**Prezentare generală a arhitecturii REST**

Cel mai comun tip de interacțiune pe web constă în recuperarea paginilor HTML prin cereri identificate cu URL-uri, astfel încât să putem accesa bucăți de informație. Atunci când introducem un URL în browser-ul nostru, este trimisă o cerere către un server care va decide ce fel de răspuns să trimită înapoi, iar în cea mai mare parte a timpului acesta va fi HTML pe care browser-ul nostru îl poate interpreta și afișa astfel încât să putem citi conținutul său.

Arhitectura REST este aplicată în dezvoltarea aplicațiilor web pentru a ajuta la accesarea și modificarea informațiilor legate de acestea. Arhitectura este definită de un set de componente și procese care sunt conturate de un set de constrângeri.

În următoarea secțiune, vom acoperi o prezentare generală a REST, câțiva termeni și concepte importante necesare pentru a-l înțelege.

***Ce este REST-ul?***

REST este un stil arhitectural, un set de reguli care pot fi aplicate în crearea serviciilor web. Scopul său este de a defini o modalitate uniformă de accesare și modificare a resurselor web. Informațiile pe care browser-ul le poate solicita sunt organizate în entități, fiecare dintre acestea returnând informații specifice care pot fi accesate prin URL-uri.

Solicitarea informațiilor printr-un URL, sau în alte cuvinte accesarea unei resurse, poate returna informațiile în mai multe formate, sau frecvent denumite reprezentări.

Odată cu fiecare cerere, informațiile includ toate datele necesare pentru a face cereri ulterioare către server fără a avea nevoie să cunoaștem întreaga imagine a ceea ce se întâmplă în aplicație (denumită și stare) pe partea de browser. De asemenea, ne indică ce alte resurse sunt disponibile și care pot fi accesate în continuare astfel încât să putem continua să folosim aplicația fără a recurge la documentație.

Aplicarea conceptului de entități și resurse ca modalitate de organizare, accesare și modificare a informațiilor s-a dovedit a fi o abordare utilă în dezvoltarea aplicațiilor.

Aceasta rezumă ceea ce este REST-ul la un nivel superior, dar arhitectura în sine este compusă din mai multe componente și constrângeri, fiecare cu un rol în definirea modului în care un serviciu proiectat cu REST ar trebui să funcționeze. Putem acum să mergem într-o ușoară detaliere asupra acestor constrângeri și să vedem cum se corelează cu conceptele introduse în această secțiune.

***Protocolul HTTP – facilitează schimbul de informații***

Pentru început, un protocol HTTP este protocolul World Wide Web-ului. Acesta definește regulile de comunicare între un client și un server. HTTP este fără stare și funcționează într-o abordare de cerere-răspuns.

Un client trimite o cerere legată de o anumită resursă. Aceasta poate fi un site web HTML, un fișier sau cod JavaScript. HTTP nu definește ce ar trebui să fie o resursă. Fiecare resursă are propriul său identificator numit URI. O structură generală a URI-ului arată în felul următor: URI-ul este doar o parte a cererii HTTP. Cererea constă din:

• linia de cerere cu calea, versiunea protocolului și metoda HTTP

• zero sau mai multe antete

• o linie goală care indică sfârșitul antetelor

• un corp(body) opțional

Serverul apoi își dă seama ce tip de cerere este aceasta citind-o cu atenție. Un aspect important de avut în vedere este metoda HTTP folosită pentru a efectua cererea. Există patru metode de bază, cele mai des utilizate: GET, POST, PUT și DELETE.

***GET*** - este folosit pentru a citi resursa. Serverul returnează resursa pentru URI-ul dat. Metoda GET nu conține un corp. Acesta doar preia resursa și nu o modifică în niciun fel.

***POST*** - este folosit pentru a transfera date către server. Prin urmare, este de obicei asociat cu crearea unei resurse. Datele sunt trimise în corpul cererii. După ce resursa este creată, serverul ar trebui să răspundă cu URI-ul său.

***PUT*** - este similar cu POST-ul. Cu toate acestea, diferă semnificativ. Este folosit pentru a actualiza resursa existentă. Suprascrie întreaga resursă cu datele transferate.

***DELETE*** - este folosit pentru a șterge o resursă existentă.

Aceste metode ne permit nouă și serverului să înțeleagă ce acțiune trebuie să fie efectuată cu o resursă. O resursă este compusă dintr-o entitate definită ca un substantiv (de exemplu, postări) și poate fi accesată ca o colecție sau ca o instanță.

***Restricțiile REST***

După cum am menționat, REST-ul este un stil arhitectural software. Pe de altă parte, acesta nu este standardizat (ca și SOAP-ul). Acesta oferă multă elasticitate în ceea ce privește implementarea. De exemplu, nu există o singură și adecvată modalitate de a implementa paginarea. Prin urmare, REST-ul definește un set de constrângeri principale, generale, de urmat în timpul dezvoltării API-urilor RESTful:

**Interfața uniformă**

Este o constrângere cheie care diferențiază între un API REST și unul Non-REST. Sugerează că ar trebui să existe o modalitate uniformă de interacțiune cu un server dat, indiferent de dispozitiv sau tip de aplicație (website, aplicație mobilă).

Există patru principii de bază ale Interfeței Uniforme:

* Bazat pe Resurse: Resursele individuale sunt identificate în cereri. De exemplu: API/utilizatori.
* Manipularea Resurselor Prin Reprezentări: Clientul are o reprezentare a resursei și conține suficiente informații pentru a modifica sau șterge resursa pe server, cu condiția să aibă permisiunea de a face acest lucru. Exemplu: De obicei, utilizatorul primește un id de utilizator când solicită o listă de utilizatori și apoi folosește acel id pentru a șterge sau modifica acel utilizator în mod particular.
* Mesaje Autodescriptive: Fiecare mesaj include suficiente informații pentru a descrie modul de procesare a mesajului, astfel încât serverul să poată analiza ușor cererea.
* Hypermedia ca Motorul Stării Aplicației (HATEOAS): Este necesar să se includă linkuri pentru fiecare răspuns, astfel încât clientul să poată descoperi cu ușurință alte resurse.

**Stateless**

Un API RESTful ar trebui să fie fără stare. Acest lucru înseamnă că nu stochează nicio informație despre sesiunea utilizatorului. Prin urmare, fiecare cerere trebuie să furnizeze date complete pentru a o procesa. Astfel, se ajunge la o mai mare disponibilitate a API-ului.

**Cacheable**

Răspunsul serverului ar trebui să furnizeze informații despre dacă ar trebui sau nu să fie cache-abil și pentru cât timp. Caching-ul datelor care se actualizează rar îmbunătățește performanța și elimină interacțiunile redundante client-server.

**Client-Server**

Arhitectura REST se bazează pe o configurație client-server. Clientul utilizează URI-urile pentru a accesa resursele, fără a fi necesar să cunoască detaliile procesării cererii de către server. În același timp, serverul se ocupă de procesarea și furnizarea resurselor, fără a afecta experiența utilizatorului. Atât clientul, cât și serverul sunt independente unul față de celălalt și nu necesită cunoștințe despre responsabilitățile celuilalt. Această abordare facilitează utilizarea unui singur API într-o varietate de medii diferite, cum ar fi browserele web și aplicațiile mobile.

**Sistem stratificat (Layered system)**

Un API REST poate consta din mai multe straturi, de exemplu, business logic, prezentarea, accesul la date. Mai mult, straturile nu ar trebui să afecteze direct altele. În plus, clientul nu ar trebui să știe dacă este conectat direct la serverul final sau intermediar. Prin urmare, putem scala ușor sistemul sau putem furniza straturi suplimentare cum ar fi gateway-urile, proxy-urile, echilibratoarele de sarcină.

**Cod la cerere**

Aceasta este o restricție opțională. Serverul poate returna o parte din codul în sine în locul datelor în format JSON. Scopul este de a oferi operații specifice pe datele pe care clientul le poate folosi direct. Nu este o practică comună.

Aceasta rezumă cele șase constrângeri care definesc REST, pe scurt fiecare constrângere încurajează o separare a preocupărilor atunci când proiectăm un serviciu web.Top of Form

***Arhitectura REST are mai multe beneficii:***

* Este simplu de construit și adaptat.
* Are un consum redus de resurse.
* Instanțele de proces sunt create explicit.
* Clienții nu necesită informații de rutare.
* Clienții pot avea o interfață 'ascultător' generică pentru notificări.
* Resursele pot fi reprezentate într-o varietate largă de formate de date (JSON, XML, etc.).
* Reduce cuplarea client/server.
* Browser-ele pot interpreta reprezentările.

În concluzie, arhitectura REST oferă o abordare flexibilă și scalabilă pentru dezvoltarea serviciilor web. Prin urmarea unui set de constrângeri principale, precum interfața uniformă, fără stare și client-server, REST permite dezvoltatorilor să creeze API-uri RESTful care sunt ușor de întreținut și eficiente. Această abordare facilitează evoluția independentă a clientului și a serverului, permițând utilizarea unui singur API într-o varietate de medii și aplicații. Cu toate că REST nu este standardizat precum alte protocoale, cum ar fi SOAP, elasticitatea sa în implementare oferă o mare libertate în dezvoltarea soluțiilor adaptate nevoilor specifice ale fiecărui proiect.