Soluzioni esercizio “Backend.pdf” di Roberto Pillitteri
10/10/2021

- REST è uno stile architetturale per sistemi distribuiti. Si basa su HTTP e i suoi metodi: GET, POST, PUT, DELETE. Permette un’interazione stateless tra client e server. E ogni Api, che rappresenta una specifica risorsa o funzionalità è identificata mediante un URI ben definito. Ad esempio posso avere l’uri: htttp://www.miositoweb.it/libri che mi permettere di scaricare tutta una serie di informazioni su dei libri contenuti in un db e restituirli in formato json. Oppure /login alla quale si effettua una POST con i dati per il login(email,password) per dirne una. I parametri della GET sono contenuti nell’URI, solitamente con *?param1=value1&param2=value2* . DELETE solitamente utilizzata per eliminare un contenuto mentre PUT per aggiornarlo, le convenzioni sono queste. Poi in base allo stato della richiesta viene restituito anche uno status code che indica se l’operazione è andata a buon fine o perché è fallita, tra i più noti: 4xx errore client => 404 not found. 5xx errore lato server.
- SOAP (simple object access protocol): è un protocollo per lo scambio di messaggi tra componenti software. Solitamente utilizza Http come protocollo di rete. Si basa sul linguaggio XML ed un suo messaggio è composto da un header e un body, quest’ultimo contiene il vero e proprio contenuto informativo(payload).
- Un database relazionale raccoglie e organizza elementi che sono in relazione tra di loro. Sono presenti delle tabelle, con un certo numero di colonne che rappresentano i campi di quella determinata informazione che vogliamo rappresentare. Ad esempio una tabella libro contenente il nome, l’autore e una descrizione. Ogni tabella contiene una chiave primaria, che è un identificativo univoco, spesso per convenzione e prestazioni si utilizza un id numerico. E possono esservi chiavi esterne che legano le tabelle tra loro in queste relazioni. Ad esempio potrei avere una tabella prezzi al cui interno vi è un campo book_id che lega il prezzo all’id del libro. Si basano sull’algebra relazionale e sono ampiamente utilizzati nelle applicazioni. Per interrogarli, creare tabelle, aggiornarle, inserire valori ecc si utilizza il linguaggio SQL.
- Un database NoSql al contrario è un database non relazionale, quindi non normalizza i dati in tabelle come uno relazionale. Al contrario questo tipo di database è più flessibile nel suo schema e orientato alle applicazioni, facilmente scalabile e permette facilità di sviluppo. Se vogliamo nuovamente usare l’esempio del libro, in un database nosql tipo Mongodb il libro potrebbe essere rappresentato in formato Json con tutti i suoi attributi.
- ORM: è una tecnica di programmazione utilizzata per eseguire operazioni all’interno di un database nei linguaggi di programmazione orientati agli oggetti. Diciamo quindi come una sorta di traduttore dal mondo ad oggetti a quello dei database, in particolare quelli relazionali. Permette di scrivere query ottimizzate e più piccole anche se non tutte le query possono essere scritte così e ovviamente non deve essere una scusa per non conoscere l’sql e tutti i meccanismi sottostanti. Come esempio potrei fare direttamente quello dell’entity framework di C# che permette di mappare oggetti presenti in un db come classi e effettuare operazioni su di esso, ad esempio sempre il nostro libro, tabella del db contente (id,titolo,descrizione,autore)

puo diventare una classe con ogni campo public più attributi per specificare la chiave primaria.

- La SQL injection è un attacco ad applicazioni che gestiscono database in cui non viene fatto un opportuno controllo dell'input. Facciamo l'esempio in cui dobbiamo ritornare tutti i dati di un libro preso in input un id numerico, preso tramite un semplice textbox presenta in una pagina html, se non filtro opportunamente alcuni caratteri di escape potrei incorrere in una sql injection. ad esempio, la query per recuperare il libro sarebbe una query del tipo select * from books where id = input_utente, ma se l'input dell'utente invece di essere un id numerico fosse qualcosa del tipo 3 ma fosse 3; drop table books; Non essendoci alcun controllo, il db selezionerebbe si il libro con id 3 ma poi eseguirebbe anche l'altra query, come risultato si sarebbe cancellata tutta la tabella contenente libri! Un danno incalcolabile. è facilmente risolvibile con alcuni accorgimenti tra cui: l'utilizzo di orm, l'opportuno filtraggio di caratteri di escape, oppure parametrizzando il valore che si deve passare alla query e con controllo dei permessi del db.
- L'autenticazione a due fattori prevede che l'autenticazione venga effettuata tramite due canali differenti. Ad esempio, siamo da sempre abituati alla classica autenticazione mediante username e password, ne consegue pero' che qualora qualcuno entri in possesso della nostra password il nostro account, o cio' che vogliamo proteggere, è stato compromesso. Se introduciamo l'autenticazione a due fattori questo problema viene meno, poiché bisognerebbe "hackerare" due canali per poter prendere il controllo dell'account o della risorsa. Ad esempio oltre al semplice login con password, l'invio di un codice otp sullo smartphone tramite sms o app di autenticazione, anche se l'attaccante conoscesse la nostra password dovrebbe comunque avere accesso anche al nostro smartphone per poter procedere, cosa molto più improbabile o viceversa se entrasse in posesso dello smartphone non conoscerebbe comunque la password, questo rende le procedure di autenticazione molto più robuste.
- Le password su un database non vanno mai salvate in chiaro. Vanno salvate dopo essere state processate da un algoritmo di hash, ad esempio sha256. Così non salviamo la password ma soltanto il suo hash, quando l'utente inserisce la password viene calcolato l'hash e si fa il confronto se combaciano o meno. Questo protegge da attacchi, se un hacker entrasse in possesso di tutto il db non disporrebbe comunque delle password in chiaro. La cosa puo' essere ulteriormente complicata aggiungendo un salt alle password prima di essere "hashate", ad esempio invertendo alcune lettere o altre operazioni del genere.
- Un sistema di load balancing prevede la distribuzione di richieste da parte di un client a vari server, per permettere una distribuzione equa del carico. Con sticky session intendiamo che il sistema di load balancing instrada la richiesta del client al primo server che ha gestito la prima richiesta. Questo puo' portare a una distribuzione non equa del carico sulle varie macchine.

ESERCIZI PRATICI:
Scrivi una query che estragga il numero di persone con meno di 30 anni che guadagnano più di 50.000 dollari l'anno

```
select count(*) from records where age < 30 and over_50k = true
```

Scrivi una query che riporti il guadagno di capitale medio per ogni categoria lavorativa

```
select round(avg(capital_gain)), name from records r join workclasses w
on r.workclass_id = w.id group by(w.id)
```

(Mi sono permesso di usare round per evitare numeri con la virgola)

Il resto è implementato come codice In Asp .Net Core 5.

```
create view records_denormalized as
select r.id, r.age, r.education_num, r.capital_gain, r.capital_loss,
r.hours_week, r.over_50k, co.name as country, ra.name as race, o.name as
occupation, re.name as relationship, ed.name as education_level,
ma.name as marital_status, s.name as sex
from records r join races ra on r.race_id = ra.id
join occupations o on r.occupation_id = o.id join relationships re on
r.relationship_id = re.id
join education_levels ed on r.education_level_id = ed.id
join countries co on r.country_id = co.id
join marital_statuses ma on r.marital_status_id = ma.id
join sexes s on r.sex_id = s.id
```