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 Absolvent

Dr. Ing. Dragoş Ioan Săcăleanu Anton DOLETE

2023

Universitatea “Politehnica” din Bucureşti

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei Departamentul ELECTRONICA APLICATA SI INGINERIA INFORMATIEI

Anexa 2

**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.

1. Proprietatea intelectuală asupra proiectului aparţine: *studentului*
2. Locul de desfăşurare a activităţii: *UPB*
3. Realizarea practică rămîne în proprietatea: *studentului*
4. Data eliberării temei:

##### CONDUCĂTOR LUCRARE: STUDENT:

Dr. Ing. Dragoș-Ioan Săcăleanu 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 învățămînt 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 5](#_Toc187502049)

[Introducere 6](#_Toc187502050)

[Capitol 1. INTRODUCERE 7](#_Toc187502051)

[1.1 Aplicabilitatea aplicației și motivația lucrării 7](#_Toc187502052)

[Capitol 2. PREZENTAREA TEORETICĂ A METODOLOGIILOR SI A COMPONENTELOR SOFTWARE 8](#_Toc187502053)

[2.1. Dezvoltarea Front-End 8](#_Toc187502054)

[2.1.1. Limbajul de programare JavaScript 8](#_Toc187502055)

[2.1.2 Cascading Style Sheets împreuna cu Bootstrap 10](#_Toc187502056)

[2.1.3. HTML5 13](#_Toc187502057)

[2.1.4. TailWind CSS 13](#_Toc187502058)

[2.2. Modulele utilizate pentru Back-end 15](#_Toc187502059)

[2.2.1. NodeJS 15](#_Toc187502060)

[2.2.2. ExpressJS 18](#_Toc187502061)

[2.2.3. MongoDB 21](#_Toc187502062)

[Capitol 3. Componente pentru dezvoltare 24](#_Toc187502063)

[3.1 Controlul Versiunilor și Git 24](#_Toc187502064)

[3.2 Mutarea bazei Mongo in Cloud 25](#_Toc187502065)

[Capitol 4. Descrierea aplicatiei 27](#_Toc187502066)

[4.1. Descriere a utilității practice 27](#_Toc187502067)

[4.2. Proiectarea si Implementarea(MongoDB) 28](#_Toc187502068)

[4.3. Arhitectura aplicației 31](#_Toc187502069)

[4.3.1. Controller 31](#_Toc187502070)

[4.3.2. Rutare 31](#_Toc187502071)

[4.3.3. Securizarea Parolei 47](#_Toc187502072)

[4.3.4. EJS Vizualizările(Componentele Template) 49](#_Toc187502073)

[4.4. Utilizarea aplicației 50](#_Toc187502074)

[4.4.1. Utilizare (prezentare) 50](#_Toc187502075)

[4.4.2. Utilizare (Interfata) 52](#_Toc187502076)

[Capitol 5. Concluzii 55](#_Toc187502077)

[BIBLIOGRAFIE 56](#_Toc187502078)

# Lista figuri

[Figură 1-Standardul ECMAScript / (DOM) / (BOM) 9](#_Toc187499487)

[Figură 2- Boostrap Layer 12](#_Toc187499488)

[Figură 3 - Tailwind Instalare Direct în HTML 14](#_Toc187499489)

[Figură 4- Tailwind Layout Responsiv 14](#_Toc187499490)

[Figură 5- Fire de execuție Asincrone 16](#_Toc187499491)

[Figură 6- Thread-uri NodeJS 17](#_Toc187499492)

[Figură 7- Instructiune non-asincrona 17](#_Toc187499493)

[Figură 8- Interogarea bazei de date 18](#_Toc187499494)

[Figură 9- Șablon Server NodeJS 18](#_Toc187499495)

[Figură 10- Flux de solicitări HTTP NodeJS 20](#_Toc187499496)

[Figură 11- Șablon Sever ExpressJS 21](#_Toc187499497)

[Figură 12 - Colecția User-ului in baza de date 28](#_Toc187499498)

[Figură 13 - Proiectarea arhitecturii 28](#_Toc187499499)

[Figură 14 - Schema Mongoose a Topicului de discuție 29](#_Toc187499500)

[Figură 15 - Schema Mongoose a User-ului 30](#_Toc187499501)

[Figură 16 - Organizarea Controllerelor 31](#_Toc187499502)

[Figură 17 - URI 32](#_Toc187499503)

[Figură 18 - Inregistrarea unui nou utilizator 33](#_Toc187499504)

[Figură 19 - Rută POST /register 34](#_Toc187499505)

[Figură 20 - Răspuns /register 34](#_Toc187499506)

[Figură 21 - Rută POST /login 35](#_Toc187499507)

[Figură 22 - Răspuns /login 36](#_Toc187499508)

[Figură 23 - Logarea unui utilizator 36](#_Toc187499509)

[Figură 24 - Token Autorizare Creare postare 37](#_Toc187499510)

[Figură 25 - Rută Creare Postare 37](#_Toc187499511)

[Figură 26 - Răspuns Creare Postare 38](#_Toc187499512)

[Figură 27 - Crearea unei postari 38](#_Toc187499513)

[Figură 28 - Rută Creare Comentariu 40](#_Toc187499514)

[Figură 29 - Răspuns Creare Comentariu 40](#_Toc187499515)

[Figură 30 - Crearea unui comentariu 41](#_Toc187499516)

[Figură 31 - Editarea unei Postări de către autor 42](#_Toc187499517)

[Figură 32 - Rută editarea unei Postări 43](#_Toc187499518)

[Figură 33 - Răspuns Editare Postare 43](#_Toc187499519)

[Figură 34 - Ștergerea unei Postări de către autor 45](#_Toc187499520)

[Figură 35 - Rută de Ștergere a unei Postări 45](#_Toc187499521)

[Figură 36 - Răspuns ștergere postare 46](#_Toc187499522)

[Figură 37 - Rută de obținere a postărilor 47](#_Toc187499523)

[Figură 38 - Configurarea Passport API 49](#_Toc187499524)

[Figură 39 - Exemplu EJS view 50](#_Toc187499525)

[Figură 40 - Diagrama pentru vizitatori 51](#_Toc187499526)

[Figură 41 - Diagrama pentru utilizatori înregistrați 52](#_Toc187499527)

[Figură 42 - Template Pagină de pornire 54](#_Toc187499528)

# 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 „Dev Forum” 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

### Aplicabilitatea aplicației și motivația lucrării

#### 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.

#### Motivația E-community

E-community 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.

#### 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.

### Dezvoltarea Front-End

### Limbajul de programare JavaScript

Limbajul principal de programare a site-urilor web este JavaScript. Majoritatea paginilor web utilizează JavaScript și fiecare browser web actual — desktop, tabletă, telefon — include un interpretor JavaScript.

JavaScript este popular in rândul industriei software datorită lui NodeJS, care a făcut posibilă scrierea codului JavaScript în afara browserelor.[1]

*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]

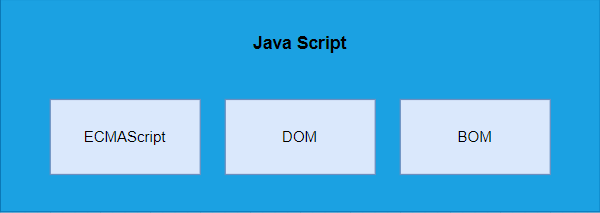
JavaScript 1.1 a fost adus la cunoștința Asociației Europene a Producătorilor de Calculatoare

(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 2.1[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] .

### Cascading Style Sheets împreuna cu Bootstrap

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.](http://WWW/)

*Scurta istorie a CSS*

Când Hkon Wium Lie a început să lucreze la CERN, locul de naștere al rețelei, în `94, abia atunci începe utilizarea ca spațiu de publicare digitală. Acesta este momentul în care începe povestea CSS. Cu toate acestea, lipsea o componentă fundamentală a unei spațiu digital de publicare, nu era o metodă pentru stilare lucrărilor. De exemplu, nu a fost posibilă formatarea unei pagini Web cu un stil de ziar. Hkon a văzut cererea pentru un limbaj de design ca si o necesitate cruciala.

Propunerea creării unui limbaj de design , a fost făcută la una din conferințele web ale acelei vremi, la Chicago in anul `94, iar prezentarea a făcut furori sunt multe dezbateri pe aceasta tema. Mulți au susținut că CSS este ușor pentru task-urile care a fost creat, cu argumente ca si cum necesitatea cunoștințelor de programare cuprinzătoare pentru a stila corect documentele. CSS are o săgeată precisă îndreptată în

direcția corectă, derulând o sintaxă clară și declarativă.

Este greu de spus cât de larg este folosit CSS acum, de la debutul său oficial din `96, deși procentul de pagini care nu se folosesc CSS probabil nu poate fi mai mare de un număr mic de puncte procentuale.

*CascadingStyleSheets*

Prin utilizarea CascadingStyleSheets, posibil să se separe estetica unei aplicații web, cum ar fi aspectul paginii, culorile și font-urile, de textul acesteia. Această împărțire poate crește accesibilitatea conținutului, oferind, de asemenea, specificatorului elementelor de prezentare mai multă libertate și control. În plus, elimină complexitatea și duplicarea codului, permițând furnizarea codului CascadingStyleSheets, pertinent in fișiere diferite. Reținerea în cache a fișierelor pe care l-au partajat, timpul de încărcare a paginii este, de asemenea, accelerat.

Putem susține că CascadingStyleSheets, se deosebește in domeniul dezvoltării software pe o serie de aspecte. Deși nu este din punct de vedere tehnic un limbaj de programare, necesită gândire abstractă. Necesită puțină ingeniozitate și este mai mult decât un instrument de proiectare.

Cuvântul „în cascadă” se referă la structura de prioritate declarată care determină ce regulă de stil este implementată atunci când multiple seturi de instrucțiuni se refera la un element. Acest set de priorități poate fi prezis, dar poate cauza probleme pentru proiecte de o complexitate considerabilă, ajungând în unele situații (când front-end se dezvolta de mai mulți dezvoltatori), mai ales dacă regulile sunt omise si încălcate, când se introduc seturi de reguli greșite în cod.

Atunci când este necesar să efectuăm o acțiune folosind tehnici standard de programare, putem identifica rapid problema care trebuie rezolvată (de exemplu, „Cum descopăr componente de tip x într-un tablou?”). Tehnica ideală pentru a realiza orice este adesea determinată de limitările specifice ale aplicației noastre și de modul exact în care dorim să abordăm anumite situații. Deși sunt necesare „tricks” practice pe care le putem folosi, înțelegerea CascadingStyleSheets necesită o definire aprofundată a regulilor care permit astfel de actiuni. [3] CascadingStyleSheets adaugă un vocabular declarativ ușor de greșit, de îndată ce îl folosim pentru aplicații care sunt cel puțin moderat complicate, devine din ce în ce mai dificil să implementăm regulile CSS într-o manieră ordonată.

Începând cu anii 2000, odată cu creșterea numărului de dispozitive mobile, a crescut si cerea pentru dispozitive mult mai speciale si specifice. La acea vreme, WAP sau I-mode, se puteau

folosii pentru a converti documentele web pentru dispozitive mobile, dar acest lucru a necesitat un efort considerabil, deoarece aproape toate site-urile trebuiau reconstruite complet.

În prezent avem sute de resurse CSS din care să alegem datorită muncii depuse de diverse organizații în efortul de a remedia limitările prezentate , pentru ca in prezent site-urile web sa aibă o structura ușor inteligibila, un design plăcut și optimizat pentru toate dispozitivele.

*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]

*Framework-ul Bootstrap*

În timp ce lucrau la Twitter, Mark Otto și Jacob Thornto au creat aplicația open source cunoscută sub numele de Bootstrap. Nevoia de Bootstrap, subliniază Mark Otto în post-ul pe blog care s-a anunțat introducerea sa, a izvorât din necesitatea de a standardiza numeroasele tehnologii front-end utilizate

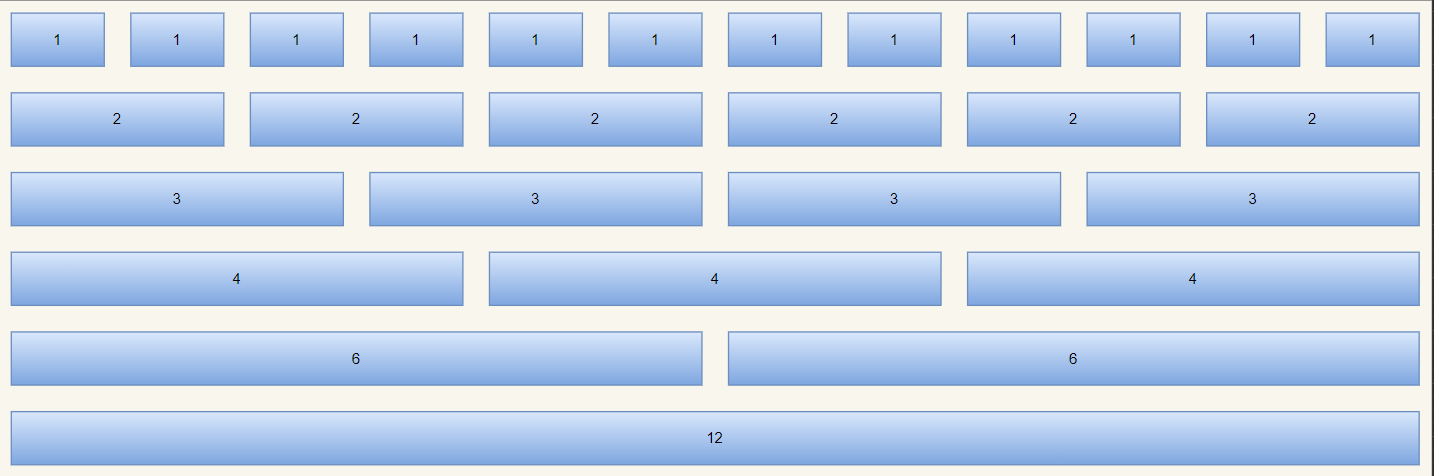
de dezvoltatorii din întreaga organizație.

În 2011 a fost lansat Bootstrap v1.0, care a inclus doar elemente CSS și HTML. Până in versiunea v1.3 a Bootstrap, aceasta fiind compatibila cu InternetExplorer7 și InternetExplorer8, nu erau prezente plugin-uri JavaScript. O altă schimbare semnificativă a venit în 2012 cu Bootstrap v2.0. Biblioteca Bootstrap a fost rescrisă în întregime, iar ulterior a evoluat într-un cadru receptiv. Telefoanele mobile, tabletele și desktop-urile puteau folosi toate componentele sale și au fost incluse și numeroase pachete CSS și JavaScript noi. Bootstrap 3, o altă lansare mare, devenind un cadru „Mobile First and Always Responsive” după 15 modificări substanțiale. Crearea unui site web receptiv a fost opțională în iterațiile anterioare ale cadrului. Structura de directoare a proiectului, precum și numele claselor au fost modificate în versiunea 2013. 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 2) 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 2 ilustrează modul în care piesele pot fi aranjate pe un layer de 12. [7]



Figură 2- 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.

### HTML5

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

*WWW si HTML*

Accesul la informații prin Internet a fost o provocare tehnică semnificativă înainte de `90. De fapt, in acest tot timp, pana si cele mai clipitoare minți si utilizatori din diferite domenii de activitate din cadrul academic aveau dificultăți, în timp ce încercau să partajeze date. Tim Berners- Lee, a conceput o metodă de accesare rapidă a textului prin Internet, și anume prin utilizarea legăturilor hipertext. Deși acesta nu a fost un concept nou, ușurința de utilizare a HTML i-a permis să supraviețuiască atunci când eforturile de hypertext mai sofisticate 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.

### TailWind CSS

*Istoric*

Tailwind CSS a fost lansat în 2017 de Adam Wathan și echipa sa. Tailwind CSS este un framework CSS bazat pe utilități care permite dezvoltatorilor să creeze interfețe web personalizate rapid și eficient.

Spre deosebire de framework-uri precum Bootstrap, care oferă componente predefinite, Tailwind CSS oferă o abordare atomică a designului, concentrându-se pe clase utilitare reutilizabile pentru stilizarea elementelor.

**Avantajele utilizării Tailwind CSS:**

* **Flexibilitate ridicată:** Fiecare clasă CSS este concepută pentru a realiza un singur scop specific (ex. text-center, bg-blue-500, p-4), oferind control complet asupra designului.
* **Personalizare avansată:** Tailwind CSS permite utilizatorilor să modifice fișierul de configurare (tailwind.config.js) pentru a adapta framework-ul la cerințele aplicației.
* **Performanță optimizată:** Utilizarea PostCSS și a plugin-ului PurgeCSS reduce dimensiunea fișierului final CSS, eliminând clasele neutilizate.

*Instalare Direct în HTML (CDN)*

**O imagine care conține text, captură de ecran, Font

Descriere generată automat**

Figură 3 - Tailwind Instalare Direct în HTML

*Layout Responsiv*

Tailwind CSS facilitează crearea layout-urilor responsive prin utilizarea claselor bazate pe breakpoint-uri:

**O imagine care conține text, Font, captură de ecran

Descriere generată automat**

Figură 4- Tailwind Layout Responsiv

**Integrarea în aplicație:** Tailwind CSS a fost folosit pentru a dezvolta componentele UI ale aplicației E-community, contribuind la crearea unui design coerent și responsiv. De exemplu, butoanele pentru navigație și secțiunile de afișare a discuțiilor au fost stilizate folosind clase precum hover:bg-gray-700 și rounded-lg pentru un aspect modern.

### Modulele BACKEND

### 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*

Aplicațiile web sunt esențiale în infrastructura tehnologică a societății în era digitală contemporană. Pentru a obține produse eficiente, scalabile și ușor de întreținut, este esențial să alegeți un cadru adecvat pentru dezvoltarea acestora.

Datorită comunității sale extinse de dezvoltatori și sintaxei sale clare, Python este adesea limbajul de programare interpretat și multiparadigmă preferat pentru crearea aplicațiilor web. Această lucrare va examina framework-ul Flask, care este unul dintre cele mai populare microframework-uri Python, din punct de vedere al arhitecturii, beneficiilor, dezavantajelor și aplicabilității sale în proiecte reale.

*Prezentarea generală a framework-ului Flask*

Flask a fost creat de Armin Ronacher în 2010, în timp ce lucra la proiectul Pocoo. A fost gândit ca un cadru simplu, astfel încât dezvoltatorii să aibă cea mai mare libertate în alegerea și integrarea componentelor necesare. Folosind toolkit-ul Werkzeug și motorul de template-uri Jinja2, ambele dezvoltate de echipa Pocoo, Flask a fost construit.

Filosofia „micro” în Flask: Un „micro-cadru” nu are restricții de funcționalitate, ci are componente predefinite precum ORM-ul sau sistemul de autentificare. Această metodă oferă un grad ridicat de flexibilitate, permițând dezvoltatorului să adapteze aplicația la cerințele specifice ale proiectului.

*Arhitectura Flask și fluxul de lucru*

Două biblioteci principale servesc ca suport pentru componentele sale fundamentale:  
Werkzeug – un set de instrumente WSGI care facilitează comunicarea între serverul web și aplicația Python. Jinja2 este un motor de template-uri care permite ca logica aplicației să fie separată de partea 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*

Avantaje sunt următoarele: flexibilitatea ridicată – dezvoltatorul are control total asupra modului în care este structurată aplicația. Curba de învățare redusă este ideală atât pentru prototipuri rapide, cât și pentru începători. Există multe extensii oficiale și comunitare, cum ar fi Flask-Login, Flask-WTF și Flask-SQLAlchemy.

Iar limitările sunt următoarele: nu există funcționalități implicite; dezvoltatorii trebuie să integreze componente manual. Managementul aplicațiilor mari: proiectele complexe pot fi dificile de întreținut dacă nu au o structură strictă.

*Cazuri de utilizare și exemple practice*

Flask este utilizat frecvent în proiecte educaționale, startup-uri, aplicații RESTful și prototipuri MVP (Minimum Viable Product). De exemplu, Pinterest și LinkedIn au folosit Flask pentru anumite componente ale infrastructurii lor. Un exemplu simplu de aplicație cu Flask constă în realizarea unui blog sau a unei interfețe CRUD conectate la o bază de date SQLite, folosind Flask-SQLAlchemy.

*Concluzii*

Framework-ul Flask reprezintă o soluție eficientă și elegantă pentru dezvoltarea aplicațiilor web cu Python, adresându-se în special celor care apreciază simplitatea, controlul asupra arhitecturii și extensibilitatea. Deși nu este la fel de cuprinzător ca alte framework-uri de tip „full-stack”, Flask își păstrează relevanța prin abordarea sa minimalistă, ceea ce îl face ideal pentru învățare, prototipuri și proiecte flexibile. În concluzie, Flask nu este doar un punct de plecare excelent pentru începători, ci și un instrument valoros în mâinile dezvoltatorilor experimentați.

*Exemplu*



### SQLAlchemy ORM – Abordare Modernă și Scalabilă pentru 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*

Object-Relational Mapping (ORM) este o tehnologie care facilitează conversia datelor între sistemele incompatibile de tip obiectual și cele relaționale. Mai exact, ORM-ul permite manipularea datelor dintr-o bază relațională prin intermediul obiectelor dintr-un limbaj de programare orientat pe obiecte, eliminând necesitatea scrierii directe a interogărilor SQL.

*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*

SQLAlchemy a fost creat de Michael Bayer și lansat în 2005, cu scopul de a oferi un framework extensibil, eficient și non-invaziv pentru lucrul cu baze de date în Python. Spre deosebire de alte ORM-uri care impun o arhitectură rigidă, SQLAlchemy promovează libertatea de alegere între lucrul direct cu SQL („Core”) și utilizarea ORM-ului de nivel înalt.

SQLAlchemy este împărțit în două componente principale: SQLAlchemy Core fiind un set de instrumente pentru generarea dinamică a SQL-ului, gestionarea conexiunilor și a tranzacțiilor. SQLAlchemy ORM un strat obiectual peste Core, care mapează clase Python la tabele SQL. Această separare permite integrarea progresivă a ORM-ului, oferind dezvoltatorilor un control granular asupra fiecărui nivel al interacțiunii cu baza de date.

*Maparea claselor și relațiilor*

În SQLAlchemy, clasele Python sunt mapate la tabele SQL prin declararea atributelor ca instanțe ale obiectelor Column. Tipurile de date sunt specificate prin tipuri explicite (Integer, String, Boolean, etc.), iar relațiile între tabele se definesc cu ajutorul funcțiilor relationship() și foreign\_key().

SQLAlchemy oferă mecanisme sofisticate de optimizare a performanței prin controlul modului în care sunt încărcate datele relaționate: **lazy loading** in care datele sunt încărcate doar la momentul accesului. Iar **eager loading** in care interogările sunt generate pentru a include datele relaționate într-o singură execuție.

*Managementul sesiunilor*

Session este un obiect-cheie în SQLAlchemy ORM, care menține o sesiune tranzacțională între aplicație și baza de date. Prin Session, se realizează adăugarea, modificarea, commit-ul și rollback-ul operațiilor, asigurând consistența datelor.

Modelul „Unit of Work” implementat de SQLAlchemy asigură agregarea tuturor modificărilor asupra obiectelor într-o singură tranzacție atomică, ceea ce permite rollback în caz de eroare și păstrează integritatea datelor.

*SQLAlchemy vs Django ORM*

Deși Django ORM este mai simplu de utilizat pentru aplicații standard, SQLAlchemy oferă un grad de flexibilitate și control net superior, fiind alegerea preferată în proiecte complexe sau enterprise. SQLAlchemy permite generarea dinamică a interogărilor, suport pentru baze de date multiple și mapări avansate (ex. joined table inheritance). Alte ORM-uri populare: Peewee fiind mai simplu, dar limitat în funcționalități avansate. Pony ORM oferă un DSL propriu (Domain Specific Language), dar cu mai puțină documentație și suport comunitar comparativ cu SQLAlchemy.

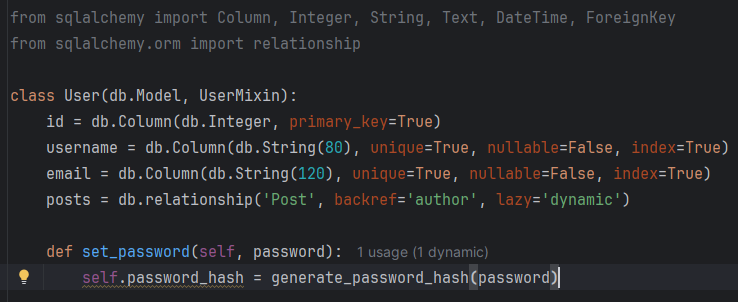
*Proiect real: API RESTful cu Flask și SQLAlchemy*

SQLAlchemy este adesea utilizat în tandem cu Flask pentru a crea API-uri REST. Într-un proiect tipic, modelele SQLAlchemy definesc structura bazei de date, în timp ce Flask gestionează rutele și logica de business. Persistența se realizează prin intermediul sesiunilor și se pot aplica validări cu ajutorul Flask-WTF sau Marshmallow. SQLAlchemy este utilizat în numeroase proiecte de amploare, inclusiv aplicații interne pentru companii tech, sisteme CRM sau ERP, platforme de analiză de date și aplicații cu scalabilitate distribuită.

*Concluzii*

SQLAlchemy ORM reprezintă un exemplu elocvent de integrare între programarea orientată pe obiecte și modelul relațional de baze de date. Prin arhitectura sa flexibilă, SQLAlchemy oferă o soluție scalabilă și puternică pentru dezvoltatorii care au nevoie de control asupra interacțiunilor cu baza de date, fără a renunța la productivitate sau abstractizare. Utilizarea eficientă a ORM-ului presupune o bună înțelegere a fundamentelor SQL, dar și a principiilor OOP, ceea ce transformă SQLAlchemy într-un instrument formativ esențial în pregătirea profesională a oricărui inginer software.

*Exemplu*



### 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*

În contextul dezvoltării aplicațiilor web, autentificarea utilizatorilor reprezintă o componentă esențială pentru asigurarea securității și personalizării serviciilor. Procesul de autentificare nu se reduce doar la verificarea identității, ci implică întreaga gestiune a sesiunilor, protecția datelor și controlul accesului la resursele aplicației. În ecosistemul Python, framework-ul Flask se remarcă prin flexibilitate și extensibilitate, permițând integrarea unor soluții modulare pentru diverse funcționalități. Una dintre cele mai utilizate extensii în acest sens este Flask-Login, o bibliotecă dezvoltată pentru a facilita implementarea autentificării și a gestionării sesiunilor într-un mod sigur, coerent și eficient.

*Prezentarea extensiei Flask-Login și integrarea în Flask*

Flask-Login este o extensie destinată aplicațiilor construite cu Flask, care furnizează un set de instrumente pentru gestionarea autentificării utilizatorilor și menținerea sesiunilor active. Această bibliotecă este construită pentru a fi ușor de integrat, păstrând în același timp flexibilitatea necesară pentru proiecte de diferite dimensiuni. Prin intermediul său, dezvoltatorul poate gestiona login-ul, logout-ul, protecția paginilor, încărcarea utilizatorilor dintr-o sursă de date și verificarea autentificării în mod automat. Arhitectura extensiei se bazează pe conceptul de user loader, o funcție care preia obiectul asociat unui anumit utilizator autentificat, și pe un sistem de sesiuni care utilizează cookies pentru a păstra 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*

Una dintre funcționalitățile centrale oferite de Flask-Login este persistența sesiunii utilizatorului autentificat prin intermediul mecanismului de cookies. Atunci când un utilizator se conectează cu succes, extensia generează un identificator unic care este stocat în browser-ul clientului și verificat la fiecare cerere ulterioară. Acest mecanism permite menținerea autentificării pe durata vizitei și, opțional, persistarea stării chiar și după închiderea browserului prin activarea opțiunii „remember me”.

Pentru a asigura protecția datelor de autentificare, Flask-Login nu gestionează parolele direct, ci lucrează în tandem cu alte biblioteci precum Werkzeug sau Flask-Bcrypt, care oferă funcționalități pentru criptarea și verificarea parolelor. Securitatea aplicației este consolidată și prin utilizarea metodelor de protecție împotriva atacurilor de tip Cross-Site Request Forgery (CSRF), care se pot integra facil cu alte extensii precum Flask-WTF. În plus, Flask-Login permite și personalizarea comportamentului în cazul în care un utilizator neautentificat încearcă să acceseze o resursă protejată, oferind suport pentru redirecționarea către o pagină de login sau pentru returnarea unui mesaj de eroare explicit.

*Considerații arhitecturale și bune practici*

Implementarea unui sistem de autentificare robust trebuie să țină cont nu doar de funcționalitatea de bază, ci și de modul în care aceasta se integrează în ansamblul aplicației. În acest sens, Flask-Login oferă dezvoltatorului un echilibru între simplitate și extensibilitate, permițând separarea clară a logicii de autentificare de restul componentelor aplicației. O bună practică constă în crearea unui blueprint dedicat pentru toate rutele de autentificare și în păstrarea clasei User într-un fișier distinct, ceea ce facilitează organizarea codului și reutilizarea componentelor în aplicații mai mari.

Este esențial ca autentificarea să fie dublată de un sistem solid de autorizare, iar Flask-Login, deși nu oferă un mecanism nativ pentru gestionarea permisiunilor, poate fi extins prin utilizarea altor biblioteci sau prin implementarea manuală a logicii de acces pe baza unor roluri sau privilegii definite în modelul utilizatorului. O atenție deosebită trebuie acordată și validării datelor transmise prin formulare, evitării păstrării parolelor în format clar text și protejării fișierelor sensibile de configurare, precum secret key-ul aplicației.

*Aplicații și studii de caz*

Flask-Login este utilizat pe scară largă în proiecte comerciale, aplicații educaționale, API-uri RESTful și prototipuri de produs. Într-un scenariu obișnuit, aplicația include o bază de date relațională în care sunt stocate informațiile despre utilizatori, împreună cu un sistem de înregistrare, login și logout. Interfața de autentificare este adesea implementată cu ajutorul extensiei Flask-WTF, care permite generarea și validarea formularelor într-un mod sigur și estetic. În cazul unei aplicații multi-utilizator, este posibilă extinderea funcționalităților Flask-Login prin definirea de sesiuni paralele, verificări ale tipului de utilizator și integrarea cu sisteme de autentificare externe precum OAuth 2.0 sau LDAP.

Un caz concret de utilizare ar putea fi un sistem intern de management al proiectelor în cadrul unei organizații, în care fiecare utilizator are acces la propriile resurse, iar administratorii dispun de permisiuni extinse. În astfel de cazuri, Flask-Login se dovedește a fi o soluție fiabilă și scalabilă pentru gestionarea autentificării fără a compromite ușurința de utilizare sau performanța aplicației.

*Concluzii*

Flask-Login se afirmă ca o extensie esențială în arsenalul dezvoltatorilor care construiesc aplicații web sigure și bine structurate în Python. Prin abordarea sa minimalistă, dar eficientă, extensia facilitează implementarea proceselor de autentificare și menținere a sesiunilor, fără a impune constrângeri arhitecturale rigide. Flexibilitatea sa permite integrarea cu diverse alte componente din ecosistemul Flask, iar documentația amplă și comunitatea activă contribuie la adoptarea sa pe scară largă. În concluzie, Flask-Login este mai mult decât un instrument de login – este un element fundamental pentru construirea unor aplicații moderne, conforme cu cerințele actuale de securitate și funcționalitate.

*Exemplu*

O imagine care conține text, captură de ecran, software, Font

Conținutul generat de inteligența artificială poate fi incorect.

### 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 eră în care timpul de răspuns al unei aplicații web poate influența direct rata de retenție a utilizatorilor și succesul comercial al produsului digital, optimizarea performanței devine un obiectiv central în arhitectura sistemelor moderne. Una dintre cele mai eficiente metode de a reduce timpul de procesare și de a diminua încărcarea serverului constă în implementarea unor mecanisme de caching, care permit stocarea temporară a rezultatelor unor operațiuni costisitoare pentru a fi reutilizate ulterior. În cadrul aplicațiilor dezvoltate cu Flask, un micro-framework scris în Python, extensia Flask-Caching oferă o soluție versatilă și extensibilă pentru integrarea strategiilor de cache. Această lucrare își propune să investigheze în profunzime modul în care Flask-Caching contribuie la eficientizarea aplicațiilor web, să explice conceptele fundamentale pe care se bazează și să analizeze diferite strategii și modele de implementare.

*Conceptul de caching și importanța sa în arhitectura web*

Caching-ul, în sensul său fundamental, reprezintă procesul de stocare temporară a unor date care sunt dificil sau costisitor de obținut în mod repetat, în scopul reducerii timpului de răspuns și al creșterii eficienței aplicației. În mediul web, acest mecanism se poate aplica la nivel de client, server sau rețea și poate implica stocarea de răspunsuri HTTP, pagini HTML generate dinamic, rezultate ale interogărilor către baza de date sau conținutul unor API-uri externe.

Prin eliminarea necesității de a regenera aceeași resursă la fiecare cerere, caching-ul reduce nu doar timpul de procesare, ci și volumul de muncă al serverului, scade consumul de bandă și contribuie la scalabilitatea întregului sistem. Cu toate acestea, caching-ul introduce complexitate în gestiunea datelor, deoarece conținutul poate deveni rapid învechit, iar sincronizarea între datele cache și sursa lor de adevăr devine un aspect critic.

*Introducere în Flask-Caching și principiile sale de funcționare*

Flask-Caching este o extensie dedicată aplicațiilor construite cu framework-ul Flask, care oferă un sistem unificat și extensibil pentru implementarea politicilor de cache. Dezvoltată pentru a fi ușor de utilizat, dar suficient de flexibilă pentru aplicații complexe, Flask-Caching funcționează prin decoratori care pot fi atașați la funcții costisitoare, permițând stocarea rezultatelor acestora într-un backend de cache configurabil. Extensia este proiectată să fie agnostică în privința backend-ului, oferind suport pentru o gamă variată de sisteme, cum ar fi cache-ul în memorie (SimpleCache), fișiere (FileSystemCache), Redis, Memcached, sau chiar baze de date relaționale.

În procesul de caching, fiecare funcție decorată este evaluată o singură dată pentru un set dat de argumente, iar rezultatul ei este stocat împreună cu o cheie unică generată pe baza acestor argumente. La cererile ulterioare cu aceleași intrări, rezultatul este extras din cache fără a mai executa funcția originală, reducând semnificativ timpul de execuție. Flask-Caching permite controlul fin asupra duratei de viață a valorilor cache prin parametri precum timeout, oferind astfel un echilibru între performanță și actualitatea datelor.

*Modele de caching și strategii de invalidare*

Implementarea unui sistem de caching eficient presupune alegerea unei strategii adaptate specificului aplicației. Una dintre cele mai comune forme este caching-ul la nivel de funcție, în care rezultatul unei funcții de procesare intensă – precum generarea unei pagini de dashboard sau calculul unor agregări statistice – este memorat pentru o durată prestabilită. În mod similar, caching-ul poate fi aplicat la nivelul unor răspunsuri HTTP întregi sau la nivelul componentelor HTML, atunci când se lucrează cu motoare de template-uri precum Jinja2. Pe lângă aceste modele, caching-ul per utilizator sau pe bază de permisiuni poate fi folosit pentru personalizarea eficientă a interfețelor fără a repeta procese intensive de calcul.

Invalidarea cache-ului, adică ștergerea sau reîmprospătarea datelor stocate, este un aspect esențial și adesea problematic în arhitectura sistemelor care se bazează pe caching. Flask-Caching oferă metode explicite pentru curățarea cache-ului fie global, fie pe baza unor chei specifice, precum și opțiuni pentru a seta un timp de expirare automat. Alegerea unei strategii de invalidare depinde în mod direct de natura și volumul datelor, precum și de nivelul de consistență dorit. De exemplu, într-o aplicație de e-commerce, modificarea prețului unui produs ar trebui să invalideze imediat toate cache-urile asociate acelui produs, în timp ce într-un blog, reîmprospătarea articolelor se poate face periodic, fără un impact critic asupra coerenței informațiilor.

*Performanță și scalabilitate în aplicațiile Flask cu caching*

Prin reducerea numărului de operațiuni costisitoare, caching-ul contribuie direct la scăderea timpilor de răspuns, la diminuarea încărcării asupra procesorului și a bazei de date, precum și la creșterea capacității aplicației de a deservi un număr mai mare de utilizatori simultan. Testele comparative demonstrează că, în aplicații cu un trafic ridicat și conținut dinamic dar repetitiv, utilizarea corectă a Flask-Caching poate reduce timpul de răspuns cu peste 50% și poate scădea semnificativ rata de erori provocate de suprasarcină.

În implementările distribuite, caching-ul trebuie proiectat ținând cont de caracterul paralel al procesării și de posibila fragmentare a memoriei cache între noduri. Backend-uri precum Redis sau Memcached permit partajarea valorilor cache între multiple instanțe ale aplicației, asigurând astfel o coerență relativă și performanță constantă. Flask-Caching permite integrarea acestor sisteme externe cu ușurință, oferind parametri de configurare care se aliniază bunelor practici în infrastructura de producție.

*Considerații de securitate și limitări*

Deși caching-ul aduce multiple beneficii din punct de vedere al performanței, utilizarea necontrolată a acestuia poate introduce riscuri de securitate și confidențialitate. Stocarea temporară a datelor sensibile în cache-uri partajate fără izolare adecvată poate conduce la expunerea informațiilor către utilizatori neautorizați. De aceea, în proiectarea politicilor de caching trebuie avute în vedere nu doar considerentele de performanță, ci și cele legate de confidențialitate și integritate. Caching-ul per utilizator ar trebui să fie implementat cu identificatori unici, iar datele rezultate ar trebui să fie criptate sau restricționate atunci când este cazul.

Totodată, Flask-Caching are limitările sale, în special în ceea ce privește managementul avansat al dependențelor dintre date sau invalidarea automată a cache-ului în funcție de modificările surselor externe. În aplicațiile complexe, este adesea necesară completarea Flask-Caching cu instrumente suplimentare sau dezvoltarea unor mecanisme proprii de urmărire a coerenței datelor cache.

*Concluzii*

Extensia Flask-Caching oferă o soluție robustă și eficientă pentru optimizarea aplicațiilor web dezvoltate în Flask, contribuind semnificativ la îmbunătățirea performanței și a scalabilității acestora. Printr-o arhitectură simplă dar flexibilă, extensia permite dezvoltatorilor să integreze rapid politici de caching, adaptate diferitelor tipuri de date și niveluri de acces. În același timp, implementarea corectă a caching-ului implică o înțelegere profundă a aplicației, a fluxurilor de date și a relațiilor dintre componente, pentru a asigura atât eficiența, cât și consistența informațională. Cu o documentație clară și o comunitate activă, Flask-Caching se impune ca o extensie esențială în arsenalul dezvoltatorului Python modern, contribuind la realizarea unor aplicații performante, fiabile și pregătite pentru scalare.

*Exemplu*



### 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*

În arhitectura modernă a aplicațiilor web, manipularea datelor persistente constituie una dintre componentele esențiale ale oricărui sistem funcțional. Indiferent de dimensiunea aplicației, fie că este vorba despre un prototip academic sau despre un produs enterprise, gestionarea eficientă a bazelor de date devine un obiectiv tehnologic central. În contextul dezvoltării cu Python, limbaj recunoscut pentru expresivitatea sa sintactică și ecosistemul bogat în biblioteci, framework-ul Flask se remarcă prin minimalism și extensibilitate, oferind dezvoltatorilor un grad ridicat de control asupra arhitecturii aplicației. Totuși, Flask, în forma sa de bază, nu oferă funcționalități avansate pentru interacțiunea cu baze de date. Din acest motiv, extensia Flask-SQLAlchemy a devenit un instrument indispensabil pentru dezvoltatorii care doresc să integreze un sistem robust de mapare obiect-relațională (ORM) 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-ul reprezintă o punte între aceste două paradigme, permițând dezvoltatorului să lucreze cu baze de date folosind concepte familiare programării orientate pe obiecte, fără a scrie direct interogări SQL. SQLAlchemy, biblioteca de bază pe care se sprijină Flask-SQLAlchemy, este una dintre cele mai puternice și flexibile implementări ORM din ecosistemul Python, oferind atât un nivel înalt de abstractizare, cât și posibilitatea de a lucra direct cu expresii SQL, atunci când este necesar. SQLAlchemy permite definirea modelelor de date ca clase Python, mapate automat la structura bazei de date, și oferă suport pentru relații complexe, moștenire, validări, tranzacții și integrare multiplă cu diferite tipuri de baze de date, printre care PostgreSQL, MySQL, SQLite sau Oracle.

*Integrarea SQLAlchemy în Flask prin Flask-SQLAlchemy*

Flask, ca micro-framework, nu include în mod nativ suport pentru gestionarea bazelor de date, însă datorită arhitecturii sale modulare permite adăugarea acestui tip de funcționalitate prin extensii externe. Flask-SQLAlchemy este una dintre cele mai mature și utilizate extensii ale ecosistemului Flask și oferă o interfață coerentă pentru utilizarea SQLAlchemy în interiorul aplicației.

Aceasta simplifică 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. Prin înregistrarea instanței SQLAlchemy în contextul aplicației Flask, dezvoltatorul are acces la o interfață unificată pentru toate operațiunile legate de persistarea datelor, fără a fi nevoit să gestioneze manual conexiunile, tranzacțiile sau configurarea dialectelor SQL.

Definirea modelelor de date în Flask-SQLAlchemy se face prin extinderea clasei de bază oferite de ORM, fiecare model fiind reprezentat de o clasă Python care definește atributele tabelei corespondente. Atributele clasei devin coloane ale tabelei, iar relațiile între tabele sunt reprezentate prin referințe directe între modele. Accesul la date se realizează prin instanțe ale acestor clase, în timp ce operațiile CRUD (create, read, update, delete) sunt realizate prin metode standard oferite de SQLAlchemy, toate acestea fiind 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 proiectată pentru a integra ciclul de viață al unei aplicații web cu logica de acces la date, păstrând în același timp caracterul asincron și stateless al cererilor HTTP. La inițializarea aplicației, obiectul SQLAlchemy este configurat cu parametrii de conexiune către baza de date și devine responsabil pentru crearea tabelelor, gestionarea tranzacțiilor și păstrarea integrității operațiilor asupra datelor. Fiecare cerere HTTP care implică o interacțiune cu datele este procesată printr-o sesiune, care captează toate modificările realizate asupra obiectelor model și le persistă în baza de date la finalul cererii, prin commit. Dacă apare o excepție sau o eroare logică, sesiunea poate fi anulată prin rollback, astfel încât datele să nu fie afectate de operații incomplete sau invalide.

Această separare clară a sesiunilor pe fiecare cerere este esențială pentru scalabilitatea aplicației și pentru evitarea blocajelor sau a inconsistențelor. În plus, Flask-SQLAlchemy oferă un set de mecanisme pentru migrarea bazei de date, integrare cu ale extensii precum Flask-Migrate și Flask-Login, și permite utilizarea unor modele de moștenire și relații avansate care reflectă complexitatea logicii aplicației.

*Avantajele utilizării Flask-SQLAlchemy în proiecte web*

Adoptarea extensiei Flask-SQLAlchemy în dezvoltarea aplicațiilor web vine cu o serie de avantaje care o recomandă drept o soluție viabilă atât pentru aplicații de dimensiuni mici, cât și pentru arhitecturi enterprise. Printre beneficiile cele mai evidente se numără simplificarea codului și eliminarea redundanței, deoarece majoritatea operațiunilor de bază sunt gestionate printr-o interfață obiectuală intuitivă. De asemenea, extensia oferă un nivel înalt de coerență între modele și structura bazei de date, reducând semnificativ riscul apariției erorilor de logică sau inconsistențe între nivelul aplicației și cel de stocare. Integrarea facilă cu alte extensii din ecosistemul Flask permite construirea rapidă a aplicațiilor complete, cu autentificare, migrații, cache și validare, toate bazate pe modele comune.

Un alt avantaj important constă în portabilitatea aplicației, întrucât Flask-SQLAlchemy permite comutarea între diverse sisteme de gestiune a bazelor de date prin modificarea simplă a stringului de conexiune, fără a necesita schimbări în codul logic al aplicației. În plus, suportul pentru relații complexe între modele, moștenire multiplă și interogări dinamice prin API-ul ORM-ului, permite dezvoltarea unor aplicații complexe fără a compromite lizibilitatea sau testabilitatea codului.

*Limitări, riscuri și bune practici în utilizarea Flask-SQLAlchemy*

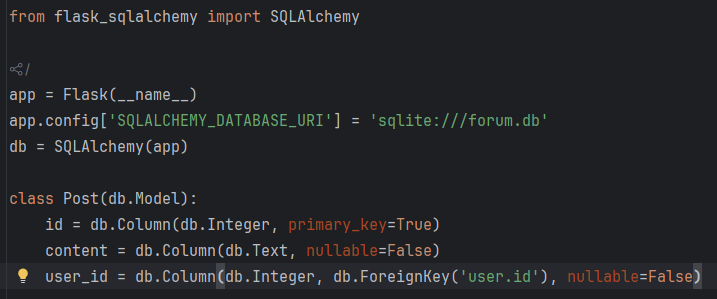
Deși Flask-SQLAlchemy oferă un set impresionant de funcționalități, utilizarea sa necorespunzătoare poate duce la probleme de performanță, securitate sau mentenanță pe termen lung. Una dintre limitările cele mai frecvente este legată de gestionarea ineficientă a interogărilor în aplicațiile mari, în care ORM-ul poate genera SQL-uri suboptime, dacă relațiile dintre modele nu sunt clar definite sau dacă se abuzează de lazy loading în contexte neadecvate. În astfel de cazuri, interogările generate pot duce la N+1 queries, o problemă clasică în ORM-uri, care afectează performanța aplicației 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.

Un alt aspect critic este securitatea. Deși ORM-ul oferă protecție implicită împotriva atacurilor de tip SQL injection, aceasta nu elimină necesitatea validării riguroase a datelor introduse de utilizatori, protejării sesiunilor și criptării datelor sensibile. În plus, orice aplicație care folosește Flask-SQLAlchemy ar trebui să adopte practici solide de versionare a schemei bazei de date, folosind instrumente precum Flask-Migrate, pentru a păstra coerența în mediile de dezvoltare, testare și producție.

*Concluzii*

Flask-SQLAlchemy se impune ca un instrument esențial în dezvoltarea aplicațiilor web cu Python, oferind un echilibru ideal între simplitate, flexibilitate și putere expresivă. Prin integrarea sa elegantă cu framework-ul Flask și prin valorificarea potențialului ORM-ului SQLAlchemy, această extensie permite dezvoltatorilor să construiască aplicații robuste, scalabile și ușor de întreținut, în conformitate cu standardele moderne ale industriei software. În ciuda complexității ascunse în spatele unei interfețe aparent simple, Flask-SQLAlchemy reușește să ofere un model de lucru care încurajează bunele practici de programare și care contribuie la reducerea semnificativă a timpului de dezvoltare. Astfel, adoptarea sa nu reprezintă doar o alegere tehnică, ci și un angajament față de calitate, eficiență și durabilitate în ingineria aplicațiilor web.

*Exemplu*



### 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 epoca dezvoltării rapide a aplicațiilor web, separarea logicii de business de interfața utilizatorului a devenit o practică fundamentală în arhitectura software modernă. Acest principiu de proiectare, cunoscut sub numele de separarea preocupărilor (separation of concerns), permite crearea unor aplicații mai ușor de întreținut, scalabile și testabile. În contextul Python, unul dintre cele mai populare limbaje de programare, dezvoltatorii beneficiază de o serie de unelte și biblioteci care sprijină această abordare modulară. Printre acestea, motorul de template Jinja2 ocupă un loc central, fiind componenta principală utilizată pentru generarea dinamică a conținutului HTML în cadrul framework-urilor web precum Flask. Această lucrare își propune să analizeze în profunzime arhitectura și funcționalitatea motorului Jinja2, să evidențieze avantajele și limitele sale, și să ilustreze 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 rolul Jinja2 în cadrul unei aplicații web, este necesară o incursiune în conceptele de bază ale motoarelor de template. Un motor de template este un sistem software care permite combinarea conținutului static, de regulă în format HTML sau XML, cu date dinamice provenite din aplicație, în vederea generării unei ieșiri complete livrate către utilizatorul final. Această tehnică elimină nevoia inserării codului HTML direct în logica aplicației, facilitând astfel separarea clară între partea de prezentare și cea de procesare. Motoarele de template au evoluat semnificativ în ultimele decenii, de la sisteme rudimentare de concatenare a șirurilor de caractere până la framework-uri expresive și sigure, care oferă suport pentru condiții, iterații, includeri, moștenire de template-uri și chiar apeluri către funcții sau filtre definite de utilizator.

În mediul Python, această nevoie a fost acoperită inițial de template-uri simple precum cele oferite de biblioteca standard string.Template, dar odată cu complexificarea aplicațiilor web, s-a resimțit nevoia unui instrument mai robust. Astfel, Jinja2 a apărut ca o soluție modernă, inspirată de motorul de template Django, dar cu un nivel sporit de flexibilitate și o sintaxă intuitivă care reflectă în mod natural structura HTML.

*Arhitectura și principiile de funcționare ale Jinja2*

Jinja2 este un motor de template puternic, scris în Python, care funcționează prin procesarea unor fișiere template ce conțin atât cod HTML cât și expresii proprii limbajului Jinja, interpretate și înlocuite la runtime cu valori dinamice. La bază, Jinja2 transformă aceste fișiere într-un arbore de sintaxă abstractă, pe care îl compilează apoi într-o funcție Python internă, rezultatul fiind un șir de caractere HTML care reflectă conținutul final prezentat utilizatorului.

Această arhitectură compusă din parser, compilator și executor asigură un nivel ridicat de performanță, deoarece template-urile pot fi precompilate și memorate în cache, reducând astfel timpul de procesare la cereri repetate. Jinja2 oferă un limbaj de template expresiv, care include variabile, blocuri condiționale, bucle, macrocomenzi, extensii personalizate și posibilitatea de a defini filtre. Blocurile de cod sunt delimitate clar prin simboluri speciale ({{ }} pentru expresii, {% %} pentru instrucțiuni și {# #} pentru comentarii), ceea ce permite o integrare elegantă cu HTML-ul fără a compromite lizibilitatea.

Un aspect deosebit de valoros al arhitecturii Jinja2 este sistemul său de moștenire a template-urilor. Acesta permite definirea unui fișier de bază, numit adesea „template master”, care conține secțiunile comune tuturor paginilor, cum ar fi antetul, bara de navigare sau subsolul. Paginile individuale pot extinde acest template, suprascriind doar blocurile relevante, ceea ce reduce duplicarea codului și facilitează întreținerea interfeței.

*Integrarea Jinja2 în Flask și fluxul de generare al paginilor*

Flask utilizează Jinja2 ca motor de template implicit, facilitând astfel generarea conținutului HTML dinamic într-un mod simplu și elegant. În cadrul unei aplicații Flask, interacțiunea cu Jinja2 se realizează prin funcția render\_template, care primește ca parametru un nume de fișier template și un set de variabile ce vor fi disponibile în interiorul acelui template. Aceste variabile pot proveni din baza de date, din logica aplicației sau din input-ul utilizatorului și sunt accesibile în template printr-o sintaxă specifică Jinja2.

Procesul prin care Flask generează o pagină web începe cu recepționarea unei cereri HTTP, urmată de determinarea funcției de view corespunzătoare rutei cerute. Această funcție procesează datele necesare, apoi apelează render\_template, care la rândul său inițiază motorul Jinja2 pentru a combina datele primite cu structura definită în fișierul template. Rezultatul este o pagină HTML completă, returnată clientului ca răspuns al cererii.

Un avantaj major al acestei integrări constă în separarea completă a codului HTML de codul Python, ceea ce permite dezvoltatorilor front-end și back-end să colaboreze eficient pe același proiect, fără a interfera unul cu structura logică a celuilalt. De asemenea, Jinja2 permite folosirea de macro-uri și includeri, permițând astfel reutilizarea componentelor vizuale în diferite secțiuni ale aplicației.

*Securitate, performanță și extensibilitate în Jinja2*

În ceea ce privește securitatea, Jinja2 implementează în mod implicit mecanisme de protecție împotriva unor atacuri frecvent întâlnite în aplicațiile web, cum ar fi Cross-Site Scripting (XSS). Toate valorile introduse în template sunt, prin default, auto-escapate, ceea ce înseamnă că caracterele potențial periculoase sunt convertite într-un format sigur pentru HTML. Acest comportament poate fi dezactivat manual în cazuri specifice, dar este recomandat ca dezvoltatorii să evite acest lucru decât dacă sunt absolut siguri de sursa datelor respective.

Pe partea de performanță, Jinja2 se bazează pe un sistem de caching care stochează template-urile compilate în memorie, astfel încât procesarea să fie rapidă și eficientă chiar și în aplicații cu volum mare de trafic. Flask permite configurarea acestui comportament, oferind opțiuni pentru cache în memorie sau pe disc, în funcție de specificul infrastructurii.

Extensibilitatea Jinja2 este un alt punct forte, deoarece motorul permite definirea de filtre personalizate, funcții globale, testere și extensii, toate acestea permițând dezvoltatorilor să adapteze comportamentul motorului la nevoile aplicației. De exemplu, un proiect care lucrează cu date financiare poate include un filtru personalizat care formatează automat sumele monetare, în timp ce un alt proiect poate avea o funcție care încarcă conținut tradus în funcție de limba utilizatorului.

*Aplicații practice și relevanța Jinja2 în ecosistemul Python*

Motorul Jinja2 este folosit pe scară largă în proiecte dezvoltate cu Flask, dar și în alte framework-uri sau platforme Python care necesită generarea de conținut HTML sau XML. Utilizarea sa este comună în proiecte de tip dashboard, sisteme de management al conținutului, aplicații educaționale, interfețe administrative sau platforme de e-commerce. De asemenea, Jinja2 poate fi utilizat și în afara contextului web, pentru generarea de fișiere de configurare, template-uri de e-mail sau rapoarte PDF, prin conversia prealabilă în HTML și apoi în formatul dorit. Această versatilitate îl face un motor de template extrem de valoros în contextul automatizărilor și al aplicațiilor 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 reprezintă un exemplu elocvent de echilibru între simplitate și putere expresivă, oferind dezvoltatorilor un instrument solid pentru generarea dinamică a interfețelor web. Prin integrarea nativă în framework-uri precum Flask și prin arhitectura sa modulară, Jinja2 sprijină principiile unei dezvoltări clare, organizate și eficiente. Capacitatea sa de a separa logicile aplicației de partea vizuală, împreună cu mecanismele sale de moștenire, filtrare și protecție, îl transformă într-o componentă indispensabilă în dezvoltarea web modernă. În concluzie, Jinja2 nu este doar un motor de template, ci o verigă esențială în lanțul tehnologic care face posibilă construirea unor aplicații web scalabile, sigure și ușor de întreținut.

*Exemplu*

O imagine care conține text, captură de ecran, Font

Conținutul generat de inteligența artificială poate fi incorect.

### Baza de Date SQLite – Arhitectură, Utilizare și Relevanță în Ecosistemul Software Contemporan

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

*Introducere*

În cadrul dezvoltării aplicațiilor software moderne, alegerea unui sistem de gestiune a bazelor de date reprezintă un pas critic ce influențează performanța, scalabilitatea și mentenabilitatea proiectului pe termen lung. Într-un peisaj tehnologic caracterizat prin diversitate și inovație rapidă, sistemele de baze de date relaționale rămân coloana vertebrală a stocării și prelucrării datelor structurate. Deși soluții precum PostgreSQL, MySQL sau Microsoft SQL Server domină segmentul bazelor de date server-based, SQLite oferă o abordare radical diferită, bazată pe simplitate, portabilitate și integrare nativă în aplicație. Această lucrare urmărește să analizeze în profunzime caracteristicile arhitecturale ale SQLite, modul său de funcționare, avantajele și limitele tehnice, precum și scenariile practice în care acest motor de baze de date se dovedește a fi cea mai eficientă alegere.

*Contextul istoric și conceptual al dezvoltării SQLite*

SQLite a fost creat în anul 2000 de către D. Richard Hipp, cu scopul de a oferi un motor de baze de date relaționale care să fie integrabil direct în aplicații, fără a necesita un server dedicat sau configurare externă. Filosofia care a stat la baza proiectării sale a fost aceea de a furniza un instrument care să respecte cât mai fidel standardul SQL, oferind în același timp o implementare ușoară, de tip embedded, care să funcționeze „out-of-the-box” pe orice platformă. De-a lungul anilor, SQLite a evoluat constant, ajungând să fie utilizat în milioane de aplicații, de la browsere web precum Firefox și Chrome, până la sisteme de operare mobile ca Android și iOS. Această răspândire se datorează în mare parte atât licenței publice care permite utilizarea sa gratuită, cât și fiabilității demonstrată în medii critice, inclusiv în aviația comercială, automotive și aplicații industriale.

*Arhitectura internă și particularitățile de implementare*

Spre deosebire de bazele de date clasice care funcționează pe model client-server, SQLite este proiectat ca o bibliotecă C care se integrează direct în codul sursă al aplicației. Astfel, întreaga bază de date este stocată într-un singur fișier pe disc, iar toate operațiile sunt gestionate prin apeluri directe la bibliotecă, fără implicarea unui proces server extern. Acest model de arhitectură face ca SQLite să fie ideal pentru aplicații standalone, mobile, desktop sau embedded, în care resursele hardware sunt limitate sau conectivitatea este instabilă.

Intern, SQLite utilizează un format binar propriu pentru stocarea datelor, proiectat pentru a fi portabil între arhitecturi diferite și rezistent la corupere. Sistemul de tranzacții este implementat printr-un mecanism bazat pe jurnalizare, cu suport pentru atomicitate, consistență, izolare și durabilitate (ACID), asigurând astfel integritatea datelor chiar și în cazul întreruperii neașteptate a aplicației. SQLite suportă atât modul rollback journal, cât și varianta Write-Ahead Logging (WAL), oferind dezvoltatorului posibilitatea de a alege între performanță crescută și fiabilitate sporită, în funcție de contextul aplicației.

*Funcționalități și compatibilitate cu standardul SQL*

Deși dimensiunile sale sunt reduse, SQLite oferă un set impresionant de funcționalități care respectă în mare parte standardele SQL-92, SQL-99 și parțial SQL:2003. Motorul suportă tipuri de date declarative, chei primare și străine, constrângeri de unicitate, agregări, subinterogări, funcții de fereastră și expresii comune. Un aspect distinctiv al SQLite este abordarea sa dinamică asupra tipurilor de date, cunoscută sub numele de „typelessness”, care permite stocarea unor valori de tip diferit în coloane declarate cu un anumit tip, atât timp cât constrângerile definite nu sunt încălcate.

De asemenea, SQLite oferă suport complet pentru tranzacții, văzute ca un set de operații indivizibile care fie se finalizează cu succes, fie se anulează integral. Indicii, triggerele și vizualizările sunt de asemenea disponibile, iar extensibilitatea este facilitată printr-un sistem de funcții definite de utilizator (UDF) și prin posibilitatea de a scrie extensii native în C sau alte limbaje compatibile. Integrarea cu limbajele de programare moderne, în special Python, este excepțional de bine documentată, existând biblioteci oficiale și comunitare care oferă un strat de abstractizare peste API-ul nativ SQLite.

*Avantaje, limitări și scenarii de utilizare*

Printre avantajele notabile ale SQLite se numără ușurința în utilizare, care permite dezvoltatorului să înceapă lucrul imediat, fără configurări sau instalări suplimentare. Fiabilitatea ridicată, dovedită prin utilizarea sa în sisteme critice, îl face o alegere ideală pentru aplicații mobile, dispozitive IoT și aplicații desktop. Dimensiunea redusă a bibliotecii, compatibilitatea multiplatformă și absența unui server de baze de date extern contribuie la eficiența resurselor și la ușurința distribuirii aplicației. Prin faptul că toate datele sunt stocate într-un singur fișier, SQLite facilitează backup-ul, migrarea și replicarea bazelor de date fără eforturi semnificative.

Cu toate acestea, există și limitări care trebuie avute în vedere. SQLite nu este proiectat pentru medii concurente cu un număr mare de scrieri simultane, din cauza blocării la nivel de fișier. Astfel, performanța poate scădea în aplicații multi-user care presupun scrieri frecvente și concurență ridicată. De asemenea, lipsa unui sistem avansat de autentificare și control al accesului face ca SQLite să nu fie adecvat pentru aplicații enterprise unde securitatea granulară este o cerință esențială. În plus, unele funcționalități avansate ale standardului SQL, cum ar fi procedurile stocate sau suportul nativ pentru tipuri de date binare complexe, nu sunt implementate complet.

În practică, SQLite este utilizat frecvent ca soluție de stocare pentru aplicații mobile, în special pe platforma Android, unde servește drept bază de date locală implicită. În aplicațiile web, SQLite este adesea folosit în faza de dezvoltare și testare, datorită simplității și vitezei de configurare. De asemenea, este utilizat pe scară largă în aplicații de tip desktop, instrumente de dezvoltare, clienți de e-mail, browsere web și chiar în unele platforme de testare automatizată a algoritmilor de învățare automată.

*Comparativ cu alte sisteme de gestiune a bazelor de date*

Comparativ cu sistemele tradiționale de gestiune a bazelor de date precum MySQL sau PostgreSQL, SQLite oferă un model de utilizare radical diferit. Dacă primele sunt orientate spre performanță și scalabilitate în medii multi-user și multi-instanță, SQLite este orientat spre simplitate și integrare rapidă, fiind ideal pentru aplicații standalone sau embedded. Spre deosebire de bazele de date server-based care implică conectarea prin protocoale de rețea și gestionarea separată a instanțelor, SQLite funcționează în cadrul procesului aplicației, eliminând astfel latența comunicării inter-proces și permițând o manipulare directă și rapidă a datelor.

În ceea ce privește consumul de resurse, SQLite are un avantaj considerabil, deoarece necesită spațiu minim de stocare, memorie redusă și nu presupune rularea unui daemon de fundal. Totuși, în aplicații care presupun replicare distribuită, control granular al permisiunilor sau execuție de operațiuni complexe în paralel, alte SGDB-uri pot deveni o alegere mai potrivită.

*Concluzii*

SQLite se dovedește a fi un instrument extrem de valoros în arsenalul dezvoltatorului modern, oferind o soluție simplă, robustă și performantă pentru gestionarea datelor structurate într-o varietate de contexte. Fie că este utilizat în aplicații mobile, soluții embedded sau ca mediu de prototipare, SQLite reușește să satisfacă majoritatea cerințelor esențiale fără a introduce complexitate inutilă. Arhitectura sa minimalistă, compatibilitatea cu standardele SQL și fiabilitatea demonstrată în mii de proiecte reale îl transformă într-o alegere pragmatică pentru scenarii în care simplitatea, portabilitatea și eficiența sunt prioritare. În același timp, înțelegerea limitărilor sale și alegerea conștientă a contextului de utilizare sunt elemente-cheie pentru valorificarea optimă a acestei tehnologii. În concluzie, SQLite rămâne o componentă indispensabilă în ecosistemul software contemporan, aducând o contribuție semnificativă la democratizarea accesului la tehnologii de gestiune a datelor.

*Exemplu*

O imagine care conține text, captură de ecran, Font

Conținutul generat de inteligența artificială poate fi incorect.

# Capitol 3. Componente pentru dezvoltare

## Controlul Versiunilor și Git

Controlul versiunilor este un modul software care permite gestionarea fiecărei versiuni a unui fișier sau a unui set de fișiere dintr-o aplicație, oferind informații despre modificările realizate, marcajele de timp și autorii acestora. Un astfel de sistem este cunoscut sub denumirea de **Version Control System (VCS)**. Aceste sisteme sunt utilizate în principal pentru a menține o istorie detaliată a modificărilor aduse codului sursă sau altor documente, facilitând colaborarea și prevenind pierderea datelor.

Git este un exemplu de VCS gratuit și open-source, dezvoltat inițial de Linus Torvalds pentru gestionarea versiunilor codului sursă al kernelului Linux. Datorită eficienței și flexibilității sale, Git este utilizat pe scară largă în diverse proiecte software. Spre deosebire de alte VCS-uri precum Subversion (SVN), Git adoptă o arhitectură distribuită, ceea ce înseamnă că fiecare utilizator deține o copie completă a istoricului repository-ului. Această funcționalitate elimină dependența de un server central și permite lucrul offline.

*GitHub și Integrarea cu Git*

GitHub este o platformă care extinde funcționalitățile Git, combinând gestionarea versiunilor cu instrumente puternice pentru colaborare. Repository-urile Git pot fi găzduite pe GitHub, ceea ce oferă un mediu centralizat unde programatorii pot face backup codului lor, colabora la proiecte comune, propune îmbunătățiri și revizui modificările. GitHub nu doar că sprijină toate funcționalitățile de bază ale Git, dar introduce și caracteristici suplimentare, cum ar fi urmărirea problemelor, gestionarea proiectelor, documentația integrată și multe altele.

#### *Instantaneele în Git*

O caracteristică esențială a Git este modul în care gestionează modificările prin crearea de instantanee (snapshots) ale întregului proiect în loc să țină evidența diferențelor între versiuni, cum fac alte VCS-uri. De fiecare dată când un utilizator face o schimbare și creează un commit, Git salvează un instantaneu complet al stării fișierelor din proiect. Aceasta face ca Git să fie mai rapid și mai eficient în comparație cu alte sisteme. Dacă fișierele nu au fost modificate, Git doar creează o referință către versiunea anterioară, economisind astfel spațiu și resurse.

#### *Arhitectura Distribuită*

Fiind un VCS distribuit, Git permite fiecărui utilizator să aibă o copie completă a repository-ului, incluzând întregul istoric al proiectului. Acest lucru oferă mai multe avantaje:

1. Utilizatorii pot lucra offline, făcând modificări și salvând versiuni local.
2. Sincronizarea cu repository-ul central sau alte repository-uri se face doar când este necesar.
3. Eliminarea dependenței de un server central reduce riscul pierderii de date din cauza erorilor de server.

#### *Mecanismul Hash și Verificarea Integrității*

Git utilizează un mecanism avansat de hash, bazat pe algoritmul SHA-1, pentru a identifica unic fiecare instantaneu creat. Fiecare fișier din repository primește o sumă de control (hash) la momentul commit-ului, iar Git compară aceste hash-uri pentru a detecta modificările. Această metodă asigură integritatea datelor și permite identificarea rapidă a modificărilor aduse fișierelor.

#### *Mecanismul „Cu Trei Stări” din Git*

Git organizează lucrul cu fișierele prin intermediul unui sistem denumit „mecanism cu trei stări”. Acesta include următoarele etape:

1. **Working Directory**: Reprezintă spațiul local unde utilizatorul efectuează modificările asupra fișierelor.
2. **Staging Area**: Este un spațiu temporar unde sunt adăugate fișierele ce urmează să fie incluse într-un commit. Acest proces se face cu comanda git add.
3. **Repository (Committed)**: Este baza de date unde sunt stocate definitiv fișierele și istoricul acestora după comanda git commit.

Fișierele trec prin stările „modificat”, „etapizat” și „committed” în timpul procesului de lucru cu Git. Această organizare permite un control precis asupra modificărilor care vor fi salvate.

#### *Fluxul de Lucru în Git*

Fluxul obișnuit de lucru cu Git include următorii pași:

1. **Modificarea** fișierelor din directorul de lucru.
2. **Adăugarea** fișierelor modificate în zona de staging cu comanda git add.
3. **Crearea** unui commit pentru a salva modificările în istoricul repository-ului utilizând comanda git commit.

#### *Beneficiile Git și GitHub*

Prin natura sa distribuită și eficientă, **Git** oferă avantaje semnificative pentru gestionarea proiectelor software. Platforma GitHub completează această funcționalitate prin instrumente care facilitează colaborarea în echipă, revizuirea codului și gestionarea problemelor. În plus, posibilitatea de a găzdui repository-uri publice și private face ca **GitHub** să fie o alegere ideală pentru proiecte open-source sau comerciale.

**Git** este considerat un standard în gestionarea controlului versiunilor datorită vitezei, flexibilității și capacității sale de a gestiona eficient proiecte de orice dimensiune. **GitHub**, pe de altă parte, contribuie la crearea unei comunități de dezvoltatori în care colaborarea, transparența și inovarea sunt promovate activ.

## Mutarea bazei Mongo in Cloud

Platforma (*DBaaS*) *MongoDB Atlas* a fost a fost dezvoltată ce aceeași dezvoltatori care au proiectat *MongoDB*.

*Atlas* va oferi instanței tot ce avem nevoie după construirea bazei de date și vom oferi instanței următoarele:

Un cloud privat virtual (*VPC*) este utilizat pentru a izola rețeaua prin implementarea instanțelor de baze de date cu măsuri de securitate automate. Printre măsurile de securitate suplimentare se numără

liste de adrese *IP* care au acces la bază, autentificare în curs de desfășurare, criptare în tranzit și în repaus, acces complex de management bazat pe roluri și alte caracteristici de securitate.

Servere suplimentare sunt oferite de către platforma , pentru a fi mereu la îndemâna continuă, asigurând că aplicația continuă să funcționeze pana când chiar si cel principal e offline.

Sistemul *MongoDB* Atlas utilizează copii de rezervă și recuperare la un moment dat pentru a se proteja cu fermitate împotriva coruperii datelor, care se poate întâmpla ca urmare a unei greșeli umane sau virtual. Aceasta se numește replicare pre-făcută.

Date care sunt aranjate într-o varietate de moduri pe care le putem folosi pentru a determina când este timpul să trecem lucrurile la următorul nivel (pot fi furnizate mai multe cazuri pe măsură ce numărul utilizatorilor crește), aceasta se numește monitorizare.

Atlas folosește o metodă cunoscută sub numele de sharding pentru a permite scalarea orizontală pentru bazele de date. Datele sunt dispersate pe fragmente, care sunt colecții de replici. In funcție de

volumul datelor care crește, echilibrarea automată în *MongoDB* garantează că simt distribuite uniform datele. Fără a crește complexitatea aplicației, shardingul permite instalațiilor *Mongo sa crească* dincolo de constrângerile de a fi doar un singur server.

*AWS, Azure sau Google Cloud* pot fi baza pentru serviciul *MongoDB Atlas.*

Serviciul va fi implementat, rulat folosind Mongo. Vom folosi *Amazon Web Services (AWS)* pentru proiectul curent pentru a găzdui baza de date cât mai aproape posibil fata de locația noastră curenta.

Baza de date o sa ruleze pe o *VM(Virtual Machine)* care in versiunea de baza poate stoca pana la 500 de colecții si conexiuni in același timp.

# Capitol 4. Descrierea aplicatiei

## Descriere a utilității practice

O platformă Q&A dedicată programării este un spațiu online unde programatorii de toate

nivelurile pot pune și primi răspunsuri la întrebări legate de programare. Această platformă poate fi un instrument valoros pentru învățarea de noi concepte, rezolvarea problemelor complexe și conectarea cu alți programatori.

Iată câteva caracteristici cheie ale unei platforme Q&A despre programare:

* **Un sistem de întrebări și răspunsuri:** Utilizatorii pot pune întrebări detaliate despre programare, iar alți utilizatori pot răspunde cu soluții și explicații.
* **Un sistem de voturi:** Utilizatorii pot vota pentru răspunsurile pe care le consideră utile și informative, ajutând la evidențierea celor mai bune răspunsuri.
* **Un sistem de etichete:** Întrebările și răspunsurile pot fi etichetate cu cuvinte cheie relevante, facilitând navigarea și găsirea informațiilor specifice.
* **Un forum de discuții:** Utilizatorii pot discuta despre diverse subiecte legate de programare, pot împărtăși sfaturi și trucuri și pot construi relații cu alți programatori.
* **Un sistem de căutare:** Utilizatorii pot căuta întrebări și răspunsuri existente pe baza cuvintelor cheie, subiectelor sau limbajelor de programare.
* **Un sistem de notificări:** Utilizatorii pot primi notificări atunci când primesc răspunsuri la întrebările lor sau când sunt menționați în discuții.

Beneficiile utilizării unei platforme Q&A despre programare**:**

* **Îmbunătățirea abilităților de programare:** Prin punerea și primirea de răspunsuri la întrebări, programatorii pot învăța concepte noi, pot rezolva probleme complexe și pot dobândi noi abilități.
* **Creșterea productivității:** Platformele Q&A pot ajuta programatorii să găsească rapid și ușor soluții la problemele lor, economisind timp și efort.
* **Conectarea cu alți programatori:** Platformele Q&A pot facilita conectarea programatorilor cu

alți programatori din întreaga lume, oferind o oportunitate de a colabora, de a împărtăși cunoștințe și de a construi relații profesionale.

* **Rămânerea la curent cu cele mai noi tendințe:** Platformele Q&A pot fi o sursă excelentă de informații despre cele mai noi tendințe și dezvoltări din domeniul programării

##### Exemple de platforme Q&A populare despre programare:

* **Stack Overflow:** <https://stackoverflow.com/>
* **Quora:** <https://www.quora.com/>
* **Reddit:** <https://www.reddit.com/>
* **GitHub:** <https://github.com/>
* **Programmers Forum:** <https://www.thecodingforums.com/>

##### Sfaturi pentru utilizarea unei platforme Q&A despre programare:

* **Puneți întrebări clare și concise:** Formulați întrebările într-un mod clar și concis, oferind cât mai multe detalii posibil pentru a facilita identificarea problemei.
* **Utilizați etichete relevante:** Etichetați întrebările cu cuvinte cheie relevante pentru a facilita găsirea răspunsurilor de către alți utilizatori.
* **Fiți respectuoși:** Respectați alți utilizatori, chiar dacă nu sunteți de acord cu opiniile lor.
* **Oferiți feedback:** Votați pentru răspunsurile pe care le considerați utile și oferiți feedback constructiv altor utilizatori.

O platformă Q&A despre programare poate fi un instrument valoros pentru orice programator, indiferent de nivelul de experiență. Prin utilizarea acestor platforme, programatorii pot învăța concepte noi, pot

rezolva probleme complexe, pot conecta cu alți programatori și pot rămâne la curent cu cele mai noi tendințe din domeniul programării.

## Proiectarea si Implementarea(MongoDB)

La crearea unui cont nou, un document este pus în ultima colecție, colecția utilizatorilor.

Următoarele câmpuri sunt prezente în document pentru un utilizator înregistrat:



Figură 12 - Colecția User-ului in baza de date

„*ID*-*ul*” fiind primul parametru, este distinct pentru fiecare utilizator, cum am văzut deja legătura între lucrările din colecțiile anterioare.

Al doilea parametru „*salt*”; conține aleatoriu o valoare care este folosit pentru a cripta parola

înainte de a fi introdusa. În secțiunea privind problemele legate de securitate, vom aprofunda acest lucru. Câmpul „*hash*” (*amprenta digitală*) a parolei, care a fost produsă printr-o funcție unidirecțională, se află în câmpul următor.

*Figura 13* ilustrează cum Mongo comunică cu celelalte componente ale programului.

O imagine care conține text, diagramă, captură de ecran, Plan

Descriere generată automat

Figură 13 - Proiectarea arhitecturii

#### Mongoose (Modelele concepute)

*Mongoose* este o bibliotecă de modelare a datelor ODM amintita mai devreme, atât în Node cat și Mongo. Oferă scheme de validare, gestionează asocierile de date și tranzițiile între obiectele si scheme lor.

Modelele pot fi considerate constructori creați din definiții ale schemelor. Un document este o anumită instanță a unui model. Responsabilitatea modelelor este citirea/scrierea datelor in baza de date Mongo.

Ar trebui să le separăm pentru fiecare tip de „*obiect*” în timp ce ne creăm modelele, este doar logic (informații legate). *Subiectele de discuție, comentariile și utilizatorii* sunt posibilitățile evidente în situația noastră pentru aceste modele. Pentru a ne ajuta să introducem postări în baza de date, am inclus următoarele modele:

O imagine care conține text, captură de ecran, Font, număr

Descriere generată automat

Figură 14 - Schema Mongoose a Topicului de discuție

**

*Figură 14 - Schema Mongoose a Comentariului*

Figură 15 - Schema Mongoose a User-ului

## Arhitectura aplicației

*Modelele* și metodele utilizate în proiectarea și construcția unei aplicații sunt descrise în arhitectura aplicației. Pentru a crea o aplicație care este bine structurată și simplu de întreținut, este necesara o arhitectura minimalista.

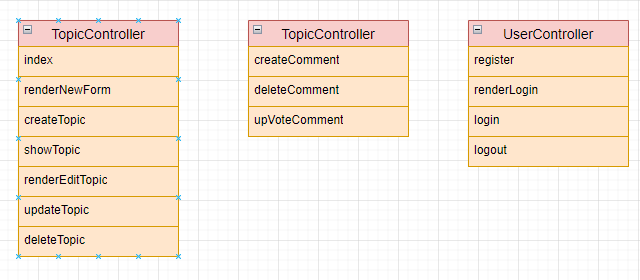
*Figura 16* arata arhitectura aplicației, aceasta folosește o serie de concepte și idei, in care implementarea va fi discutata pe scurt în paragrafele care urmează.

Am acoperit anterior elementele fundamentale ale bazei de date (*MongoDB* și *Mongoose*), așa că vom trece la *controller*, *views* și *rute*.

### Controller

Un *controler*, în general, este un grup de funcții care separă *codul* destinat direcționării cererilor de codul destinat executării respectivelor cereri.

Colecția de funcții care compun controlerul primește solicitări de la modele, produce o pagină cu informațiile necesare si apoi trimite pagina si conținutul acesteia către client.



Figură 16 - Organizarea Controllerelor

### Rutare

*URI-urile* unui site care reacționează pentru interogările utilizatorului este denumit *rutare*.

*Rutarea* se realizează folosind metode obiectului aplicației *Express* care sunt echivalente cu metodele

*HTTP*, cum ar fi app.get() pentru cererile „*GET*” și *app*.*post*() pentru cererile „*POST*”.

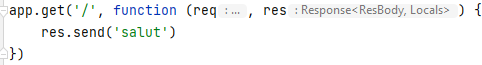
Când o aplicație primește o solicitare specifică metodei *HTTP* către ruta dată (punctul final),

funcția call-back definită de aceste metode de rutare - denumită adesea „*handler*” - este invocată. Cu alte cuvinte, software-ul „ascultă” în esență request-urile care se mulează pe anumite rute, iar in momentul in

care detectează o potrivire si rulează funcția necesara.

De exemplu, ruta unei solicitări „*GET*” pentru pagina principală cu mesajul „bună ziua” arată

astfel:



Figură 17 - URI

Rutele sunt fracționate in 3 documente pentru aplicația construită pentru a permite întreținerea.

Un prim fișier conține rutele utilizate pentru acțiunile de locație (*vederea*, *adăugarea*, remodelarea,

*ștergerea*). Acest lucru a fost realizat dezactivând rutele pentru *review*-uri, care sunt stocate în alt document, dar si cele care se ocupa de *autentificarea user*-ului, care sunt, de asemenea, adunate într-un alt fișier.

##### Ruta de înregistrare a unui nou utilizator

Acest modul definește rutele pentru autentificarea utilizatorilor într-o aplicație web bazată pe framework-ul **Express**, oferind funcționalitatea de înregistrare a utilizatorilor. Implementarea include procese esențiale precum validarea datelor introduse de utilizator, verificarea existenței unui cont asociat adresei de email furnizate, criptarea parolelor pentru securitate sporită și generarea **unui token JWT (JSON Web Token)** utilizat pentru autentificare.

Modulul utilizează mai multe librării pentru a atinge aceste obiective. **Express** facilitează gestionarea cererilor și răspunsurilor **HTTP**, **bcryptjs** este responsabil pentru criptarea parolelor, în timp ce **jsonwebtoken** generează token-urile necesare autentificării. De asemenea, modelul *User* este utilizat pentru a interacționa cu baza de date **MongoDB**, stocând informațiile despre utilizatori. Routerul **Express** este inițializat pentru a defini rutele dedicate autentificării, cum ar fi ruta **/register**, care implementează logica înregistrării utilizatorilor.

Ruta /register folosește metoda **HTTP POST** și primește date în format **JSON**. Corpul cererii trebuie să conțină un nume de utilizator (*username*), o adresă de email (*email*) și o parolă (*password*). În procesul de înregistrare, sistemul verifică mai întâi dacă adresa de email este deja asociată unui cont existent. Dacă da, se returnează un răspuns cu statusul **HTTP 400** și mesajul "*Utilizatorul există deja*". În cazul în care adresa este validă, parola este criptată utilizând un salt generat dinamic, oferind astfel un nivel ridicat de protecție a datelor.

După criptarea parolei, este creat un nou document pentru utilizator, care este apoi salvat în baza de date. După succesul acestei operațiuni, se generează un token **JWT**, conținând identificatorul unic al utilizatorului. Token-ul este semnat cu o cheie secretă definită în variabila de mediu și are o durată de valabilitate configurată, în mod uzual de o oră. **Token-ul** este returnat utilizatorului în cazul unei înregistrări reușite, împreună cu statusul **HTTP 200**.

În cazul apariției unor erori neașteptate pe parcursul procesului, acestea sunt capturate și returnate sub forma unui mesaj generic cu statusul **HTTP 500**, indicând o eroare de server. Pentru a facilita utilizarea, cererile către ruta /register trebuie să includă un header de tip **Content-Type: application/json** și să conțină în corp datele corespunzătoare unui utilizator.

Pentru a îmbunătăți implementarea, se recomandă utilizarea unor librării precum **Joi** sau **express-validator** pentru validarea datelor de intrare, ceea ce ar asigura conformitatea cu cerințele aplicației. De asemenea, gestionarea erorilor poate fi extinsă pentru a oferi mesaje mai detaliate, iar durata token-ului **JWT** ar putea fi configurată în mod dinamic, în funcție de nevoile aplicației. Testele unitare și de integrare ar trebui implementate pentru a verifica funcționalitatea rutei **/register**, iar adăugarea unor mecanisme de securitate suplimentare, cum ar fi rate-limiting, ar preveni atacurile de tip brute-force. Această abordare modulară asigură o bază solidă pentru extinderea funcționalităților autentificării, oferind în același timp un standard ridicat de securitate și fiabilitate.

*O imagine care conține text, captură de ecran, Font, software

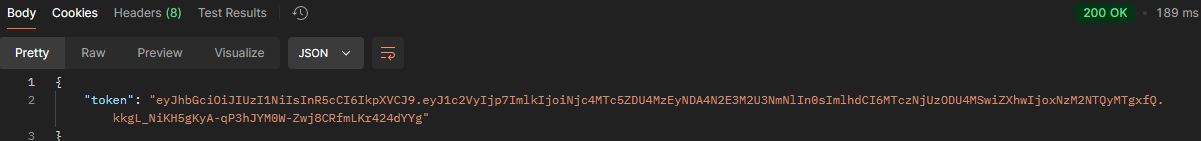
Descriere generată automat*

Figură 18 - Înregistrarea unui nou utilizator

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat

Figură 19 - Rută POST /register



Figură 20 - Răspuns /register

##### Rută de login

Această rută implementează funcționalitatea de autentificare a utilizatorilor utilizând module externe specializate, cum ar fi **bcryptjs**, pentru compararea parolelor introduse de utilizator cu cele criptate și stocate în baza de date, și **jsonwebtoken**, pentru generarea unui token JWT utilizat în accesarea resurselor protejate.

Totodată, modelul **MongoDB** **User** este utilizat pentru gestionarea informațiilor despre utilizatori, permițând interogarea bazei de date pentru autentificare.

Ruta /login, definită în cadrul aplicației, folosește metoda **HTTP POST** și este configurată pentru a primi cereri în format **JSON**. În corpul cererii sunt necesari doi parametri principali: adresa de email a utilizatorului și parola acestuia, furnizată în format necriptat.

Procesul de autentificare începe prin căutarea utilizatorului în baza de date pe baza adresei de email. Dacă nu este găsit un utilizator asociat, serverul returnează un răspuns cu **statusul HTTP 400**, indicând că utilizatorul nu există. În cazul în care utilizatorul este identificat, serverul verifică validitatea parolei introduse prin compararea acesteia cu versiunea criptată din baza de date, utilizând funcția bcrypt.compare. Dacă parola nu corespunde, se returnează un răspuns similar cu statusul **HTTP 400** și mesajul „*Parolă invalidă*”.

Dacă atât utilizatorul, cât și parola sunt validați, următorul pas constă în generarea unui token **JWT**. Acest token conține un payload cu ID-ul unic al utilizatorului și este semnat utilizând o cheie secretă, stocată în variabila de mediu **JWT\_SECRET**. Token-ul are o valabilitate prestabilită, de obicei de o oră, și este returnat clientului cu un răspuns de succes și statusul **HTTP 200**. Token-ul poate fi utilizat ulterior pentru autentificarea cererilor către resursele protejate ale aplicației.

În cazul apariției unor erori neprevăzute pe parcursul procesului de autentificare, acestea sunt gestionate prin returnarea unui răspuns cu statusul **HTTP 500**, indicând o eroare de server. Această abordare asigură că utilizatorii primesc feedback corespunzător asupra stării cererii lor.

Fluxul de date implică mai întâi trimiterea unei cereri **POST** din partea clientului către ruta **/login**, care include email-ul și parola utilizatorului. Serverul validează datele prin verificarea existenței utilizatorului în baza de date, urmată de compararea parolei utilizând **bcrypt**. Dacă toate verificările sunt trecute cu succes, se generează și se trimite un token **JWT** clientului. În caz contrar, se returnează mesaje descriptive pentru a informa utilizatorul despre natura problemei întâmpinate.

Pentru îmbunătățirea funcționalității, se poate implementa validarea inputului utilizând librării precum **Joi** sau **express-validator**, pentru a verifica atât formatul datelor, cât și conformitatea lor cu cerințele aplicației. De exemplu, validarea ar putea include verificarea unui format corect al email-ului sau asigurarea unui număr minim de caractere pentru parolă. De asemenea, adăugarea unui mecanism de rate-limiting ar preveni atacurile de tip brute-force prin limitarea numărului de cereri de autentificare pe o perioadă determinată.

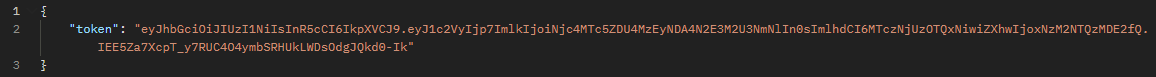
O altă îmbunătățire ar fi configurarea dinamică a duratei token-ului, astfel încât să fie adaptabilă în funcție de utilizarea specifică. Pentru o securitate sporită, se poate introduce autentificarea multi-factor **(MFA),** adăugând un strat suplimentar de protecție la procesul de autentificare. În plus, logarea detaliată a erorilor cu ajutorul unei librării precum **winston** ar putea îmbunătăți monitorizarea aplicației și depistarea problemelor.

Acest flux de autentificare oferă un echilibru optim între performanță și securitate, utilizând cele mai bune practici din domeniul dezvoltării aplicațiilor web. Totodată, oferă o bază solidă pentru extinderea funcționalităților viitoare, cum ar fi integrarea unor caracteristici suplimentare de securitate sau personalizare a experienței utilizatorului.

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat

Figură 21 - Rută POST /login



Figură 22 - Răspuns /login



Figură 23 - Logarea unui utilizator

##### Ruta de creare postare nouă

Acest fragment de cod reprezintă un endpoint creat pentru o aplicație **Node.js/Express**, destinat adăugării unei noi postări într-o bază de date, integrând middleware-uri pentru autentificare și validarea datelor, precum și mecanisme pentru gestionarea erorilor, astfel asigurând funcționalitatea și securitatea aplicației.

Endpoint-ul, definit pentru metoda **POST** și accesibil la rădăcina aplicației, primește date în format **JSON** prin intermediul corpului cererii și utilizează middleware-ul de autentificare pentru a verifica dacă utilizatorul este logat, stocând ID-ul acestuia în obiectul **req.user**. În plus, validarea intrărilor este realizată cu ajutorul bibliotecii **express-validator**, care verifică prezența câmpurilor esențiale, cum ar fi title și content. În cazul lipsei acestora, răspunsul este un **cod HTTP 400 (Bad Request**), însoțit de o listă detaliată a erorilor sub formă de **JSON**.

Logica principală a endpoint-ului implică extragerea erorilor de validare utilizând metoda **validationResult(req)** și oprirea execuției dacă acestea există. În caz contrar, câmpurile title și content sunt extrase din cerere, iar un obiect Post este creat cu ajutorul modelului corespunzător, incluzând ID-ul utilizatorului drept autor.

Obiectul este salvat asincron în baza de date, iar postarea nou creată este returnată clientului în format **JSON** cu un **cod HTTP 200 (OK).** Dacă apare o eroare neașteptată în timpul procesării, un bloc try-catch gestionează situația, returnând un răspuns **HTTP 500 (Server Error)** cu mesajul "*Eroare de server*".

Structura răspunsurilor posibile include: un mesaj de succes cu detaliile postării **(200 OK),** erori de validare cu detalii despre câmpurile problematice **(400 Bad Request),** și un mesaj generic în cazul unei erori de server **(500 Internal Server Error).** Aplicația utilizează dependințe precum express pentru crearea serverului, express-validator pentru validarea datelor și middleware-uri personalizate pentru autentificare. Modelul pentru postare, denumit **Post**, este presupus a fi definit într-un fișier separat și este utilizat pentru gestionarea obiectelor salvate în baza de date. Acest design oferă o implementare clară și robustă pentru adăugarea de conținut într-o aplicație modernă bazată pe **Node.js.**

O imagine care conține text, software, Software multimedia, captură de ecran

Descriere generată automat

Figură 24 - Token Autorizare Creare postare

O imagine care conține text, captură de ecran, software, Font

Descriere generată automat

Figură 25 - Rută Creare Postare

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat

Figură 26 - Răspuns Creare Postare

O imagine care conține text, captură de ecran, software, număr

Descriere generată automat

Figură 27 - Crearea unei postări

##### Ruta de adăugare comentariu postare

Această documentație detaliază implementarea rutei **POST** **/id/comments**, care permite adăugarea unui comentariu la o postare specifică, fiind accesibilă exclusiv utilizatorilor autentificați. **Ruta** este construită pentru a asigura validarea autentificării, verificarea corectitudinii datelor de intrare, căutarea postării în baza de date, adăugarea comentariului la lista de comentarii și gestionarea erorilor pentru a returna răspunsuri adecvate în funcție de situație.

Procesul începe cu validarea autentificării utilizatorului prin intermediul unui middleware personalizat denumit **auth**. Acesta verifică token-ul **JWT** prezent în cerere, iar dacă este valid, atașează informațiile utilizatorului la obiectul **req**. În paralel, pentru asigurarea validității datelor introduse, sunt utilizate funcționalitățile check și validationResult din biblioteca express-validator. Acestea verifică prezența câmpului obligatoriu content în corpul cererii. Dacă validarea eșuează, serverul **returnează** un răspuns cu statusul **HTTP 400** și detalii despre erorile detectate.

După ce autentificarea și validarea datelor sunt confirmate, aplicația caută postarea în baza de date folosind **ID**-ul furnizat ca parametru în **URL**. În cazul în care postarea nu este găsită, serverul trimite un răspuns cu statusul **HTTP 404** și un mesaj informativ specificând că postarea nu există.

Dacă postarea este identificată, se creează un nou obiect de tip comentariu, care include **ID**-ul utilizatorului autentificat ca autor și textul comentariului preluat din corpul cererii. Acest comentariu este adăugat la lista de comentarii a postării respective.

Odată ce comentariul este atașat, modificările sunt salvate în baza de date utilizând metoda **save**. Serverul returnează apoi postarea actualizată, care include lista completă a comentariilor, inclusiv cel nou adăugat, împreună cu un răspuns cu statusul **HTTP 200**.

Dacă în timpul procesului de salvare a datelor sau al altor operațiuni apare o eroare neașteptată, aceasta este capturată și gestionată corespunzător, iar serverul trimite un răspuns generic cu statusul **HTTP 500**, informând utilizatorul că a apărut o eroare de server.

Această rută utilizează o arhitectură robustă și modulară, ceea ce facilitează extinderea și menținerea ulterioară. Fluxul de date include primirea unei cereri **POST** de la client, care trebuie să conțină un token **JWT** valid în antetul cererii, precum și textul comentariului în corpul acesteia.

Serverul validează aceste informații, caută postarea după **ID**, și dacă toate verificările sunt satisfăcute, adaugă comentariul și returnează datele actualizate. În cazul în care una dintre aceste etape eșuează, serverul trimite răspunsuri clare și descriptive pentru a informa utilizatorul despre natura problemei.

Pentru a îmbunătăți această implementare, se poate introduce o validare mai detaliată a datelor, cum ar fi verificarea lungimii maxime admise a comentariului, prevenind astfel intrările excesiv de lungi. În cazul postărilor cu un număr mare de comentarii, se poate adăuga o funcționalitate de paginare, care să permită gestionarea eficientă a afișării și manipulării acestora.

De asemenea, utilizarea unei librării precum winston pentru logarea erorilor ar permite o monitorizare mai eficientă a problemelor apărute în timpul rulării aplicației. Aceste îmbunătățiri ar crește considerabil securitatea, performanța și ușurința de utilizare a sistemului, asigurând o experiență mai bună atât pentru dezvoltatori, cât și pentru utilizatori.



Figură 28 - Rută Creare Comentariu

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat

Figură 29 - Răspuns Creare Comentariu

O imagine care conține text, captură de ecran, software, număr

Descriere generată automat

Figură 30 - Crearea unui comentariu

##### Ruta de editare a unei postări de către autor

Această rută implementează funcționalitatea de actualizare a unei postări existente, fiind concepută astfel încât doar autorul postării să poată efectua modificările. Logica rutei asigură verificarea drepturilor de acces ale utilizatorului autentificat, actualizarea câmpurilor relevante ale postării și gestionarea erorilor care pot apărea în proces. Sistemul utilizează middleware-ul de autentificare auth, care validează token-ul **JWT** al utilizatorului și atașează datele acestuia la obiectul cererii. Aceasta reprezintă o măsură esențială pentru securizarea procesului și prevenirea accesului neautorizat.

La primirea unei cereri **PUT**, aplicația identifică postarea ce urmează a fi modificată, folosind **ID**-ul specificat în parametrii URL. Dacă postarea nu este găsită în baza de date, serverul returnează un răspuns cu statusul **HTTP 404** și un mesaj informativ care precizează că postarea nu există. În continuare, se efectuează o verificare a drepturilor de acces, comparând **ID**-ul utilizatorului autentificat cu cel al autorului postării. Dacă utilizatorul nu este autorul, cererea este respinsă cu statusul **HTTP 403**, iar serverul trimite un mesaj care indică lipsa permisiunii de editare.

În cazul în care toate verificările sunt trecute, aplicația procedează la actualizarea câmpurilor specificate în corpul cererii. Modificările sunt limitate la câmpurile title și content, care reprezintă titlul și conținutul postării. Dacă unul dintre aceste câmpuri nu este furnizat în cerere, valoarea existentă în baza de date rămâne neschimbată. După actualizare, postarea este salvată în baza de date, iar datele actualizate sunt returnate clientului într-un răspuns cu statusul **HTTP 200**, indicând succesul operațiunii.

Gestionarea erorilor este integrată în proces pentru a preveni întreruperile neprevăzute. Dacă apare o eroare internă în timpul căutării postării, validării drepturilor de acces sau actualizării datelor, aceasta este capturată, iar serverul returnează un răspuns generic cu statusul **HTTP 500**, informând clientul că a apărut o eroare de server. Această abordare standardizată permite dezvoltatorilor să identifice rapid și să remedieze problemele din aplicație.

Fluxul complet al rutei începe cu trimiterea unei cereri **PUT** din partea clientului, care include un token **JWT** valid și noile date pentru actualizarea postării. **Serverul** validează mai întâi autentificarea utilizatorului și apoi caută postarea în baza de date. După ce identifică postarea, sistemul verifică dacă utilizatorul are drepturi de editare, iar în caz afirmativ, actualizează datele și le salvează în baza de date. La final, datele postării actualizate sunt trimise înapoi clientului.

Pentru a îmbunătăți această implementare, se poate adăuga un mecanism de validare mai complex al câmpurilor title și content, verificând, de exemplu, lungimea minimă sau maximă a acestora. Acest lucru ar asigura că datele introduse respectă standardele definite ale aplicației.

De asemenea, utilizarea unei librării pentru logarea erorilor, cum ar fi **winston**, ar permite monitorizarea mai detaliată a incidentelor și a activității sistemului. O altă îmbunătățire ar fi implementarea unui sistem de stocare a versiunilor anterioare ale postării, astfel încât să fie posibilă restaurarea modificărilor sau auditarea acestora, ceea ce ar îmbunătăți transparența și flexibilitatea aplicației.

Această **rută**, prin design-ul său, asigură un echilibru între funcționalitate, securitate și performanță, fiind în același timp extensibilă și ușor de întreținut. Extinderea acesteia cu funcționalități suplimentare poate îmbunătăți considerabil experiența utilizatorilor și eficiența dezvoltatorilor care lucrează la proiect.

O imagine care conține text, captură de ecran, Font, număr

Descriere generată automat

Figură 31 - Editarea unei Postări de către autor



Figură 32 - Rută editarea unei Postări

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat

Figură 33 - Răspuns Editare Postare

##### Ruta de ștergere a unei postări de către autor

Această rută oferă funcționalitatea de ștergere a unei postări existente, însă doar autorul postării are permisiunea de a efectua această acțiune. Procesul presupune autentificarea obligatorie a utilizatorului, ceea ce înseamnă că fiecare cerere trebuie să fie însoțită de un token **JWT** valid. În plus, logica rutei integrează pași esențiali, precum verificarea existenței postării, validarea drepturilor de acces ale utilizatorului autentificat și ștergerea propriu-zisă a postării din baza de date. Aceasta este o funcționalitate crucială pentru gestionarea responsabilă și controlată a conținutului într-o aplicație web.

La primirea unei cereri **DELETE** către ruta specificată, aplicația începe prin verificarea autentificării utilizatorului utilizând middleware-ul personalizat auth. Acest middleware validează token-ul **JWT** furnizat în antetul cererii și atașează informațiile utilizatorului autentificat la obiectul cererii. După ce autentificarea este confirmată, aplicația caută postarea care trebuie ștearsă, folosind **ID**-ul unic al acesteia, specificat în parametrii URL. Dacă postarea nu este găsită în baza de date, serverul trimite un răspuns cu statusul **HTTP 404** și un mesaj clar care informează clientul că postarea nu există.

Următorul pas constă în verificarea drepturilor de acces. Aplicația compară **ID**-ul utilizatorului autentificat cu **ID**-ul autorului postării. Dacă utilizatorul care face cererea nu este autorul, cererea este respinsă, iar serverul returnează un răspuns cu statusul **HTTP 403**, indicând faptul că utilizatorul nu are permisiunea de a efectua această acțiune. Această verificare asigură că doar autorii își pot gestiona conținutul și previne accesul neautorizat.

Dacă toate verificările sunt trecute cu succes, postarea este eliminată definitiv din baza de date. După finalizarea operațiunii de ștergere, serverul trimite un răspuns cu statusul **HTTP 200**, conținând un mesaj care confirmă că postarea a fost ștearsă cu succes. Această confirmare oferă utilizatorului feedback clar despre rezultatul cererii.

În cazul în care apare o eroare pe parcursul oricărei etape, cum ar fi în timpul procesului de autentificare, căutare sau ștergere, aceasta este gestionată prin capturarea erorii și returnarea unui răspuns generic cu statusul **HTTP 500**. Mesajul transmis clientului indică faptul că a apărut o problemă internă a serverului, fără a dezvălui detalii tehnice sensibile. Această abordare nu doar că protejează aplicația, dar oferă și o experiență mai profesionistă utilizatorului final.

Fluxul complet al acestei rute presupune mai multe etape bine definite: utilizatorul trimite o cerere **DELETE** însoțită de un token **JWT** valid; serverul validează autentificarea și drepturile de acces; postarea este identificată după **ID**-ul specificat; dacă toate condițiile sunt îndeplinite, postarea este ștearsă; și, în final, clientul primește un răspuns care indică succesul sau eșecul operațiunii.

Pentru a îmbunătăți funcționalitatea, se pot implementa mai multe optimizări. De exemplu, utilizarea unei librării precum winston pentru logarea detaliată a erorilor ar permite o monitorizare mai bună și diagnosticarea rapidă a problemelor. În plus, un mecanism de confirmare a ștergerii, prin care utilizatorul trebuie să aprobe explicit înainte de ștergerea definitivă a postării, ar reduce riscul ștergerilor accidentale.

O altă îmbunătățire ar fi implementarea unui sistem de „*ștergere moale*”, prin care postările nu sunt eliminate definitiv, ci marcate ca „inactivate” sau „*șterse*”. Acest lucru ar permite restaurarea lor ulterioară, oferind o soluție mai flexibilă și mai sigură. De asemenea, se poate introduce un jurnal al acțiunilor de ștergere, stocând informații precum ID-ul utilizatorului care a efectuat acțiunea și momentul în care aceasta a avut loc, ceea ce ar facilita auditarea și urmărirea modificărilor.

Această rută este un exemplu de funcționalitate esențială care respectă principiile de securitate, claritate și eficiență. Totodată, prin adăugarea funcționalităților suplimentare, aceasta poate deveni un instrument și mai robust pentru gestionarea conținutului, crescând valoarea și utilitatea aplicației pentru utilizatorii finali.

O imagine care conține text, captură de ecran, Font, număr

Descriere generată automat

Figură 34 - Ștergerea unei Postări de către autor

O imagine care conține text, software, Software multimedia, captură de ecran

Descriere generată automat

Figură 35 - Rută de Ștergere a unei Postări

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat

Figură 36 - Răspuns ștergere postare

##### Ruta de obținere a tuturor postărilor

Această rută este proiectată pentru a permite obținerea tuturor postărilor din baza de date într-un mod eficient și detaliat, returnând informații complete despre fiecare postare împreună cu numele de utilizator al autorului. Prin utilizarea metodei populate, datele despre autor sunt extinse, oferind o imagine mai cuprinzătoare asupra conținutului și contextului fiecărei postări. Această funcționalitate este deosebit de utilă în aplicațiile care necesită prezentarea detaliată a conținutului generat de utilizatori.

Implementarea acestei **rute** utilizează modelul MongoDB denumit **Post**, care gestionează structura și manipularea datelor despre postări în baza de date. Ruta este accesată prin metoda **HTTP GET la URL-ul rădăcină (/)** și nu necesită parametri suplimentari în cerere, făcând-o ușor accesibilă pentru orice client autorizat să vizualizeze conținutul. Rezultatul returnat este o listă de obiecte **JSON**, fiecare reprezentând o postare și incluzând detalii precum **titlul**, **conținutul** și **informațiile** autorului (numele de utilizator).

La primirea unei cereri **GET**, serverul interoghează baza de date pentru a găsi toate postările existente. Acest lucru se realizează prin metoda find aplicată pe modelul **Post**. Pentru a extinde informațiile disponibile în răspuns, metoda populate este utilizată pentru a include detalii suplimentare despre autor, extrăgând doar câmpul **username** din documentul asociat. Această abordare eficientizează procesul de interogare, furnizând doar informațiile relevante, fără a încărca inutil răspunsul cu date suplimentare.

Rezultatul obținut este apoi convertit într-un format **JSON** și trimis ca răspuns clientului. Dacă operațiunea se finalizează cu succes, serverul returnează un răspuns cu statusul **HTTP 200**, indicând faptul că cererea a fost procesată corect, iar postările au fost returnate cu succes. Totuși, în cazul în care apare o eroare în timpul procesării, aceasta este gestionată corespunzător. Orice excepție este capturată, iar serverul trimite un răspuns generic cu statusul **HTTP 500**, însoțit de un mesaj care informează clientul despre existența unei erori interne a serverului.

Fluxul de procesare pentru această rută include mai mulți pași clar definiți: clientul trimite o cerere **GET** către endpoint-ul rădăcină; serverul interoghează baza de date folosind modelul **Post**; fiecare postare este îmbogățită cu informații despre autor prin metoda populate; lista completă a postărilor este trimisă înapoi clientului sub forma unui răspuns **JSON**. Acest flux simplu, dar robust, asigură o livrare eficientă și fiabilă a datelor.

Pentru a îmbunătăți funcționalitatea, se pot implementa mai multe caracteristici suplimentare. Adăugarea suportului pentru paginare ar permite returnarea unui număr limitat de postări la fiecare cerere, reducând astfel încărcarea serverului și îmbunătățind performanța pentru baze de date mari. De asemenea, integrarea unor mecanisme de filtrare ar oferi utilizatorilor posibilitatea de a restrânge rezultatele pe baza unor criterii specifice, cum ar fi autorul, cuvinte-cheie sau intervalul de timp. Funcționalitatea de sortare ar putea fi un alt aspect valoros, permițând ordonarea postărilor după data publicării, titlu sau alte câmpuri relevante. În plus, validarea cererii ar putea asigura că orice parametri opționali (precum cei pentru paginare, sortare sau filtrare) sunt furnizați corect și respectă standardele așteptate.

Această rută oferă o bază solidă pentru dezvoltarea ulterioară a funcționalităților care să îmbunătățească experiența utilizatorilor și să optimizeze performanța aplicației. Prin extinderea și îmbunătățirea continuă a acesteia, se poate transforma într-un instrument esențial pentru interogarea și gestionarea eficientă a datelor despre postări.

O imagine care conține text, captură de ecran, Font, număr

Descriere generată automat

Figură 37 - Rută de obținere a postărilor

### Securizarea Parolei

##### Parole(adăugarea de „sare” și funcțiile hash)

Documentul utilizatorului are doi parametrii utilizați pentru a verifica corectitudinea parolei, așa cum am observat în timp ce demonstrăm cum sunt salvate datele în bază. mai precis, câmpurile *salt* și hash.

În domeniul criptografiei, termenul „*sare*” se referă la informații aleatorii care sunt adăugate la funcția unidirecțională utilizată pentru a calcula hash-ul unor date sau a unei parole.

Parolele de pe mediile de stocare pot fi securizate folosind „săruri”. Dintr-o perspectivă istorică, parolele au fost mult timp stocate la vedere, dar pe măsură ce timpul a trecut, au fost create mai multe măsuri de securitate pentru a preveni citirea de sistem a parolelor utilizatorilor. O astfel de abordare este adăugarea de „*sare*”.

Pentru fiecare parolă, o „*sare*” diferită este produsă la întâmplare. Cele doua componente sunt adesea combinate, folosind o metoda hash bazată pe criptografie, parola rezultată (!originală) mai apoi

este salvată adăugând „sare” într-o DB. Hashing-ul se ocupa cu logarea viitoare stocarea nefiind necesară și, în consecință, fără a rula pericolul ca stocarea autentificării să fie piratată și să expună parola la vedere.

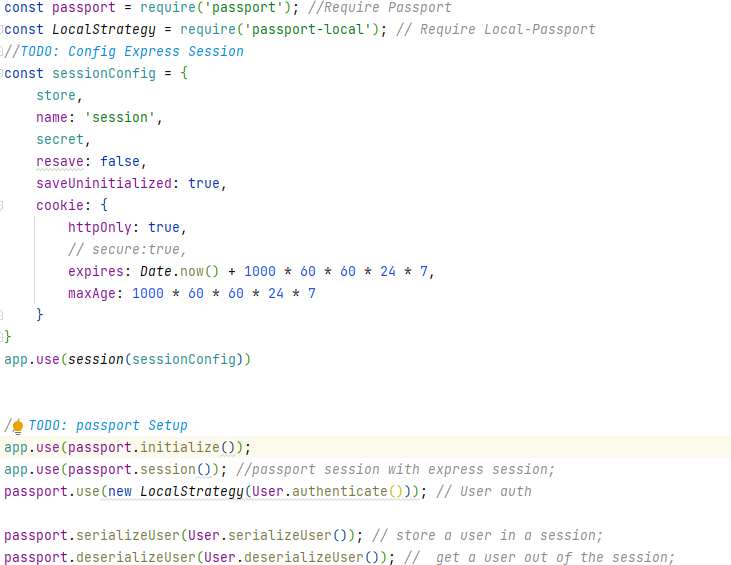
Un atac care utilizează un hash pre-generat este, de asemenea, prevenit folosind „sărurile”. Ele pot face ca dimensiunea parolei care a folosit metoda hash să fie excesiv de mare, fără a împovăra

utilizatorii, utilizatorii nu sunt nevoiți să rețină aceste valori de „sare”. De asemenea, asigură parolele

„standard”, sau cele care sunt folosite în mod regulat sau de către utilizatorii mai neexperimentați și cu (o singură parolă) pe diverse site-uri web, deoarece valorile „salt” sunt unice în fiecare situație.

De la autentificarea sistemului Unix la securitatea pe Internet, sărurile criptografice sunt adesea folosite într-o gamă largă de sisteme informatice.

Pentru aplicația noastră a fost folosit *API*-ul *Passport*. Acesta este responsabil de toate operațiunile de hash și adăugare de salt pe care le-am detaliat mai sus. În cele ce urmează va fi introdus un instant ce reprezintă modul de configurare al acestuia(*Figura 38*).



Figură 38 - Configurarea Passport API

### EJS Vizualizările(Componentele Template)

Este prescurtare pentru *Embedded JS Template* folosit in *Node*. Folosind cel mai minimal cod posibil, „*Template Engine*” ajută la crearea de șabloane *HTML*. Pe lângă producerea *HTML*-36 ului final, poate adaugă date în template *HTML* de pe partea de frontend. Folosind *JavaScript*, marcajul este produs folosind un limbaj de șabloane simplu ca si EJS.

Opusul este și adevărat, deoarece paginile *HTML* pot beneficia și de capacitatea de a încorpora

*JavaScript*. Următoarea imagine reprezintă un model de view care este aplicat la vizualizarea topicului:



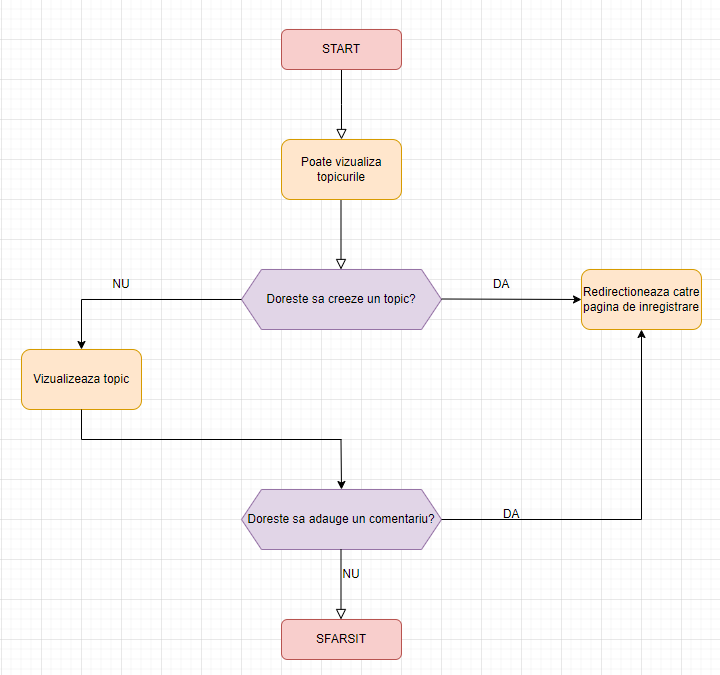
Figură 39 - Exemplu EJS view

## Utilizarea aplicației

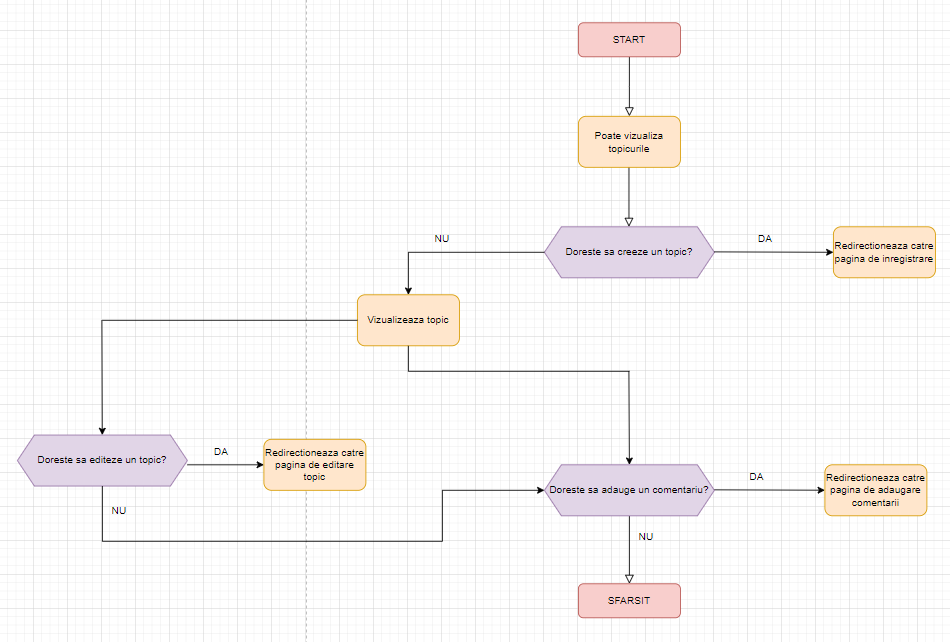
### Utilizare (prezentare)

Vizitatorii (utilizatorii care nu și-au făcut cont și nu s-au înregistrat) și utilizatorii compleți (cei

care sau înregistrat)pot accesa și utiliza programul. Aceste două grupuri de utilizatori folosesc programul în moduri ușor diferite, totuși. Următoarele vor oferi o demonstrație schematică a modului de utilizare în ambele scenarii:



Figură 40 - Diagrama pentru vizitatori



Figură 41 - Diagrama pentru utilizatori înregistrați

### Utilizare (Interfata)

Design-ul UI se focusează pe prezicerea a ce funcții vor avea userii și pe bazarea că interfața conține părți care sunt ușor de utilizat, de înțeles și de accesat.

Termenii de (design, arhitectură, si conexiuni) sunt toți combinați în interfață.

#### Utilizare (Alegerea Interfeței)

Userii se obișnuiesc cu comportamentele specifice ale componentelor UI, prin urmare trebuie să fim singuri în deciziile noastre.

Următoarele sunt doar câteva exemple de componente de interfață:

Input-uri: Buton, câmp de text, casetă de selectare, buton radio, buton radio, listă derulantă, câmp pentru dată etc.

Navigație: câmpul de căutare, glisorul, etichetele, pictogramele etc.

Componentele care oferă informații includ indicii, pictograme, bare de progres, alerte, casete de mesaje, ferestre modale etc. 48

Mai multe componente pot fi acceptabile pentru afișarea materialului uneori. Este esențial să luăm în considerare compromisurile atunci când se întâmplă acest lucru. Este posibil ca utilizatorii să fie nevoiți să muncească mai mult pentru a deduce ceea ce este într-un meniu derulant.

#### Practici bune pentru proiectarea interfețelor

Înțelegerea obiectivelor, capacităților, preferințelor și înclinațiilor utilizatorilor va fi necesară pentru proiectarea interfeței. La crearea interfeței de utilizare, vom ține cont de acești factori:

Interfața noastră este păstrată simplă. Majoritatea oamenilor pot vedea cu greu cele mai bune interfețe de utilizator.

Evitam utilizarea componentelor străine în timpul utilizării. Folosim componente standard de interfață și menținem coerența. Utilizatorii sunt mai în largul lor și pot finaliza sarcinile mai rapid atunci când interfața are aspecte familiare. Pentru a face interfața mai ușor de utilizat, este de

asemenea esențial să stabiliți modele (secvențe similare) în limbajul, stilul și designul site-ului. Odată ce un utilizator stăpânește o sarcină, el sau ea ar trebui să o poată aplica în alte zone ale site-ului web.

Folosim în mod deliberat textura și culoarea. Putem folosi totul in a avea un avantaj in a atrage atenția asupra sau a devia atenția de la anumite componente.

Pentru a oferi ierarhie și claritate, se folosește tipografia. Folosirea fontului este ceva pe care îl examinăm cu atenție. Putem îmbunătăți lizibilitatea și vizibilitatea prin diferite dimensiuni ale

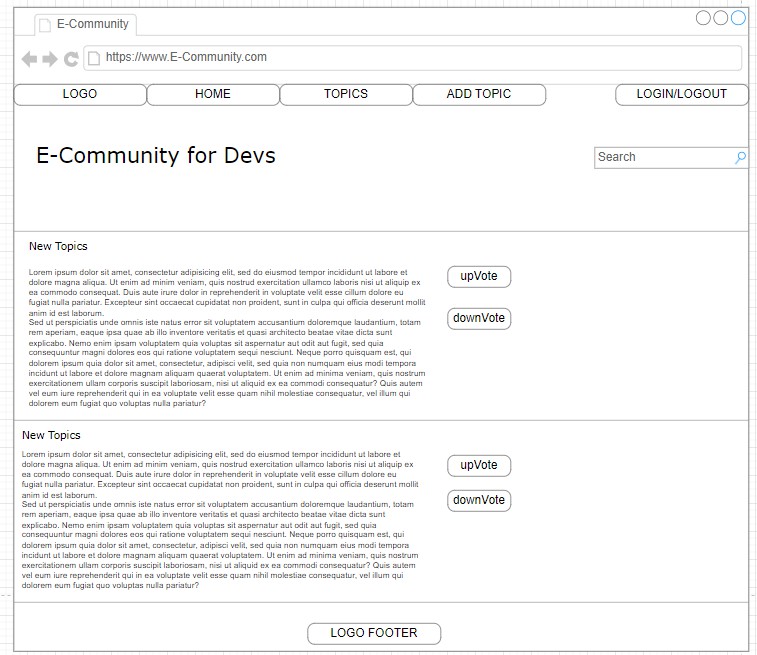
textului, fonturi și aranjamente. Ne asigurăm că sistemul transmite informații. Când există acțiuni, modificări de stare sau greșeli, îi anunțăm întotdeauna utilizatorilor noștri. Pentru a indica starea și, dacă este necesar, următoarele acțiuni, interfața ar trebui să utilizeze o varietate de caracteristici.

Acest lucru va reduce supărarea utilizatorului.

#### Descrierea interfeței de utilizare

Paginile principale au o interfață de utilizator similară. Nu putem oferi o explicație detaliată a acesteia, deoarece vorbim despre o interfață receptivă care se modifică în funcție de dimensiunea

ecranului fiecărui tip de dispozitiv. Folosind un laptop cu un ecran de 17 inch ca exemplu, vom prezenta schematic pagina de pornire după cum urmează:



Figură 42 - Template Pagină de pornire

# Capitol 5. Concluzii

Putem adăuga librăria Angular capabilă pentru dezvoltarea „frontend” ca o potențială opțiune viitoare.

O librărie pentru sit-uri dinamice, putem opta pentru AngularJS fiind structural. Permite să creăm părți ale programului nostru folosind sintaxa HTML, permițându-ne totodată să o extindem. În Angular, o mare parte din codul care ar trebui scris în mod obișnuit este eliminat datorită principiilor de binding data și injecție de dependență.

Stiva tehnologică MEAN ar fi utilizată ca urmare a acestei îmbunătățiri (MongoDB, Express.Js, AngularJs și Node.Js). În ceea ce privește arhitectura, proprietarii anumitor platforme semnificative au optat pentru suita MEAN. Aici îmi vin în minte companii precum LinkedIn, Netflix și Yahoo.

Am putea lua în considerare o variantă a utilizării limbajului Golang dezvoltat de Google pentru o eventuală actualizare a porțiunii de backend.

Deși anterior am folosit Uber ca exemplu de firmă care folosește suita MEAN, este important de reținut că o parte din microserviciile acestei organizații au fost mutate de la NodeJS la Golang. Uber

utilizează algoritmi de tipul acelor microservicii punctate în poligon ( PIP - Point In Polygon), care consumă mult din procesor, care este motorul cheie din spatele acestor ajustări. Ei au făcut conversia

necesară în Golang, deoarece funcționează pe multe thread-uri de execuție, spre deosebire de NodeJS, și pentru că au fost peste 200k de solicitări pe secundă.

Trecerea la Golang, însă, ar fi inutilă în acest moment pentru aplicația descrisă în articol,

deoarece solicitările serverului nu sunt suficient de complicate pentru ca noi să folosim capacitățile acestui limbaj de programare.

În cele din urmă, putem afirma că am structurat o aplicație de la cap la coada în JS, care ne-am întărit convingerile, făcându-l adecvat pentru implementarea proiectului în discuție

# 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

 Ronacher, Armin. *Flask Documentation*. https://flask.palletsprojects.com

 Grinberg, Miguel. *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, 2018.

 Brown, Anthony. *Python Web Frameworks*. Packt Publishing, 2020.

 Real Python. *Introduction to Flask*. https://realpython.com/flask-by-example-part-1-project-setup/

 Bayer, M. (2024). *SQLAlchemy Documentation*. https://docs.sqlalchemy.org

 Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O’Reilly Media.

 Beazley, D. M. (2020). *Python Essential Reference*. Addison-Wesley Professional.

 Copeland, R. (2013). *SQLAlchemy Essentials*. Packt Publishing.

 Curylo, J. (2022). *Mastering SQLAlchemy*. Apress.

 Real Python. *Using SQLAlchemy with Flask*. <https://realpython.com>

 Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.

 Bayer, M. (2024). *SQLAlchemy Documentation*. https://docs.sqlalchemy.org

 Flask-Login Documentation. (2024). <https://flask-login.readthedocs.io>

 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