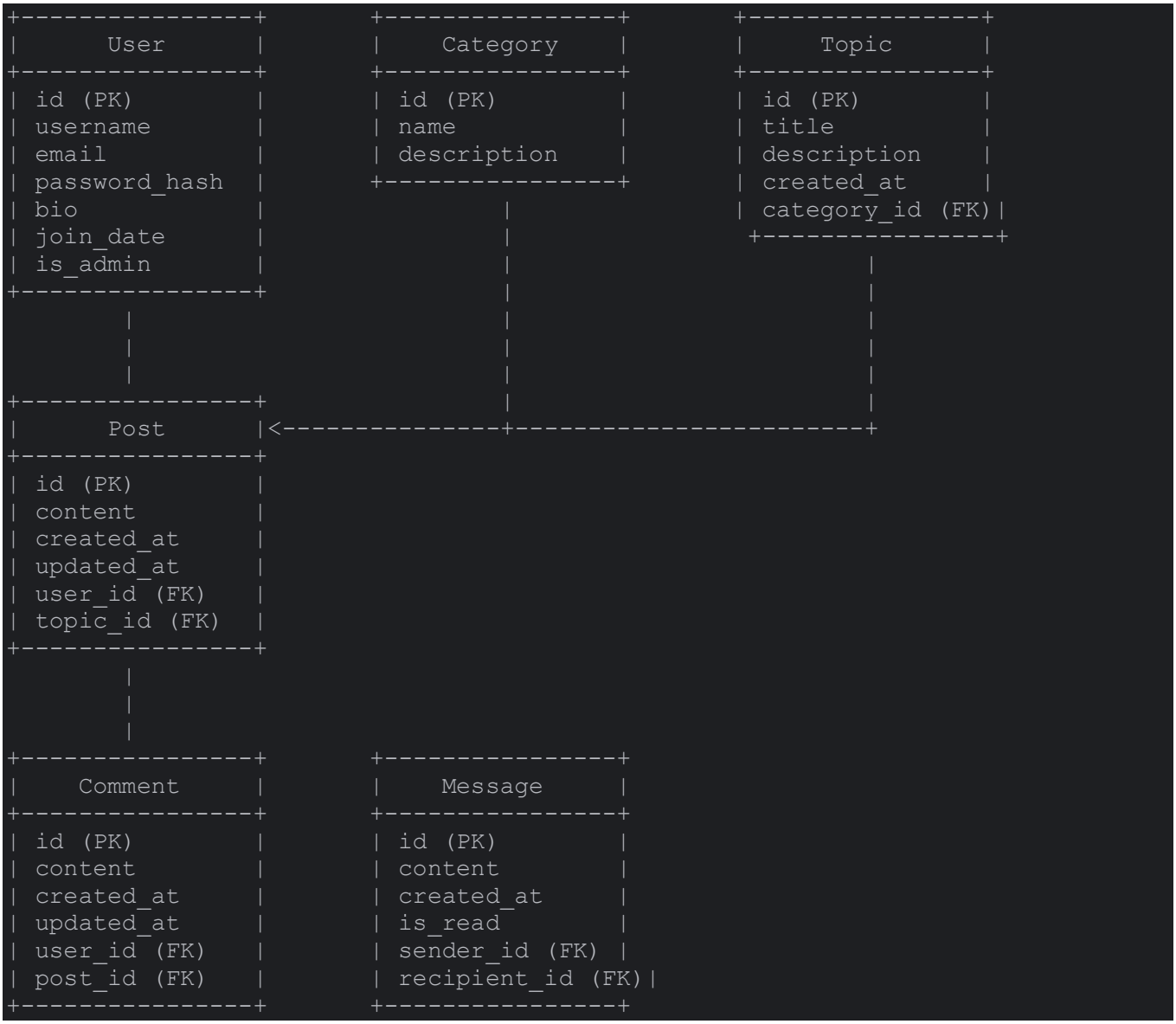
```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///forum.db'

# Initialize the database
with app.app_context():
    db.create_all()
```

Figură 12 - Database URI

Capitol 3. Descrierea aplicației
3.1. Arhitectura bazei de date SQLite

Structura bazei de date



Figură 13 - Relationarea Bazei de Date

Relaționarea in baza de date
Tabela Utilizatorilor

```sql
-- Users Table
CREATE TABLE user (
    id INTEGER PRIMARY KEY,
    username VARCHAR(80) UNIQUE NOT NULL,
    email VARCHAR(120) UNIQUE NOT NULL,
    password_hash VARCHAR(128),
    bio TEXT,
    join_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    is_admin BOOLEAN DEFAULT FALSE
);
CREATE INDEX idx_user_username ON user(username);
CREATE INDEX idx_user_email ON user(email);
CREATE INDEX idx_user_join_date ON user(join_date);
```

Figură 14 - Tabela USERS

Tabela Categoriilor

```sql
-- Categories Table
CREATE TABLE category (
    id INTEGER PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT
);
CREATE INDEX idx_category_name ON category(name);
```

Figură 15 - Tabela Categoriilor

Tabela Subiectelor

```
-- Topics Table
CREATE TABLE topic (
    id INTEGER PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    description TEXT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    category_id INTEGER NOT NULL,
    FOREIGN KEY (category_id) REFERENCES category (id)
);
CREATE INDEX idx_topic_title ON topic(title);
CREATE INDEX idx_topic_created_at ON topic(created_at);
CREATE INDEX idx_topic_category_id ON topic(category_id);
```

Figură 16 - Tabela Subiectelor

Tabela Postărilor

```
-- Posts Table
CREATE TABLE post (
    id INTEGER PRIMARY KEY,
    content TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    user_id INTEGER NOT NULL,
    topic_id INTEGER NOT NULL,
    FOREIGN KEY (user_id) REFERENCES user (id),
    FOREIGN KEY (topic_id) REFERENCES topic (id)
);
CREATE INDEX idx_post_created_at ON post(created_at);
CREATE INDEX idx_post_user_id ON post(user_id);
CREATE INDEX idx_post_topic_id ON post(topic_id);
```

Figură 17 - Tabela Postărilor

Tabela Comentariilor

```sql
-- Comments Table
CREATE TABLE comment (
    id INTEGER PRIMARY KEY,
    content TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    user_id INTEGER NOT NULL,
    post_id INTEGER NOT NULL,
    FOREIGN KEY (user_id) REFERENCES user (id),
    FOREIGN KEY (post_id) REFERENCES post (id)
);
CREATE INDEX idx_comment_created_at ON comment(created_at);
CREATE INDEX idx_comment_user_id ON comment(user_id);
CREATE INDEX idx_comment_post_id ON comment(post_id);
```

Figură 18 - Tabela Comentariilor

Tabela Mesajelor

```sql
-- Messages Table
CREATE TABLE message (
    id INTEGER PRIMARY KEY,
    content TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    is_read BOOLEAN DEFAULT FALSE,
    sender_id INTEGER NOT NULL,
    recipient_id INTEGER NOT NULL,
    FOREIGN KEY (sender_id) REFERENCES user (id),
    FOREIGN KEY (recipient_id) REFERENCES user (id)
);
CREATE INDEX idx_message_created_at ON message(created_at);
CREATE INDEX idx_message_sender_id ON message(sender_id);
CREATE INDEX idx_message_recipient_id ON message(recipient_id);
```

Figură 19 - Tabela Mesajelor

3.2. Arhitectura aplicației

Structura proiectului

```
dev-forum/
|-- app.py                  # Main application file
|-- requirements.txt        # Python dependencies
|-- instance/               # Instance-specific data
|   |-- forum.db            # SQLite database
|-- static/                 # Static files
|   |-- css/
|   |   |-- style.css       # Custom CSS styles
|   |-- js/
|       |-- script.js       # Custom JavaScript
|-- templates/              # HTML templates
    |-- admin.html          # Admin dashboard
    |-- base.html           # Base template with layout
    |-- category.html       # Category view
    |-- edit_profile.html   # Profile editing
    |-- errors/             # Error pages
    |   |-- 403.html        # Forbidden error
    |   |-- 404.html        # Not found error
    |   |-- 500.html        # Server error
    |-- index.html          # Home page
    |-- login.html          # Login page
    |-- new_category.html   # Create category
    |-- messages.html       # List of conversations
    |-- conversation.html   # Single conversation view
    |-- new_message.html    # Create new message
    |-- new_post.html       # Create post
    |-- new_topic.html      # Create topic
    |-- profile.html        # User profile
    |-- register.html       # Registration page
    |-- search.html         # Search results
    |-- topic.html          # Topic view
```

Figură 20 - Structura Proiectului

Fișiere cheie si funcționalitatea lor

Introducere

Organizarea logică a codului sursă este esențială pentru scalabilitate, mentenanță și cooperare eficientă în echipele de dezvoltare în dezvoltarea aplicațiilor web moderne. Deoarece este renumit pentru simplitatea și flexibilitatea sa, framework-ul Flask permite dezvoltatorilor să construiască proiecte în mod modular, adaptabil la diferite scenarii de complexitate. Anumite fișiere se remarcă în această arhitectură prin rolul lor vital în gestionarea logicii aplicației, prezentării și interactivității.

Scopul acestei lucrări este de a examina în detaliu cele patru componente esențiale ale fișierelor-cheie într-un proiect Flask: app.py, templates/base.html, static/css/style.css și static/js/script.js. Vom discuta

modul în care aceste fișiere colaborează pentru a crea o aplicație coerentă, interactivă și extensibilă, reflectând în același timp bunele practici în ingineria software contemporană.

app.py

Fișierul app.py servește drept centrul de comandă al unei aplicații Flask și conține toate elementele esențiale ale logicii și infrastructurii de bază ale aplicației. Acest fișier este responsabil pentru inițierea instanței aplicației Flask. Aceasta servește drept punct de intrare pentru orice cerere HTTP pe care serverul o procesează. Definirea configurațiilor esențiale, cum ar fi setările de conectare la baza de date, cheile de securitate și parametrii de funcționare în zonele de dezvoltare și producție, este parte a inițializării.

În afară de configurație, app.py oferă definirea modelelor de date și folosește de obicei un ORM precum SQLAlchemy, ceea ce permite maparea directă a claselor Python la tabelele bazei de date. Aceste modele permit o interacțiune simplă și sigură cu structurile relaționale și sunt esențiale pentru abstractizarea logicii de acces la date.

Definirea rutelor aplicației este o funcție importantă a acestui fișier, deoarece acestea stabilesc legătura între funcțiile specifice din cod și URL-urile pe care utilizatorii le accesează. În funcție de contextul aplicației, aceste funcții gestionează cererile, procesează datele și returnează răspunsuri HTML, JSON sau redirecționări. În plus, app.py poate conține filtre personalizate, ceea ce îmbunătățește capacitățile motorului de template Jinja2 și permite manipularea avansată a datelor direct în șabloane HTML.

Tot aici sunt incluse și handler-ele pentru erori, care gestionează elegant situații excepționale, cum ar fi erorile 404 sau 500, afișând pagini personalizate și oferind o experiență coerentă utilizatorului. În plus, funcțiile speciale ale procesorului de context facilitează reutilizarea și centralizarea logicii comune de afișare prin inserarea variabilelor globale în toate template-urile.

base.html

Structura fundamentală a interfeței grafice a aplicației este stabilită în fișierul templates/base.html, care servește drept șablon de bază pentru toate celelalte pagini ale sistemului. Moștenirea și suprascrierea secțiunilor specifice ale documentului este posibilă cu motorul de template Jinja2, care asigură o arhitectură coerentă și ușor de întreținut.

Printre elementele centrale definite în **base.html** se regăsește bara de navigație, care oferă utilizatorului acces rapid la secțiunile principale ale aplicației. Aceasta este adesea generată dinamic, în funcție de starea de autentificare sau de rolul utilizatorului, reflectând astfel principiile personalizării interfeței.

Gestionarea mesajelor temporare, cunoscute și sub numele de mesaje flash, care informează utilizatorul despre succesul sau eșecul anumitor acțiuni este un alt aspect important. Acestea sunt afișate într-un mod uniform pe fiecare pagină și sunt stilizate corespunzător folosind clasele CSS din fișierul relevant.
Secțiunea de jos a paginii, care este definită și în file-ul base.html, închide structura paginii și poate include informații despre autor, linkuri utile sau scripturi externe. În plus, includerea în acest șablon a fișierelor CSS și JavaScript comune este esențială, deoarece ajută la o încărcare mai eficientă a resurselor în întregul sistem. Prin urmare, baza.html servește ca nucleu al prezentării aplicației, asigurând că toate paginile componente sunt uniforme atât vizual, cât și funcțional.

style.css

Fișierul static /css/style. CSS se concentrează pe gestionarea aspectului vizual al aplicației, îmbunătățind și extinzând funcționalitatea framework-urilor CSS externe, cum ar fi Bootstrap. Acest fișier joacă un rol structural, mai degrabă decât estetic, descriind modul în care componentele se comportă în raport cu diferitele dimensiuni de ecran, stări ale interfeței și moduri de afișare.

Stilizarea personalizată a cardurilor de conținut, care delimitează vizual și ierarhizează informațiile afișate, este un exemplu comun. În plus, acest fișier gestionează aspectul elementelor de formular, secțiunilor de postări și comentariilor, oferind aplicației o identitate vizuală distinctă.

Pentru a garanta afișarea optimă a interfeței pe dispozitive cu ecrane de dimensiuni diferite, principiile de responsive design sunt esențiale. Ajustările în style.css au fost făcute pentru a păstra funcționalitatea și aspectul aplicației, indiferent dacă utilizatorul vizitează site-ul de pe un desktop, tabletă sau telefon mobil.

Suportul pentru modurile întunecate este un alt element esențial, care răspunde tendințelor actuale de design și oferă utilizatorului posibilitatea de a schimba între moduri vizuale contrastante. Clasele și regulile suplimentare CSS care schimbă paletele de culori în funcție de preferințele salvate sau alese permit utilizarea acestei caracteristici.

script.js

fișierul static /js/script. JS gestionează aspectele logice dinamice și interacționare a interfeței, completând funcționalitatea statică a HTML-ului și CSS prin mecanisme controlate cu limbajul JavaScript. Acest fișier include funcții care îmbunătățesc experiența utilizatorului prin reacții în timp real și prin reducerea necesității reîncărcării paginilor.
Un exemplu esențial este utilizarea butonului de comutare între modurile vizuale ușoare și negre. Acest buton funcționează împreună cu stilurile CSS și permite modificarea interfeței în timp real, în funcție de modul în care utilizatorul interacționează. Toate funcționalitățile de validare a formularelor sunt utilizate în acest fișier pentru a se asigura că datele care sunt introduse sunt conforme cu cerințele aplicației încă din faza de interacțiune frontală.

Alte caracteristici importante includ previzualizarea în timp real a conținutului scris în sintaxa Markdown, redimensionarea automată a zonelor de tip textarea, astfel încât utilizatorul să aibă parte de un câmp adaptabil volumului de text introdus. Aceasta din urmă este deosebit de relevantă pentru aplicațiile care permit crearea de conținut bogat și oferă utilizatorului feedback instantaneu asupra rezultatelor scrierii.
O caracteristică avansată este includerea funcțiilor AJAX care permit trimiterea de comentarii sau actualizarea conținutului fără a reîncărca pagina întreagă. Această funcționalitate permite crearea unei aplicații interactive, fluide în care utilizatorul primește confirmări și răspunsuri în timp real, fără întreruperi vizibile în fluxul de utilizare.

Concluzii

Structura fișierelor unei aplicații Flask arată o separare clară a preocupărilor, în conformitate cu standardele arhitecturii software moderne. Fiecare fișier-cheie îndeplinește o funcție specifică în ceea ce privește funcționarea sistemului: app.py gestionează logica și configurarea de bază, base.html stabilește bazele interfeței vizuale și style. Când vine vorba de stil și adaptabilitate, script.js oferă interacțiune și dinamism. Pentru a oferi o aplicație coerentă, scalabilă și orientată către utilizator, aceste componente funcționează împreună armonios. Înțelegerea detaliată a fiecărei componente ajută la crearea de aplicații funcționale, precum și la consolidarea unui mod de gândire ingineresc care se concentrează pe eficiență, modularitate și claritate.

3.3. Arhitectura API

Un set larg de puncte finale API disponibile în aplicația Dev Forum facilitează interacțiunile utilizatorilor și funcțiile administrative. Aceste puncte finale sunt implementate folosind sistemul de rutare Flask și sunt organizate în categorii logice în funcție de funcționalitatea lor. O analiză detaliată a fiecărui punct final, care include scopul, detaliile de implementare și exemple de cod din baza de cod a aplicației, este oferită în documentația următoare.

Sistemul de Autentificare

Sistemul de autentificare formează fundamentul managementului identității utilizatorilor în cadrul aplicației, oferind mecanisme pentru înregistrarea utilizatorului, autentificare și terminarea sesiunii.

ENDPOINT-ul de înregistrare a utilizatorului

ENDPOINT-ul de înregistrare facilitează crearea de noi conturi de utilizator în cadrul sistemului. Acest punct final acceptă atât cererile *GET*, cât și *POST*, unde cererile *GET* redau formularul de înregistrare, iar cererile *POST* procesează datele utilizatorului trimise.

```python
@app.route( rule: '/register', methods=['GET', 'POST'])  6 usages  TonyDN98
def register():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
    if request.method == 'POST':
        username = request.form.get('username')
        email = request.form.get('email')
        password = request.form.get('password')
        user_exists = User.query.filter_by(username=username).first()
        email_exists = User.query.filter_by(email=email).first()
        if user_exists:
            flash('Username already exists.')
            return redirect(url_for('register'))
        if email_exists:
            flash('Email already exists.')
            return redirect(url_for('register'))
        new_user = User(username=username, email=email)
        new_user.set_password(password)
        db.session.add(new_user)
        db.session.commit()
        flash('Registration successful! Please log in.')
        return redirect(url_for('login'))
    return render_template('register.html')
```

Figură 21 - Ruta de /register

Implementarea folosește capacitățile de gestionare a sesiunii ale Flask-Login pentru a păstra starea de autentificare a utilizatorului în fiecare cerere. După autentificarea cu succes, sistemul acceptă și redirecționarea către paginile solicitate inițial, îmbunătățind experiența utilizatorului prin păstrarea contextului de navigare.

ENDPOINT-ul de conectare

ENDPOINT-ul de conectare gestionează autentificarea utilizatorilor prin validarea acreditărilor și stabilirea sesiunilor de utilizator. Similar cu punctul final de înregistrare, acesta gestionează atât cererile ***GET*** pentru redarea formularului de conectare, cât și cererile ***POST*** pentru procesarea încercărilor de autentificare.

```python
@app.route( rule: '/login', methods=['GET', 'POST']) 7 usages  TonyDN98
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        user = User.query.filter_by(username=username).first()
        if user and user.check_password(password):
            login_user(user)
            next_page = request.args.get('next')
            return redirect(next_page or url_for('index'))
        flash('Invalid username or password')
        return redirect(url_for('login'))
    return render_template('login.html')
```

Figură 22 - Ruta de /login

Implementarea folosește capacitățile de gestionare a sesiunii ale Flask-Login pentru a păstra starea de autentificare a utilizatorului în fiecare cerere. După autentificarea cu succes, sistemul acceptă și redirecționarea către paginile solicitate inițial, îmbunătățind experiența utilizatorului prin păstrarea contextului de navigare.

ENDPOINT-ul de deconectare

ENDPOINT-ul de deconectare oferă utilizatorilor un mecanism de a-și termina sesiunile active, deconectandu-le efectiv de la aplicație.

```python
@app.route('/logout')  1 usage  TonyDN98
@login_required
def logout():
    logout_user()
    return redirect(url_for('index'))
```

Figură 23 - Ruta de /logout

Acest punct final este protejat de decoratorul ***@login_required***, asigurându-se că numai utilizatorii autentificați îl pot accesa. La invocare, utilizează funcția ***logout_user()*** de la *Flask-Login* pentru a șterge datele de sesiune ale utilizatorului și pentru a redirecționa către pagina de pornire a aplicației.

Sistemul de management al profilului utilizatorului

Sistemul de management al profilului permite utilizatorilor să vizualizeze și să modifice informațiile personale în cadrul aplicației.

ENDPOINT-ul de vizualizare a profilului

ENDPOINT-ul de vizualizare a profilului afisează profilul public al unui utilizator, afișând informațiile biografice și istoricul activității acestuia.

```
/profile/{username}
@app.route('/profile/<username>')  TonyDN98
def profile(username):
    user = User.query.filter_by(username=username).first_or_404()
    posts = Post.query.filter_by(user_id=user.id).order_by(Post.created_at.desc()).all()
    return render_template( template_name_or_list: 'profile.html', user=user, posts=posts)
```

[Ruta de /profile/user]

Informațiile utilizatorului ales și istoricul postărilor lor, sortate în ordine cronologică inversă, sunt colectate din baza de date de implementare. Dacă numele de utilizator solicitat nu există, sistemul generează o eroare 404 și oferă utilizatorului informații adecvate.

ENDPOINT-ul de editare a profilului

Acest ENDPOINT-ul permite utilizatorilor să își modifice informațiile de profil, inclusiv detaliile *biografice* și *parola*.

```
@app.route( rule: '/profile/edit', methods=['GET', 'POST'])  5 usages  TonyDN98
@login_required
def edit_profile():
    if request.method == 'POST':
        bio = request.form.get('bio')
        current_user.bio = bio
        # Check if password change was requested
        current_password = request.form.get('current_password')
        new_password = request.form.get('new_password')
        confirm_password = request.form.get('confirm_password')
        if current_password and new_password and confirm_password:
            if not current_user.check_password(current_password):
                flash('Current password is incorrect.')
                return redirect(url_for('edit_profile'))
            if new_password != confirm_password:
                flash('New passwords do not match.')
                return redirect(url_for('edit_profile'))
            current_user.set_password(new_password)
            flash('Password updated successfully.')
        db.session.commit()
        flash('Profile updated successfully.')
        return redirect(url_for( endpoint: 'profile', username=current_user.username))
    return render_template('edit_profile.html')
```

Figură 24 - Ruta de editare a profilului

Implementarea permite validarea completă a modificărilor parolei; utilizatorul trebuie să furnizeze parola actuală pentru a fi verificat și apoi să confirme noua parolă pentru a evita erorile de tipar. Acest proces de validare în mai mulți pași sporește securitatea, asigurându-se că numai proprietarul legitim al contului poate modifica datele sensibile ale contului.

Sistemul de management al conținutului forumului

Sistemul de gestionare a conținutului forumului cuprinde puncte finale pentru vizualizarea și crearea de categorii, subiecte, postări și comentarii, formând funcționalitatea de bază a platformei de discuții.

ENDPOINT-ul Paginii de pornire

ENDPOINT-ul paginii de pornire servește ca punct de intrare în aplicație, afișând toate categoriile de discuții disponibile împreună cu statisticile forumului.

```
@app.route('/')  TonyDN98
def index():
    categories = Category.query.all()
    topic_count = Topic.query.count()
    user_count = User.query.count()
    return render_template( template_name_or_list: 'index.html', categories=categories, topic_count=topic_count, user_count=user_count)
```

Figură 25 - Ruta HomePage

Această aplicație recuperează toate categoriile din baza de date și generează statisticile forumului, care includ numărul total de subiecte și utilizatori înregistrați. Acest lucru oferă vizitatorilor o înțelegere a activității și a domeniului forumului.

ENDPOINT-ul vizualizare a categoriilor

ENDPOINT-ul afișează subiecte dintr-o anumită categorie, susținând paginarea pentru o navigare eficientă prin colecții mari de subiecte.

```
/category/{category_id}
@app.route('/category/<int:category_id>')  TonyDN98
/category/{category_id}/page/{page}
@app.route('/category/<int:category_id>/page/<int:page>')
@cache.cached(timeout=60, query_string=True)
def category(category_id, page=1):
    category = Category.query.get_or_404(category_id)
    topics = Topic.query.filter_by(category_id=category_id).order_by(Topic.created_at.desc()).paginate(
        page=page, per_page=app.config['TOPICS_PER_PAGE'], error_out=False)
    return render_template( template_name_or_list: 'category.html', category=category, topics=topics)
```

Figură 26 - Ruta de vizualizare categorii

Implementarea limitează numărul de subiecte afișate pe pagină folosind funcționalitatea de paginare a Flask-SQLAlchemy, îmbunătățind performanța și experiența utilizatorului. În plus, Flask-Caching reduce încărcarea bazei de date pentru categoriile frecvent accesate, stocând pagina redată în cache timp de 60 de secunde.

ENDPOINT-ul de creare a subiectului

ENDPOINT-ul permite utilizatorilor autentificați să creeze noi subiecte de discuție într-o anumită categorie.

```python
@app.route( rule: '/topic/new/<int:category_id>', methods=['GET', 'POST']) 3 usages  TonyDN98
@login_required
def new_topic(category_id):
    category = Category.query.get_or_404(category_id)
    if request.method == 'GET':
        flash('Tip: To insert code, wrap your code in triple backticks with a language name, e.g. ```python for Python code.')
    if request.method == 'POST':
        title = request.form.get('title')
        description = request.form.get('description')
        content = request.form.get('content')
        if not title or not content:
            flash('Title and content are required.')
            return redirect(url_for( endpoint: 'new_topic', category_id=category_id))
        topic = Topic(title=title, description=description, category_id=category_id)
        db.session.add(topic)
        db.session.commit()
        post = Post(content=content, user_id=current_user.id, topic_id=topic.id)
        db.session.add(post)
        db.session.commit()
        return redirect(url_for( endpoint: 'topic', topic_id=topic.id))
    return render_template( template_name_or_list: 'new_topic.html', category=category)
```

Figură 27 - Ruta de creare subiect

Implementarea implică verificarea formularelor pentru a se asigura că conținutul și titlul sunt corecte. În plus, atunci când redați formularul, vă oferă sfaturi utile despre formatarea fragmentelor de cod. Sistemul menține consistența datelor creând atât un subiect nou, cât și o postare inițială într-o singură tranzacție după transmiterea cu succes.

ENDPOINT-ul de vizualizare a subiectului

ENDPOINT-ul afișează postări dintr-un anumit subiect, susținând paginarea pentru o navigare eficientă prin discuții îndelungate.

```python
@app.route('/topic/<int:topic_id>')  TonyDN98
/topic/{topic_id}/page/{page}
@app.route('/topic/<int:topic_id>/page/<int:page>')
@cache.cached(timeout=30, query_string=True)
def topic(topic_id, page=1):
    topic = Topic.query.get_or_404(topic_id)
    posts = Post.query.filter_by(topic_id=topic_id).order_by(Post.created_at).paginate(
        page=page, per_page=app.config['POSTS_PER_PAGE'], error_out=False)
    return render_template( template_name_or_list: 'topic.html', topic=topic, posts=posts)
```

Figură 28 - Ruta de vizualizare subiect

Această implementare utilizează paginarea și stocarea în cache pentru a optimiza performanța, la fel ca și punctul final de vizualizare a categoriilor. Dar expirarea memoriei cache este mai scurtă – 30 de secunde în loc de 60 de secunde – pentru a asigura că utilizatorii văd conținut relativ proaspăt în discuțiile active.

ENDPOINT-ul de creare a postării

ENDPOINT-ul permite utilizatorilor autentificați să răspundă la subiectele existente prin crearea de noi postări.

```python
@app.route( rule: '/post/new/<int:topic_id>', methods=['GET', 'POST']) 3 usages  TonyDN98
@login_required
def new_post(topic_id):
    topic = Topic.query.get_or_404(topic_id)
    if request.method == 'GET':
        flash('Tip: To insert code, wrap your code in triple backticks with a language name, e.g. ```python for Python code.')
    if request.method == 'POST':
        content = request.form.get('content')
        if not content:
            flash('Content is required.')
            return redirect(url_for( endpoint: 'new_post', topic_id=topic_id))
        post = Post(content=content, user_id=current_user.id, topic_id=topic_id)
        db.session.add(post)
        db.session.commit()
        return redirect(url_for( endpoint: 'topic', topic_id=topic_id))
    return render_template( template_name_or_list: 'new_post.html', topic=topic)
```

Figură 29 - Ruta de creare postare

Implementarea implică furnizarea de recomandări de formatare similare cu punctul final de creare a subiectului și validarea pentru a se asigura că conținutul postării este furnizat. După ce mesajul este trimis cu succes, utilizatorii sunt redirecționați la vizualizarea subiectului, unde pot vedea postarea lor în contextul întregii discuții.

ENDPOINT-ul de creare a comentariilor

ENDPOINT-ul permite utilizatorilor autentificați să adauge comentarii la anumite postări, acceptând atât trimiterile de formulare tradiționale, cât și solicitările AJAX pentru o experiență îmbunătățită a utilizatorului.

```python
@app.route( rule: '/comment/new/<int:post_id>', methods=['POST']) 1 usage  TonyDN98 *
@login_required
def new_comment(post_id):
    post = Post.query.get_or_404(post_id)
    content = request.form.get('content')
    if not content:
        flash('Comment content is required.')
        return redirect(url_for( endpoint: 'topic', topic_id=post.topic_id))
    comment = Comment(content=content, user_id=current_user.id, post_id=post_id)
    db.session.add(comment)
    db.session.commit()
    # Check if request is AJAX
    if request.headers.get('X-Requested-With') == 'XMLHttpRequest':
        # Return JSON response for AJAX requests
        return {
            'id': comment.id,
            'content': comment.content,
            'created_at': comment.created_at.strftime('%Y-%m-%d %H:%M:%S'),
            'time_since': app.jinja_env.filters['time_since'](comment.created_at),
            'author': {
                'id': current_user.id,
                'username': current_user.username,
                'profile_url': url_for( endpoint: 'profile', username=current_user.username)
            },
            'formatted_content': app.jinja_env.filters['format_content'](comment.content)
        }
    # Return normal redirect for non-AJAX requests
    return redirect(url_for( endpoint: 'topic', topic_id=post.topic_id))
```

Figură 30 - Ruta de creare a comentariilor

Implementarea identifică cererile AJAX folosind antetul X-Requested-With și oferă răspunsuri potrivite în funcție de tipul cererii. Pentru solicitările AJAX, returnează un obiect JSON care conține datele noului comentariu. Acest lucru permite clientului să redare fără a reîncărca pagina. Redirecționează înapoi la vizualizarea subiectului pentru cerințele tradiționale.

Sistemul de căutare

Sistemul de căutare permite utilizatorilor să găsească conținut relevant pe forum interogând subiecte și postări.

ENDPOINT-ul de căutare

ENDPOINT-ul procesează interogările de căutare și returnează subiecte și postări care se potrivesc.

```python
@app.route('/search')  1 usage  TonyDN98
def search():
    query = request.args.get('query', '')
    if not query:
        return render_template( template_name_or_list: 'search.html', results=None)
    topics = Topic.query.filter(Topic.title.contains(query)).all()
    posts = Post.query.filter(Post.content.contains(query)).all()
    return render_template( template_name_or_list: 'search.html', query=query, topics=topics, posts=posts)
```

Figură 31 - Ruta de cautare

Implementarea potrivește subșirurile cu titlurile postărilor și conținutul. Dacă nicio interogare nu este furnizată, șablonul de căutare rămâne fără rezultate. Atunci când este trimisă o întrebare, aceasta preia și afișează toate postările și subiectele care se potrivesc, oferind utilizatorilor rezultate de căutare detaliate.

Sistemul de mesagerie

Sistemul de mesagerie facilitează comunicarea privată între utilizatori, susținând atât listarea conversațiilor, cât și schimburile individuale de mesaje.

ENDPOINT-ul listei de conversații

ENDPOINT-ul afișează toate conversațiile la care participă utilizatorul curent, sortate după cea mai recentă activitate.

```python
@app.route('/messages')  5 usages  TonyDN98 *
@login_required
def messages():
    # Get all users that the current user has conversations with
    sent_to_users = db.session.query(User).join(Message, User.id == Message.recipient_id).filter(Message.sender_id == current_user.id).disti
    received_from_users = db.session.query(User).join(Message, User.id == Message.sender_id).filter(Message.recipient_id == current_user.id)
    # Combine the lists and remove duplicates
    conversation_users = list(set(sent_to_users + received_from_users))
    # Get the last message for each conversation
    conversations = []
    for user in conversation_users:
        # Get the last message between the current user and this user
        last_message = Message.query.filter(
            ((Message.sender_id == current_user.id) & (Message.recipient_id == user.id)) |
            ((Message.sender_id == user.id) & (Message.recipient_id == current_user.id))
        ).order_by(Message.created_at.desc()).first()
        # Count unread messages
        unread_count = Message.query.filter_by(
            sender_id=user.id,
            recipient_id=current_user.id,
            is_read=False
        ).count()
        conversations.append({
            'user': user,
            'last_message': last_message,
            'unread_count': unread_count
        })
    # Sort conversations by the timestamp of the last message
    conversations.sort(key=lambda x: x['last_message'].created_at if x['last_message'] else datetime.min, reverse=True)
    return render_template( template_name_or_list: 'messages.html', conversations=conversations)
```

Figură 32 - Ruta de listare a conversatiilor

Implementarea face interogări complicate în bazele de date pentru a găsi fiecare utilizator cu care utilizatorul actual a schimbat mesajele. Preia cel mai recent mesaj din fiecare conversație și numără mesajele care nu au fost citite, oferind utilizatorilor o imagine completă a activității lor de mesagerie.

ENDPOINT-ul de vizualizare a conversației și de răspuns

ENDPOINT-ul afișează istoricul mesajelor dintre utilizatorul actual și alt utilizator și îi permite utilizatorului curent să trimită mesaje noi în conversație.

```python
@app.route( rule: '/messages/<username>', methods=['GET', 'POST'])  4 usages  TonyDN98 *
@login_required
def conversation(username):
    user = User.query.filter_by(username=username).first_or_404()
    if request.method == 'POST':
        content = request.form.get('content')
        if content:
            message = Message(
                content=content,
                sender_id=current_user.id,
                recipient_id=user.id
            )
            db.session.add(message)
            db.session.commit()
            flash('Message sent.')
            return redirect(url_for( endpoint: 'conversation', username=username))
    # Get all messages between the current user and the other user
    messages = Message.query.filter(
        ((Message.sender_id == current_user.id) & (Message.recipient_id == user.id)) |
        ((Message.sender_id == user.id) & (Message.recipient_id == current_user.id))
    ).order_by(Message.created_at).all()
    # Mark unread messages as read
    unread_messages = Message.query.filter_by(
        sender_id=user.id,
        recipient_id=current_user.id,
        is_read=False
    ).all()
    for message in unread_messages:
        message.is_read = True
    db.session.commit()
    return render_template( template_name_or_list: 'conversation.html', user=user, messages=messages)
```

Figură 33 - Ruta de vizualizare si raspuns a unei conversatii

Toate mesajele care au fost schimbate între cei doi utilizatori sunt preluate de implementare, sortate cronologic. Oferind o experiență de utilizator fără întreruperi, marchează automat toate mesajele necitite atunci când vizualizați o conversație. Creează și stochează mesaje noi pentru solicitările POST, permitând comunicarea în timp real.

ENDPOINT-ul unei noi conversaţii

ENDPOINT-ul permite utilizatorilor să inițieze noi conversații cu alți utilizatori.

```
@app.route( rule: '/messages/new/<username>', methods=['GET', 'POST'])  2 usages  TonyDN98
@login_required
def new_message(username):
    user = User.query.filter_by(username=username).first_or_404()

    if request.method == 'POST':
        content = request.form.get('content')
        if content:
            message = Message(
                content=content,
                sender_id=current_user.id,
                recipient_id=user.id
            )
            db.session.add(message)
            db.session.commit()
            flash('Message sent.')
            return redirect(url_for( endpoint: 'conversation', username=username))

    return render_template( template_name_or_list: 'new_message.html', user=user)
```

Figură 34 - Ruta de creare a unei noi conversatii

Implementarea este o funcție asemănătoare funcționalității de răspuns la conversație, dar este special conceput pentru a începe conversații noi. Pentru a menține o experiență de utilizator consistentă, utilizatorii sunt redirecționați la vizualizarea conversației obișnuite după trimiterea primului mesaj.

Sistemul Administrativ

Sistemul administrativ oferă utilizatorilor autorizați instrumente pentru gestionarea forumului, inclusiv supravegherea utilizatorilor și crearea categoriilor.

ENDPOINT-ul Admin Dashboard

ENDPOINT-ul afișează o prezentare generală a stării forumului și oferă acces la funcțiile administrative.

```python
@app.route('/admin')  4 usages  TonyDN98
@login_required
def admin():
    if not current_user.is_admin:
        flash('You do not have permission to access this page.')
        return redirect(url_for('index'))
    users = User.query.all()
    categories = Category.query.all()
    topic_count = Topic.query.count()
    post_count = Post.query.count()
    return render_template( template_name_or_list: 'admin.html', users=users, categories=categories, topic_count=topic_count,
```

Figură 35 - Ruta de admin dashboard

Implementarea implică verificarea permisiunilor pentru a se asigura că această funcționalitate este accesibilă doar administratorilor. Acesta colectează statistici complete despre forum și informații despre utilizatori, oferind administratorilor datele necesare pentru administrarea forumului eficient.

ENDPOINT-ul de creare a categoriilor

ENDPOINT-ul permite administratorilor să creeze noi categorii de discuții.

```python
@app.route( rule: '/admin/category/new', methods=['GET', 'POST'])@login_required  4 usages  TonyDN98
def new_category():
    if not current_user.is_admin:
        flash('You do not have permission to access this page.')
        return redirect(url_for('index'))
    if request.method == 'POST':
        name = request.form.get('name')
        description = request.form.get('description')
        if not name:
            flash('Category name is required.')
            return redirect(url_for('new_category'))
        category = Category(name=name, description=description)
        db.session.add(category)
        db.session.commit()
        flash('Category created successfully.')
        return redirect(url_for('admin'))
    return render_template('new_category.html')
# Initialize the database
with app.app_context():
    db.create_all()
    # Create admin user if it doesn't exist
    admin = User.query.filter_by(username='admin').first()
    if not admin:
        admin = User(username='admin', email='admin@example.com', is_admin=True)
        admin.set_password('admin')
        db.session.add(admin)
```

Figură 36 - Ruta de creare categorie

Acest punct final, asemănător cu tabloul de bord administrativ, implică verificarea permisiunilor pentru a limita accesul administratorilor. Validează faptul că este furnizat un nume de categorie și generează noi categorii pe baza informațiilor trimise, permitând administratorilor să organizeze structura conținutului forumului.

Sistemul de gestionare a erorilor

Aplicația implementează instrumente de gestionare a erorilor personalizate pentru codurile de eroare ***HTTP*** comune, oferind utilizatorilor pagini de eroare informative și ușor de utilizat.

HANDLER-ul 404 NOT FOUND

Acest handler gestionează situațiile în care resursa solicitată nu există.

```python
# Error handlers
@app.errorhandler(404)
def page_not_found(e):
    return render_template('errors/404.html'), 404
```

Figură 37 - Handler 404 NOT FOUND

HANDLER-ul 505 Internal Server Error

Acest handler gestionează erorile neașteptate ale serverului, inclusiv eșecurile tranzacțiilor în baza de date.

```python
@app.errorhandler(500)
def internal_server_error(e):
    db.session.rollback()  # Roll back the session in case of database errors
    return render_template('errors/500.html'), 500
```

Figură 38 - Handler 505 Internal Server Error

Implementarea include o retragere a sesiunii de bază de date pentru a se asigura că tranzacțiile eșuate nu lasă baza de date într-o stare inconsistentă, sporind fiabilitatea sistemului.

HANDLER-ul 403 Forbidden

Acest handler gestionează situațiile în care utilizatorii încearcă să acceseze resurse pentru care nu au permisiunea.

```python
@app.errorhandler(403)
def forbidden(e):
    return render_template('errors/403.html'), 403
# Context processors
@app.context_processor
def utility_processor():
    def recent_topics(limit=5):
        return Topic.query.order_by(Topic.created_at.desc()).limit(limit).all()
    def recent_posts(limit=5):
        return Post.query.order_by(Post.created_at.desc()).limit(limit).all()
    def user_count():
        return User.query.count()
    def topic_count():
        return Topic.query.count()
    def post_count():
        return Post.query.count()
    def unread_messages_count():
        if current_user.is_authenticated:
            return Message.query.filter_by(
                recipient_id=current_user.id,
                is_read=False
            ).count()
        return 0
    return {
        'recent_topics': recent_topics,
        'recent_posts': recent_posts,
        'user_count': user_count,
        'topic_count': topic_count,
        'post_count': post_count,
        'unread_messages_count': unread_messages_count
```

Figură 39 - Handler 403 Forbidden

Aplicația Dev Forum oferă o platformă puternică pentru interacțiunea utilizatorilor, crearea de conținut și gestionarea comunității prin aceste puncte finale API cuprinzătoare. Toate acestea se realizează în timp ce se asigură că securitatea, performanța și nivelul de utilizare sunt menținute.

BIBLIOGRAFIE

[1] DAVID FLANAGAN, JavaScript: The Definitive Guide - 7th Edition, O'Reilly Media, Inc., 2020
[2] MATT FRISBIE, Professional JavaScript for Web Developers - 4th Edition, Wrox, 2019
[3] KEITH J. GRANT, CSS in Depth, Manning Publications, 2018;
[4] HENRIK STORMER, Personalized Websites for Mobile Devices using dynamic Cascading Style Sheets, University of Fribourg, 2004;
[5] SYED FAZLE RAHMAN, Jump Start Bootstrap, SitePoint Pty. Ltd, 2014;
[6] SYED FAZLE RAHMAN, Your First Week With Bootstrap, SitePoint Pty. Ltd, 2018;
[7] JAKE SPURLOCK, Bootstrap: Responsive Web Development, O'Reilly Media, Inc., 2013;
[8] JOE CASABONA, HTML and CSS: Visual QuickStart Guide, Peachpit Press, 2020;
[9] JULIE C. MELONI, Sams Teach Yourself HTML, CSS, and JavaScript All in One - Third Edition, Sams, 2019;
[10] DAVID HERRON, NodeJS Web Development - Fifth Edition, Packt Publishing, 2020
[11] BRADLEY MECK, NodeJS in Action, Manning Publications, 2017
[12] DAVID GREEN, Your First Week With NodeJS, SitePoint, 2020
[13] JON WEXLER, Get Programming with NodeJS, Manning Publications, 2019
[14] HAGE YAAPA, Express Web Application Development, Packt Publishing, 2013
[15] ETHAN BROWN, Web Development with Node and Express, O'Reilly Media, Inc., 2019
[16] EVAN M. HAHN, Express in Action: Writing, building, and testing NodeJS applications, Manning Publications, 2016
[17] NICHOLAS MCCLAY, MEAN Cookbook, Packt Publishing, 2017
[18] SHANNON BRADSHAW, MongoDB: The Definitive Guide, O'Reilly Media, Inc., 2019
19 Ronacher, Armin. *Flask Documentation*. https://flask.palletsprojects.com
20 Grinberg, Miguel. *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, 2018.
21 Brown, Anthony. *Python Web Frameworks*. Packt Publishing, 2020.
22 Real Python. *Introduction to Flask*. https://realpython.com/flask-by-example-part-1-project-setup/
23 Bayer, M. (2024). *SQLAlchemy Documentation*. https://docs.sqlalchemy.org
24 Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.
25 Beazley, D. M. (2020). *Python Essential Reference*. Addison-Wesley Professional.
26 Copeland, R. (2013). *SQLAlchemy Essentials*. Packt Publishing.
27 Curylo, J. (2022). *Mastering SQLAlchemy*. Apress.
28 Real Python. *Using SQLAlchemy with Flask*. https://realpython.com
29 Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.
30 Bayer, M. (2024). *SQLAlchemy Documentation*. https://docs.sqlalchemy.org
31 Flask-Login Documentation. (2024). https://flask-login.readthedocs.io
32 Real Python. (2023). *Using Flask-Login for User Authentication in Flask Apps*. https://realpython.com
Copeland, R. (2013). *Essential Flask Extensions*. Packt Publishing.
Grinberg, M. (2018). *Flask Web Development*. O'Reilly Media.
Flask-Caching Documentation. (2024). https://flask-caching.readthedocs.io
Chodorow, K. (2021). *Scaling Python with Redis*. O'Reilly Media.
Real Python. (2023). *Improve Flask Performance with Flask-Caching*. https://realpython.com
Bayer, M. (2024). *SQLAlchemy Core Concepts*. https://docs.sqlalchemy.org
D'Angelo, T. (2022). *High Performance Python Web Applications*. Apress Publishing.
Bayer, M. (2024). *SQLAlchemy Documentation*. https://docs.sqlalchemy.org
Grinberg, M. (2018). *Flask Web Development*. O'Reilly Media
Flask-SQLAlchemy Documentation. (2024). https://flask-sqlalchemy.palletsprojects.com
Real Python. (2023). *Using Flask-SQLAlchemy for Efficient Database Management*. https://realpython.com
Curylo, J. (2022). *Mastering SQLAlchemy in Web Development*. Apress
Copeland, R. (2013). *Essential Flask Extensions*. Packt Publishing
W3C. (2024). *HTML5 Specification*. https://www.w3.org/TR/html5

- Pilgrim, M. (2010). *HTML5: Up and Running*. O'Reilly Media
- Freeman, A. (2021). *Pro HTML5 and CSS3 Design Patterns*. Apress
- Duckett, J. (2011). *HTML & CSS: Design and Build Websites*. Wiley
- Keith, J. (2013). *HTML5 for Web Designers*. A Book Apart
- Mozilla Developer Network. (2024). *HTML Reference*. https://developer.mozilla.org/en-US/docs/Web/HTML
- Bootstrap Team. (2024). *Bootstrap 5 Documentation*. https://getbootstrap.com
- Freeman, A. (2021). *Pro Front-End Development with Bootstrap 5*. Apress
- Keith, J. (2019). *HTML5 and CSS3 for the Real World*. SitePoint
- Coyier, C. (2023). *Modern CSS and Bootstrap Patterns*. CSS-Tricks Publishing
- W3C. (2024). *Web Content Accessibility Guidelines (WCAG) 2.1*. https://www.w3.org/WAI
- Mozilla Developer Network. (2024). *Responsive Web Design Concepts*. https://developer.mozilla.org