UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA

FACULTATEA DE MATEMATICǍ ŞI INFORMATICǍ

SPECIALIZAREA INFORMATICĂ ROMÂNĂ

LUCRARE DE LICENŢĂ/ DISERTAŢIE/ABSOLVIRE

DEZVOLTAREA APLICATIILOR WEB FOLOSIND REST

Conducător ştiinţific

Dr. Grigoreta Sofia Cojocar

Absolvent

Vlasie Daniel - Sandrino

**2021**

# Contents

[Contents 2](#_Toc69403106)

[1 Introducere 4](#_Toc69403107)

[2 Aplicatiile web 5](#_Toc69403108)

[2.1 Ce este o aplicație web? 5](#_Toc69403109)

[2.2 Istoria aplicațiilor web 5](#_Toc69403110)

[2.3 Structura unui browser web 6](#_Toc69403111)

[2.4 Concepte de bază 7](#_Toc69403112)

[2.4.1 Arhitectura client-server 7](#_Toc69403113)

[2.4.2 HTML 8](#_Toc69403114)

[2.4.3 Cookies 9](#_Toc69403115)

[2.4.4 Securitate 10](#_Toc69403116)

[3 REST 12](#_Toc69403117)

[3.1 Ce este REST mai exact ? 12](#_Toc69403118)

[3.2 Client Server 13](#_Toc69403119)

[3.3 Lipsa stării 13](#_Toc69403120)

[3.4 Cache 13](#_Toc69403121)

[3.5 Interfața Uniformă 13](#_Toc69403122)

[3.5.1 Identificarea Resurselor 14](#_Toc69403123)

[3.5.2 Manipularea Resurselor prin Reprezentări 14](#_Toc69403124)

[3.5.3 Mesajele autodescriptive 14](#_Toc69403125)

[3.5.4 Hipermedia ca motor al stării aplicației (HATEOAS) 15](#_Toc69403126)

[3.6 Sistem Stratificat 15](#_Toc69403127)

[3.7 Cod la cerere 15](#_Toc69403128)

[4 Tehnologii Web 16](#_Toc69403129)

[4.1 Angular 16](#_Toc69403130)

[4.2 React 16](#_Toc69403131)

[4.3 Vue.js 16](#_Toc69403132)

[5 Despre Aplicație 18](#_Toc69403133)

[5.1 Cerinte 18](#_Toc69403134)

[5.2 Proiectare 18](#_Toc69403135)

[5.2.1 Diagrame de clase 18](#_Toc69403136)

[5.2.2 Diagrame de secventa 18](#_Toc69403137)

[5.2.3 Structura baza de date 18](#_Toc69403138)

[5.3 Implementare 18](#_Toc69403139)

[5.4 Manual de utilizare 18](#_Toc69403140)

[6 Concluzii 19](#_Toc69403141)

[7 Referințe 20](#_Toc69403142)

# Introducere

Aplicația pe care urmează să o prezint este o aplicație cu tematică gastronomică, care însumează o serie de atribute facil de uzitat. O primă condiție pentru a utiliza aplicația web creată de mine este autentificarea, fără de care nu este permis accesul. Autentificarea se poate realiza fie accesând contul deja existent, fie creând unul nou prin accessarea butonului „signup”, care va redirecționa utilizatorul spre pagina de creare a contului. În urma acestei acțiuni va fi afișată pagina principală unde găsim rețete culinare împărtășite de alți utilizatori. În mod prestabilit, aceste rețete se află sub următorul format: nume acompaniat de o ilustrație vizuală. Printr-un simplu click pe butonul aflat sub rețetă se accesează detaliile necesare pentru realizarea acesteia, precum: ingredientele, cantitățile și pașii de urmat.

În partea superioară a paginii, este situată bara de navigare, unde se regăsesc ,atât butonul pentru accesarea paginii principale, cât și numele de utilizator, alături de poza de profil și butonul de logout. Odată cu crearea contului personal, se obține posibilitatea de a împărtăși o rețetă proprie. Sunt necesari următorii câțiva pași: acționarea butonului din partea inferioară a paginii principale care permite completarea numelui rețetei, adăugarea imaginii descriptive, specificarea ingredientelor, a cantităților acestora și pașii de preparare. De asemenea, postările pot fi modificate sau șterse doar de către creator.

Această aplicație web îți permite conturarea unui profil culinar care conține postările personale, postările apreciate și rețetele marcate ca “gătite”. Profilul se poate personaliza prin adăugarea unei fotografii ce poate fi schimbată în mod frecvent printr-un simplu click al butonului specific. Aplicația contorizează atât postările proprii, cât și postările apreciate de utilizator și cele marcate ca fiind “gătite”. Posibilitatea de a aprecia sau de a marca “gătită” o rețetă se datorează existenței celor două butoane, primul sub forma unei inimi, respectiv sub forma unor tacâmuri, care se colorează prin intermediul unui click.

# Aplicatiile web

## Ce este o aplicație web?

O aplicație web este o aplicație software executată de un server web care răspunde unor cereri făcute de anumite browsere web, realizate prin intermediul protocolului HTTP. O aplicație web este compusă dintr-o colecție de mai multe scripturi, aflate pe un server web, ce interacționează cu baze de date sau cu alte surse de conținut dinamic. Cu ajutorul Internetului, aplicațiile web permit furnizorilor de servicii și clienților să partajeze și să manipuleze informații într-o manieră independenta de platforma pe care rulează, ceea ce înseamnă că aplicația poate fi mutată de pe o platforma pe alta cu modificări minime sau chiar deloc.

Aplicațiile web folosesc o arhitectură distribuită pe mai multe niveluri. În principal există un client, reprezentat de browserul web, un server web, unul sau mai multe servere ale aplicației și o bază de date [1]. Aplicațiile web sunt mult mai complexe decât par și totodată și decât alte tipuri de aplicații software, datorită serverelor care pot rula de la distanță și a clientului, reprezentat de browserul web, ce poate fi accesat de pe diverse platforme, fără a necesită o preinstalare. Complexitatea acestora este dată de amestecul de limbaje de programare, precum PHP și JavaScript, dar și de limbaje de formatare și conținut, cum ar fi HTML și CSS, pe care aplicațiile îl pot cuprinde.

## Istoria aplicațiilor web

Pentru o mai bună înțelegere a ceea ce reprezintă aplicațiile web contemporane, să analizăm evoluția aplicațiilor web în ultimele decenii. Dacă revenim la începutul anilor 1990, se poate observa existența conceptului de aplicații web, mult diferite de ceea ce întâlnim în zilele noastre,  și tot ceea ce ne putea oferi Internetul, referitor la acest subiect, erau pagini HTML statice, formate din simple documente text.

Această arhitectură purta numele de Web 1.0, fiind formată din două componente, server și client. Prima parte, serverul, avea ca și atribuții doar transmiterea de date text către client, cea de-a doua parte a acestei arhitecturi.  Mai târziu a devenit posibilă adăugarea de imagini, fișiere video și audio, însă paginile web au rămas statice, neavând posibilitatea de a interacționa în vreun fel cu acestea. În acest punct al dezvoltării aplicațiilor web existau în jur de 100,000 de site-uri și 50 milioane de utilizatori [15]. Dorința de a dinamiza paginile HTML a dus la dezvoltarea limbajului de programare JavaScript, pe partea de client, în 1995. Cu ajutorul acestei noi tehnologii s-a putut realiza execuția codului direct în browser, prezentarea diverselor elemente interactive, incluzând animațiile vectoriale, oferind astfel utilizatorilor posibilitatea de a interacționa cu noile pagini web create. În 1996 a fost introdus Macromedia Flash, un program care creează conținut media interactiv și animat precum grafică video, îmbunătățind considerabil animațiile. Acesta a fost momentul care a dus la dezvoltarea jocurilor video.

Anul 1999 a adus la apariția conceptului de aplicație web în limbajul Java. Tot în această perioada s-a făcut și trecerea de la Web 1.0 la Web 2.0, ajungându-se la peste 100 milioane de site-uri web și aproximativ un miliard de utilizatori. Arhitectura Web 2.0 a devenit mult mai complexă decât cea de dinaintea ei, având mai multe baze de date și servere, iar pe partea de client, majoritatea site-urilor au început să permită utilizatorilor interacțiunea cu acestea, ceea ce a dus la apariția site-urilor sociale și la posibilitatea creării de bloguri personale. În 2005 a fost introdusă Ajax, o nouă abordare a dezvoltării de aplicații web și aplicații web asincrone, ce au permis utilizarea conceptului de “responsive web design”(RWD), ceea ce înseamnă că paginilie web pot fi randate bine pe diferite dispozitive și pe o varietate mare de dimensiuni de ferestre. Datorită noii tehnologii Ajax, a  fost posibilă trecerea de la paginile web statice la cele dinamice.

Evoluția tehnologiei a determinat ca, peste un deceniu, în jurul anului 2013 să își facă apariția Web 3.0, cu un miliard de site-uri și 2,5 miliarde de utilizatori, iar mai apoi, în 2014, să fie creată nouă tehnologie HTML5, o îmbunătățire a celei deja existente, respectiv HTML. HTML5 a permis introducerea unor noi tipuri de multimedia și crearea de aplicații web independente de browsere și platforme. La scurt timp după apariția HTML5, în 2015, și-au făcut apariția aplicațiile web progresive (PWA), combinând caracteristicile majorității browserelor moderne cu cele ale aplicațiilor mobile, unul dintre beneficii fiind diferența aproape inobservabilă dintre o aplicație mobilă și una de tip PWA [2].

## Structura unui browser web

În zilele noastre aplicațiile web au ajuns să le depășească pe cele de tip desktop, fiind utilizate din ce în ce mai mult, datorită similitudinii funcționalităților oferite și, mai mult decât atât, a lejerității cu care pot fi utilizate, doar prin simpla accesare a unui browser web.  Printre componentele esențiale ale unui browser web modern se numără:

1. DOM (Document Object Model) este un mod, independent de platforma pe care rulează, de a reprezenta o multitudine de obiecte ce alcătuiesc o pagină dintr-un browser web. Acesta permite paginii să fie analizată și modificată de către JavaScript, limbajul de programare prezent pe partea de frontend a unei aplicații web. Practic, DOM este o interfață între limbajul de programare și browser, ce le permite acestora două să comunice între ele cu ușurință.

2. HTML este limbajul de marcaj predominant folosit la crearea paginilor web. Reprezintă un mijloc de a descrie structura informațiilor bazate pe text, marcându-le ca fiind paragrafe, liste, titluri sau alte tipuri de structuri. Totodată ajută și la îmbinarea textului cu diverse imagini, formulare interactive sau altele.

3. CSS este un limbaj de stilizare, folosit pentru a descrie aspectul documentelor scrise într-un limbaj de marcaj precum HTML. Un limbaj de acest gen permite ca diferite proprietăți ale documentului stilizat, precum culoarea, fontul sau efectele, să fie stabilite și definite indepentent de conținutul paginii web.

4. Pe lângă HTML și CSS, documentul necesar unui browser web poate conține și cod executabil, cel mai frecvent utilizat limbaj în acest scop din zilele noastre fiind JavaScript. Din punct de vedere sintactic, JavaScript este foarte asemănător cu limbajele Java și C. Este totuși mult mai dinamic decât acestea, poate și datorită particularităților împrumutate de la alte limbaje precum Self, Smalltalk sau Lisp [3].

## Concepte de bază

### Arhitectura client-server

În zilele noastre, avansarea tehnologiei a făcut ca folosirea internetului să fie aproape inevitabilă în tot ceea ce facem. Utilizarea internetului este, de fapt, utilizarea unei arhitecturi de tipul client-server. Clientul este cel care cere, serverul fiind cel care oferă serviciul cerut, în majoritatea aplicațiilor de acest tip, serverul procesează datele, care mai apoi sunt trimise către client [9].

Modelul client-server a devenit foarte popular în rândul aplicațiilor, fiind folosit din ce în ce mai des. Acesta poate fi definit ca o arhitectură formată atât din clienți, cât și din servere, ce împarte procesarea în cel puțin două procese, cel mai adesea pe cel puțin două mașini. Unul dintre protocoalele standardizate folosite este HTTP, cel care urmează a fi prezentat in secțiunea **2.4.2**. Orice aplicație care gestionează diverse informații stocate într-o baza de date, pe o anumită mașină, și le afișează pe o alta este o aplicație ce se folosește de modelul client-server. Ideea de bază din spatele acestui model a fost posibilitatea mai multor utilizatori de a avea acces la aceleași date.

De obicei, o arhitectură de acest tip este formată dintr-un client (de ex un browser web), un server și o bază de date, cele mai mari două categorii fiind:

* Arhitectura pe 2 nivele, formată doar din baza de date și client. În acest caz, clientul este cel care deține întreaga logică, trimițând cereri către baza de date prin intermediul unei rețele și tot el afișând răspunsurile utilizatorului. Acest client mai este numit și “client gros” (eng. thick client). Arhitectura pe 2 nivele se folosea mai ales la începuturile acestei noi structuri și era perfectă pentru pagini web statice sau cu formulare ale căror date nu necesitau procesare înainte de salvarea lor în baza de date.
* Arhitectura pe 3 nivele, compusă din client, bază de date și un server, ca parte intermediară între celelalte două. Poate fi extinsă cu ușurință la o arhitectură pe n nivele, prin adăugarea mai multor servere. În aplicațiile ce beneficiază de această structură, clientul are rolul doar de a prezenta utilizatorului informațiile din baza de date, furnizate de către nivelul intermediar (serverul), supranumit și “middleware”. Middleware este o denumire oferită nivelului interpus între client și baza de date, reprezentând un software, care rulează de obicei pe alt computer, ce are ca scop furnizarea logicii aplicației [10].

### HTML

Hypertext Transfer Protocol (HTTP) a fost creat pe la mijlocul anilor 1990 ca modalitate de comunicare între servere și browsere web. De atunci este protocolul din spatele fiecărei tranzacții web. Fiecare cerere pentru un document, trimitere a unui formular sau click al unui link, are la bază acest protocol de tip cerere/răspuns. Utilitatea oferită de acesta este capacitatea de a transfera diferite tipuri de date: text, grafică, audio și video, specificând modul în care cererile sunt făcute de către client, dar și modul în care serverul răspunde unor astfel de cereri [4].

Protocolul HTTP se folosește de Universal Resource Identifier (URI) ca nume (URN) sau ca locație (URL), pentru identificarea unei resurse. La crearea unui hiperlink, URL e de foma “http://host:port/ruta/numele-fișierului.extensie”. Portul folosit de obicei este 80, însă și alte porturi se pot folosi fără probleme.  Pentru ca un server să poată comunica cu un client, între aceștia doi se realizează o conexiune de tip TCP/IP. Clientul face o cerere corespunzătoare către server, în format URI, adăugând versiunea protocolului și un mesaj format din modificatorii necesari cererii, informațiile despre client și eventualul conținut. Ca răspuns din partea serverului, clientul primește linia de stare a răspunsului, formată din versiunea protocolului și codul corespunzător (succes sau eroare), urmată de un mesaj cu informații despre server, metainformații ale entității transmise și conținutul aferent. Atât cererea, cât și răspunsul includ câte un antet de tip MIME (Multipart Internet Mail Extension), ce conține informații despre datele transmise, cum ar fi tipul datelor sau alte informații specifice. Începând cu versiunea HTTP/1.1, se realizează o conexiune persistentă între client și server, ceea ce înseamnă că nu se creează o nouă conexiune la fiecare cerere [5].

Codurile incluse în răspunsurile de tip HTTP sunt formate din trei cifre și sunt împărțite în cinci mari categorii. Prima cifră a fiecărui cod reprezintă clasa din care acesta face parte, astfel avem:

* 1xx – cod informativ, de exemplu cererea a fost primită
* 2xx – cod de succes, ceea ce înseamnă că cererea a fost primită, înțeleasă și acceptată
* 3xx - cod de redirecționare, sunt necesare acțiuni suplimentare pentru ca cererea să fie realizata
* 4xx – cod de eroare client, cum ar fi sintaxa gresită a cererii
* 5xx – cod de eroare server, cererea nu a putut fi îndeplinită de către server

Protocolul HTTP conține un set de metode ce pot fi folosite pentru a preciza scopul tranzacției. Cele mai folosite metode sunt:

GET – Această metodă este folosită pentru a cere o anumită informație (sub forma unei entități). Este recomandat să fie folosită doar în cazul în care cererea nu afectează starea datelor stocate.

POST – O metodă utilizată cel mai des (și recomandată) pentru a transmite date de la client către server, cu scopul de a fi prelucrate sau anexate, printr-o metodă de adăugare, la o bază de date. În cazul prelucrării datelor, codul returnat cel mai potrivit este 200 (ok) sau 204 (fără conținut),  în funcție de existența sau lipsa unei entități care să descrie rezultatul. Dacă pe server a fost creată o entitate nouă, codul potrivit ar fi 201.

PUT - Cu ajutorul acestei metode se transmit date către server, pentru a se realiza actualizarea unei resurse deja existente, caz în care codurile returnate ar trebui să fie 200 sau 204. Este posibil ca datele transferate să nu facă referință la o resursă deja existentă. Dacă serverul poate defini o nouă resursă cu datele primite, atunci codul returnat va fi 201 (creat).

DELETE – Această metodă cere serverului să șteargă resursă identificată de datele trimise. Succesul trebuie garantat de server doar după ce resursă a fost ștearsă sau mutată într-un loc inaccesibil, codurile corespunzătoare fiind 202 (acceptat), dacă acțiunea urmează să fie efectuată sau 200, respectiv 204, dacă acțiunea a fost efectuată cu succes și răspunsul include sau nu o entitate [6].

Un exemplu de cerere pentru metoda GET poate fi:

GET /website.html HTTP/1.1

Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36

Host: www.mywebsite.com

Accept-Language: en-us

Accept-Encoding: gzip, deflate

Connection: Keep-Alive

Si raspunsul:

HTTP/1.1 200 OK

Date: Mon, 05 Apr 2021 15:22:43 GMT

Server: Apache/2.2.14 (Win64)

Last-Modified: Wed, 24 Jan 2021 18:35:51 GMT

Content-Length: 1147

Content-Type: text/html

Connection: Closed

### Cookies

Cookie-urile au fost create pentru a păstra informații legate de interacțiunea cu un anumit mediu lipsit de aceste date (de ex. HTML). Această tehnologie a acaparat marea majoritate a site-urilor web, ținând evidență interacțiunii dintre acestea și utilizatorii lor. În mare parte, existența unor cookie-uri pe computerele clienților aduce beneficii, însă poate fi și un motiv de îngrijorare.

Cookie-urile sunt structuri mici de date, pe care un server le trimite către browserul pe care navighează clientul, fiind apoi salvate local pe computerul acestuia într-un fișier text. Nu sunt altceva decât informațîi legate de utilizator în contextul unei anumite aplicații web, cum ar fi parole, produse adăugate în coșul de cumpărături, preferințele sau identitatea acestuia. La prima accesare a unui site web care se folosește de această tehnologie, un nou cookie este creat și salvat, iar la următoarele accesări în aceleași condiții (același browser, același computer și aceeași pagină web), clientul este recunoscut și ii este afișat un conținut adaptat pentru el. Un cookie este un antet al protocolului HTTP care are nevoie de doi parametri obligatorii că să funcționeze, și anume denumirea lui și valoarea stocată. Pe lângă aceștia, mai pot fi adăugați alți parametri opționali precum data de expirare, domeniul, calea și dacă trebuie folosită o conexiune securizată sau nu [7].

Utilitatea cookie-urilor este dată de faptul că protocolul HTTP nu păstrează niciun fel de informație legată de cererile efectuate, lucru care îngreunează unele tranzacții. Folosindu-le, apar beneficii atât pe partea de client, cât și pe partea de server.  Aplicațiile web au putut dintotdeauna să păstreze informații legate de utilizatorii lor, salvându-le într-o baza de date, cu costul memoriei necesare. Apariția cookie-urilor a făcut posibilă salvarea acestor date pe partea de client, salvând serverul de acest cost. Un beneficiu adus clienților este completarea automată a formularelor cu datele salvate (nume de utilizator, parole). Un dezavantaj văzut de către utilizatori este salvarea datelor personale ale acestora, temându-se că ar putea fi urmăriți prin intermediul cookie-urilor. Totuși, nu este o amenințare la intimitatea clienților, datele salvate fiind doar cele introduse pe pagina web cu bună știință [8].

### Securitate

Aplicațiile web interacționează cu diverse medii din exterior, făcând presupuneri (de multe ori greșite), despre acestea. Supozițiile nevalidate corespunzător pot duce la scăderea securității unei aplicații. Principala problemă care poate duce la apariția unor vulnerabilități de securitate este validarea neconformă a datelor primite de la utilizatori.

Scopul pentru care au fost create aplicațiile web este de a permite unei game cât mai largi de utilizatori să le poată folosi doar cu ajutorul unui browser web și a unei conexiuni la internet. Programatorii le pot crea cu presupunerea că utilizatorii vor introduce doar date valide, așa cum au gândit ei aplicațiile, fără să țină cont de faptul că atât valoarea datelor de intrare cât și modul în care acestea sunt introduse, pot fi greșite. Același lucru se poate întâmpla și cu bazele de date unde s-a mers pe ideea că toate interogarile vor fi autorizate, fără a fi luată in calcul posibilitatea unor interogări făcute de către un utilizator rău intenționat. Toate aceste ipoteze trebuie verificate pentru a nu permite crearea de vulnerabilități ce vor putea fi valorificate de către utilizatorii rău intenționați. Deci, principala metodă de a evita breșele de securitate este identificarea și prevenirea din timp a acestora [11].

Principalele vulnerabilități de securitate ale aplicațiilor web sunt “command injection” și “Cross-Site Scripting” (XSS). Într-un atac de tipul “Command Injection”, atacatorul încearcă să acceseze informații la care nu are acces, introducând cod și schimbând firul de execuție al aplicației. Cea mai utilizată metodă de atac din această categoria este “SQL Injection”, care presupune introducerea de comenzi SQL ca și date de intrare pentru a modifică logica interogării inițiale. Un exemplu pertinent ar fi:

String interogare = “SELECT \* FROM Users WHERE username = ‘” + username + “’ AND password = ‘” + password + “’;”;

Cele două date de intrare “username” și “password” sunt preluate de la utilizator și fără o verificare corespunzătoare, acesta ar putea insera “orice’ OR 1=1;--” ca și valoare pentru parametrul “username”, valoarea corespunzătoare variabilei “password” fiind irelevantă. Interogarea rezultată va fi: “SELECT \* FROM Users WHERE username = ‘ceva’ OR 1=1;--‘ AND password = ‘orice’;”. “--“ este suportat de cele mai multe baze de date ca fiind operatorul ce marchează un comentariu pe o singură linie. Astfel, tot ce se află după acest operator va fi marcat drept comentariu, făcând interogarea să returneze toate datele din tabelul “Users”. Câteva soluții pentru împiedicarea acestui tip de atac sunt: refuzarea datelor de intrare ce conțin anumite șiruri de caractere, cosiderate suspecte de către programator, limitarea lungimii acestora sau folosirea de “prepared statements”.

“Cross-Site Scripting” este o metodă prin care atacatorii trimit coduri (sub forma unor scripturi) altor utilizatori prin intermediul aplicației. Browserele clienților la care a ajuns codul rău intenționat îl vor rula, neavând de unde să știe că nu este o sursă de încredere. La execuția acestor scripturi pot fi furate cookie-urile, sau chiar datele introduse de utilizatori în aplicație, fără ca aceștia să aibă vreo bănuiala. “Cross-Site Scripting” este o metodă foarte eficientă în atacurile de tip “phishing” [12]. Aceste atacuri presupun ca atacatorul, mascat ca o sursă de încredere, să trimită un email victimei, convingând-o să acceseze un anumit URL și astfel datele utilizatorului păcălit să fie redirecționate către atacator.

# REST

Internetul este o rețea de mare succes, creată inițial cu scopul de a conecta între ele laboratoarele de cercetare. La scurt timp de la apariție, numărul de site-uri web a început să crească fulgerător de repede, fapt care i-a îngrijorat pe dezvoltatorii acestei rețele, ea nefiind gândită pentru un număr așa mare de utilizatori. Unul dintre acești primi pionieri a fost Roy Fielding, cel care a și creat un set de șase reguli, denumite colectiv “Representational state transfer” sau “REST”. Menirea acestor constrângeri este de a oferi simplicitate, scalabilitate, fiabilitate și alte caracteristici dorite, oricărei arhitecturi ce le respectă, din cadrul unei rețele de dimensiunea Internetului.

## Ce este REST mai exact ?

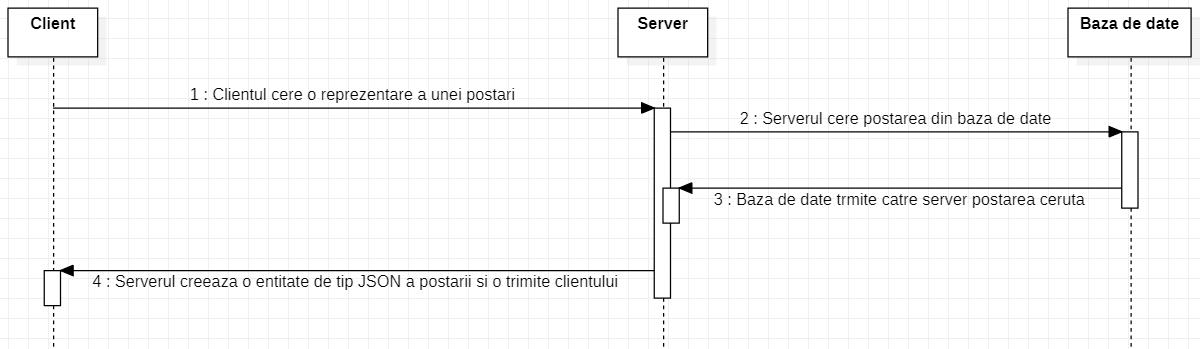
După cum am precizat și în paragraful anterior, REST este o prescurtare a termenului “Representational state transfer”, al cărui concept este unul destul de simplu. Ce vrea să zică mai exact, este că atunci când un server are o resursă dorită de client, acestuia din urmă nu îi pasă de reprezentarea internă a resursei oferită de către server, care o poate stoca sub orice formă (de ex. orice bază de date, fișier text sau chiar generată pe moment). Mai jos, în **Figura 1**, este un exemplu în care clientul cere o postare de la server, singurul lucru relevant fiind structura resursei trimisă de la server către client. Formatul JSON este unul des utilizat în cadrul REST, constând într-o simplă notație de tipul denumire/valoare. Un exemplu de reprezentare în format JSON a entității “postare” poate fi:

Figura 1 – Client / Server

{

“nume”: “Lasagna”,

“ingrediente”: “lapte-carne-brânză”

}

Deci, conceptul REST se referă doar la reprezentarea datelor transferate între client și server, iar orice API care îl folosește (respectă cele șase condiții explicate mai amănunțit în subsecțiunile **3.2 – 3.7**) poartă denumirea de “RESTful”.

## Client Server

Prima condiție și poate cea mai ușor de îndeplinit este ca arhitectura sistemului să fie de tipul client/server. Acest stil de arhitectură este atât de răspândit în zilele noastre, încât, în majoritatea cazurilor, nici nu mai sunt luate în considerare altele.

Serverele sunt cele care gestionează datele și logica aplicației, iar clienții sunt cei care se ocupă de experiența utilizatorilor și de modul în care aceștia vizualizează informațiile primite de pe partea de backend, fapt care duce la o delimitare clară între cele două părți. Separarea evidentă a acestora permite existența simultană a mai multor clienți și reduce semnificativ complexitatea serverului, îmbunătățind scalabilitatea aplicației, ceea ce permite componentelor să evolueze independent unele față de celelalte.

O chestiune importantă de precizat este faptul că utilizarea protocolului HTTP nu este impusă din niciun punct de vedere, chiar dacă mai mereu dezvoltatorii serviciilor web “RESTful” îl preferă, și oricum, nu prea se găsesc motive pertinente pentru a nu o face.

## Lipsa stării

O altă caracteristică ce simplifică și mai mult interacțiunea dintre client și server este lipsa stării. Asta înseamnă că orice informație referitoare la sesiunile clienților este stocată de către aceștia, ceea ce aduce un plus de scalabilitate, serverul nemaifiind nevoit să gestioneze nicio resursă legată de sesiuni.

O consecință a acestei condiții o reprezintă transmiterea de fiecare dată a tuturor informațiilor necesare unei cereri, datorită lipsei oricărui fel de context. Roy Fielding chiar a și scris la un moment dat că un client trece mai ușor peste niște erori atunci când nu este necesară sincronizarea contextului sesiunii între cele două componente, aducând fiabilitate aplicației. Principala cauza a scalabilitatii scăzute și a insuccesului aplicațiilor web este dată de dependență sesiunilor prezente pe server.

## Cache

Cea de-a treia condiție adusă de conceptul REST este prezența memoriei cache și presupune ca toate datele din răspunsurile aduse de pe server să fie etichetate ca fiind sau nu de tip cache. Orice informație de acest fel poate fi refolosită la cererile următoare, reducând traficul de date, nemaifiind nevoie să se facă o altă cerere către server, și îmbunătățind performanță aplicației. O mică problema a acestei constrângeri este că poate scădea fiabilitatea dacă datele stocate în cache diferă semnificativ de cele care ar fi fost returnate dacă s-ar fi făcut o nouă cerere către server.

## Interfața Uniformă

Caracteristica principală prin care o arhitectură REST se diferențiază de alte stiluri este utilizarea unei interfețe uniforme. Prin aplicarea acestei constrângeri, se pot accesa diferite aplicații web și se pot realiza acțiuni atât de diferite între ele, ceea ce este destul de impresionant. Toate acestea sunt posibile datorită lipsei dependențelor dintre interfață și implementare. Această condiție este formată din alte câteva subcondiții:

### Identificarea Resurselor

Spre deosebire de alte stiluri arhitecturale axate pe metode sau proceduri, REST este axat pe resurse. O resursă este orice poate fi reprezentat ca o entitate, de la utilizatori, postări, la conturi, comenzi s.a.m.d. De multe ori tabelele din bazele de date reprezintă un tip de resursă. Într-un cadru REST orice resursă trebuie să poată fi identificată prin propriul URI, chiar dacă aceasta este modificată. De obicei, cu ajutorul unui URI se accesează o colecție, iar un exemplu potrivit pentru o colecție de postări ar putea fi:

http:// localhost:4200/posts

Pentru a accesa o entitate specifică dintr-o colecție se poate folosi un URI ca cel de mai jos,

http:// localhost:4200/posts/{id}

unde “{id}” reprezintă identificatorul unic al entității.

### Manipularea Resurselor prin Reprezentări

Cea de-a doua condiție pentru o interfață uniformă este manipularea resurselor prin reprezentări. Ce înseamnă mai exact acest lucru este că pe partea de client nu se va lucra direct cu resursa originală, ci cu o reprezentare a acesteia într-un format neutru. Cel mai folosit format pentru un API de tip REST este JSON, fapt descris și în secțiunea **3.1**. Astfel, se evită dependențele dintre client și server, păstrându-se o oarecare independență a acestora.

### Mesajele autodescriptive

O altă constrângere este ca fiecare răspuns sau cerere să conțină suficientă informație, astfel încât să poată fi înțeles conținutul fără existența unui context. Acestea trebuie să includă neapărat formatul textului (application/json sau application/xml) pentru ca mesajul să fie analizat in mod corect. De asemenea, trebuie respectată semnificația metodelor HTTP, și anume:

GET – se folosește pentru a obține o colecție de entități (/likes) sau o anumită entitate (/likes/{id})

POST – se folosește pentru a crea o nouă entitate (/likes)

PUT – folosit pentru a modifica o anumită entitate (/likes/{id})

DELETE – se folosește pentru a șterge o anumită entitate (/likes/{id})

### Hipermedia ca motor al stării aplicației (HATEOAS)

Hipermedia se referă la texte ce conțin hiperlinkuri. O pagină web reprezintă, de fapt, o stare a unei aplicații web. Așadar, a patra subcondiție impune ca, pentru a naviga de pe o pagină web pe alta, să fie folosite link-uri. Acest lucru ajută clientul să poată înțelege cu ușurință cum se realizează deplasarea printre paginile aplicației.

## Sistem Stratificat

Altă condiție pentru îndeplinirea conceptului REST exprimă faptul că, în calitate de client, acesta nu ar trebui să cunoască straturile existente pe partea de server. Mai exact, aplicația trebuie să aibă o arhitectură stratificată, permițând astfel introducerea unui proxy, sau a unui strat de securitate între client și server, fără a fi necesară modificarea vreuneia dintre cele două componente. Totodată, acest lucru permite serverului să apeleze alte servere, fără nicio problemă, pentru a oferi un răspuns clientului.

## Cod la cerere

Ultima constrângere, care este și opțională, afirmă că, la cererea clientului, acesta poate primi cod din partea serverului, ce va fi executat la rulare. Acest procedeu este foarte des întâlnit la aplicațiile web, unde browserul descarcă cod în limbaj JavaScript, pentru a extinde funcționalitățile paginilor web la rulare [13], [14].

# Tehnologii Web

Poate cel mai folosit limbaj de astăzi în programarea web este JavaScript. Datorită utilizării în masă a acestui limbaj de programare, au fost dezvoltate mai multe framework-uri, care să ușureze munca programatorilor. Pentru aceștia, este important ca framework-ul ales să fie unul ce le îndeplinește necesitățile și le oferă un cod calitativ și de înaltă performanță [16]. Cele trei mari framework-uri ce vor fi prezentate sunt: Angular, React și Vue.js.

## Angular

Angular a fost creat în 2008 de doi angajați Google sub denumirea de AngularJS. În vremea aceea, majoritatea site-urilor web se bazau pe abordarea aplicației pe mai multe pagini, unde documentul HTML era preluat de pe server. AngularJS a fost unul dintre primele framework-uri care au introdus logică pe partea de browser și se bazau pe abordarea aplicațiilor web cu o singură pagină (SPA, eng. Single Page Application).

În 2014 a fost lansat Angular 2 (mai târziu denumit simplu Angular), o rescriere a tot ceea ce însemna AngularJS. Această a două versiune se axează mai mult pe componente, față de cea inițială care se baza pe scopuri și controlere. O îmbunătățire adusă a fost introducerea limbajului TypeScript, un succesor al limbajului JavaScript. Câteva beneficii ale acestui nou limbaj sunt buclele, funcțiile lambda și decoratorii. Elementele principale, componentele, sunt împărțite în trei mari categorii: documente HTML pentru șablonul paginilor, documente CSS pentru aspectul paginilor și fișiere TypeScript pentru logica aplicației, ceea ce a dus la o structura mult mai organizată a aplicației.

## React

React este o librărie dezvoltată de Facebook pentru a crea interfețe grafice. A fost publicată în 2013, având ca limbaj principal JavaScript. React este bazat pe componente și conținutul acestora, principala premiză fiind de a transforma o anumită stare a aplicației într-o vizualizare din browser. Deoarece React se ocupă doar de partea grafică, sunt necesare și alte tehnologii în dezvoltarea unei aplicații, și anume: un compilator pentru JSX (permite îmbinarea codului JS cu HTML), rutare și administrarea stărilor.

Componentele se pot scrie în două moduri, ca funcții, unde componenta este reprezentată printr-o funcție ce returnează un singur element react sau sub formă de clase. În cea de-a doua metodă, clasa creată trebuie să extindă superclasa “React.Component”. Aici trebuie ca logica vizualizării să fie încorporată în funcția “render()”, ce are ca restricție returnarea unui singur element. În general, se recomandă transpunerea componentelor în funcții pentru o eficientizare prin reutilizarea acestora.

## Vue.js

După cum am precizat și despre React, Vue.js se ocupă doar de partea grafică a unei aplicații. A fost creat în 2014 de un angajat de la Google care a lucrat mult cu AngularJS. Acesta a demisionat pentru a se putea dedica în totalitate dezvoltării noului limbaj, fiind sprijinit financiar de către comunitate.

Vue este un limbaj bazat pe componente, situate, de obicei, într-un singur fișier cu extensia “.vue”, fiind totuși posibilă și separarea acestora. Un principal beneficiu deținut de Vue este scalabilitatea, dar și integrarea cu ușurință este un avantaj de luat în seamă [17].

# Despre Aplicație

## Cerinte

## Proiectare

### Diagrama cazurilor de utilizare

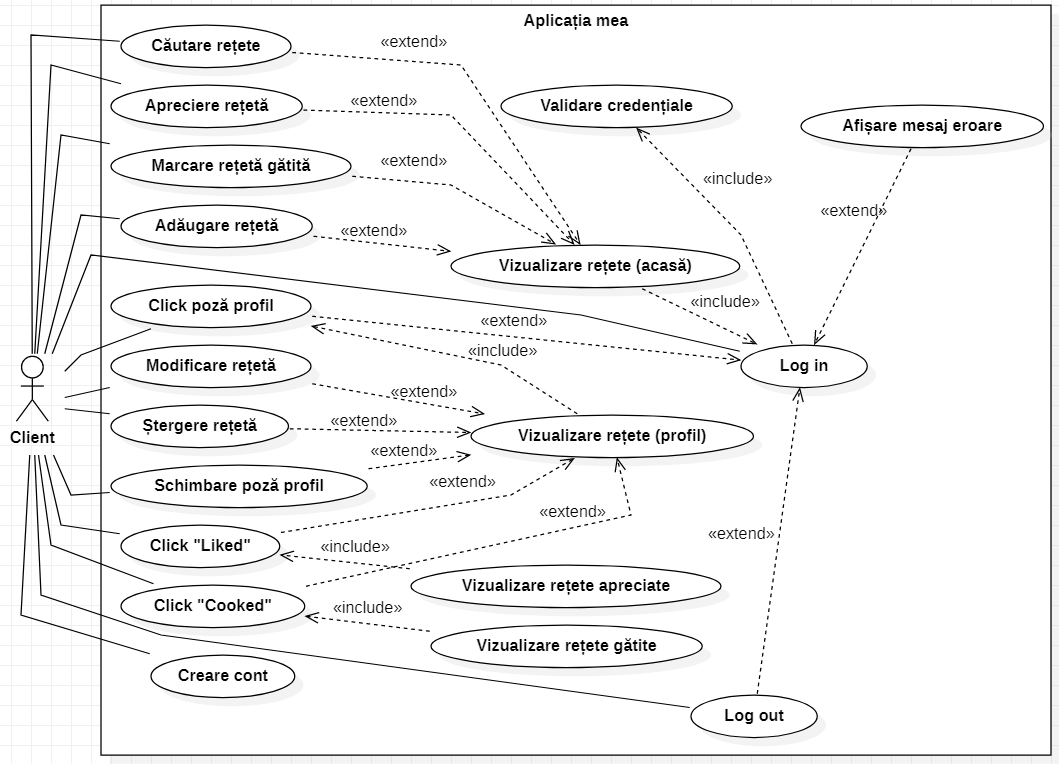


Figura 2 – Diagrama cazurilor de utilizare

1 Descriere login

|  |  |
| --- | --- |
| Nume | Login |
| Actori | Inițiat de client |
| Flux de evenimente | 1. Clientul introduce numele de utilizator și parola 2. Sistemul validează datele    1. Dacă datele sunt corecte sunt afișate rețetele de pe pagina principală    2. În caz de eroare este afișat un mesaj corespunzător |
| Precondiții | Sistemul este disponibil |
| Postcondiții | Clientul este autentificat |
| Cerințe de calitate | Sistemul va răspunde în maxim 10 secunde |

2 Descriere căutare rețete

|  |  |
| --- | --- |
| Nume | Căutare rețete |
| Actori | Inițiat de client |
| Flux de evenimente | 1. Clientul introduce un text în câmpul din stânga sus 2. Sistemul filtreaza rețetele și le afișează pe cele corespunzătoare |
| Precondiții | Clientul trebuie să fie autentificat |
| Postcondiții | Vor fi afișate doar rețetele corespunzătoare filtrării |
| Cerințe de calitate | Sistemul va răspunde în maxim 10 secunde |

3 Descriere apreciere rețetă

|  |  |
| --- | --- |
| Nume | Apreciere rețetă |
| Actori | Inițiat de client |
| Flux de evenimente | 1. Clientul dă click pe inimioara din colțul stânga jos al postării 2. Sistemul marchează rețeta ca fiind apreciată prin colorarea inimioarei și salvează în baza de date acest lucru |
| Precondiții | 1. Clientul trebuie să fie autentificat 2. Trebuie să existe postări |
| Postcondiții | Postarea este marcată ca fiind apreciată de către client |
| Cerințe de calitate | Sistemul va răspunde în maxim 10 secunde |

4 Descriere marcare rețetă gătită

|  |  |
| --- | --- |
| Nume | Marcare rețetă gătită |
| Actori | Inițiat de client |
| Flux de evenimente | 1. Clientul dă click pe tacâmurile din colțul stânga jos al postării 2. Sistemul marchează rețeta ca fiind gătită prin colorarea tacâmurilor și salvează în baza de date acest lucru |
| Precondiții | 1. Clientul trebuie să fie autentificat 2. Trebuie să existe postări |
| Postcondiții | Postarea este marcată ca fiind gătită de către client |
| Cerințe de calitate | Sistemul va răspunde în maxim 10 secunde |

5 Descriere adăugare rețetă

|  |  |
| --- | --- |
| Nume | Adăugare rețetă |
| Actori | Inițiat de client |
| Flux de evenimente | 1. Clientul dă click pe plusul din josul paginii 2. Sistemul redirecționează clientul pe o nouă pagină cu un formular (numele rețetei, ingrediente, pași, poză) 3. Clientul trebuie să completeze formularul    1. Dacă sunt completate toate câmpurile clientul poate trimite formularul    2. În caz contrar butonul de trimitere al formularului este dezactivat 4. După apăsarea butonului de trimitere al formularului, sistemul salvează rețeta in baza de date 5. Clientul este redirecționat pe pagina principală |
| Precondiții | Clientul trebuie să fie autentificat |
| Postcondiții | Rețeta este adăugată |
| Cerințe de calitate | Sistemul va răspunde în maxim 10 secunde |

6 Trebuie continuat

|  |  |
| --- | --- |
| Nume | Click poză profil |
| Actori | Inițiat de client |
| Flux de evenimente | 1. Clientul dă click pe poza de profil din bara de navigare |
| Precondiții | 1. Clientul trebuie să fie autentificat 2. Trebuie să existe postări |
| Postcondiții | Postarea este marcată ca fiind gătită de către client |
| Cerințe de calitate | Sistemul va răspunde în maxim 10 secunde |

### Diagrame de clase

### Diagrame de secventa

### Structura baza de date

## Implementare

## Manual de utilizare

# Concluzii

# Referințe

1. Fong, Elizabeth, and Vadim Okun. "Web application scanners: definitions and functions." 2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07). IEEE, 2007.
2. Oleg Uryutin, A brief history of web app, [https://aplextor.medium.com/a-brief-history-of web-app-50d188f30](https://aplextor.medium.com/a-brief-history-of%20web-app-50d188f30)d
3. Mikkonen, Tommi, and Antero Taivalsaari. "Web applications–spaghetti code for the 21st century." 2008 Sixth international conference on software engineering research, management and applications. IEEE, 2008.
4. Wong, Clinton. Http pocket reference: Hypertext transfer protocol. " O'Reilly Media, Inc.", 2000.
5. Yannakopoulos, John. "HyperText Transfer Protocol: A Short Course." University of Crete. August (2003).
6. Fielding, Roy, et al. "RFC2616: Hypertext Transfer Protocol--HTTP/1.1." (1999).
7. Peng, Weihong, and Jennifer Cisna. "HTTP cookies–a promising technology." Online Information Review (2000).
8. Kristol, David M. "HTTP Cookies: Standards, privacy, and politics." ACM Transactions on Internet Technology (TOIT) 1.2 (2001): 151-198.
9. Oluwatosin, Haroon Shakirat. "Client-server model." IOSRJ Comput. Eng 16.1 (2014): 2278-8727.
10. "Distributed Application Architecture". Sun Microsystem. Archived from the original on 6 April 2011. Retrieved 2009-06-16
11. Wassermann, Gary, and Zhendong Su. "An analysis framework for security in web applications." Proceedings of the FSE Workshop on Specification and Verification of component-Based Systems (SAVCBS 2004). 2004.
12. Nguyen-Tuong, Anh, et al. "Automatically hardening web applications using precise tainting." IFIP International Information Security Conference. Springer, Boston, MA, 2005.
13. Lange, Kenneth. "The Little Book on REST Services." Kopenhaagen 3 (2016): 5-6.
14. Roy Thomas Fielding: Representational State Transfer (REST), <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#fig_5_2>
15. Coursera: Evolution of Web Apps, <https://www.coursera.org/lecture/web-app/video-3-evolution-of-web-apps-yghKM>
16. Gizas, Andreas, Sotiris Christodoulou, and Theodore Papatheodorou. "Comparative evaluation of javascript frameworks." Proceedings of the 21st International Conference on World Wide Web. 2012.
17. Wohlgethan, Eric. SupportingWeb Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue. js. Diss. Hochschule für Angewandte Wissenschaften Hamburg, 2018.