***REPORT DI PROGETTO***

Il progetto proposto consiste nello sviluppare e realizzare un semplice servizio web che interagisca con il contenuto di un feed URL dato. Il feed contiene un elenco di post in formato JSON che l'applicazione dovrà andare a leggere.

Di seguito si riportano i requisiti richiesti.

| |
|---|
| Il web service dovrà poter leggere contenuti json da un feed |
| Il web service dovrà essere sviluppato in NodeJS+Express |
| Il web service dovrà essere deployato su un cloud server |
| Le funzionalità esposte saranno:<br>• Restituzione dell'elenco di post con eventuali opzioni di filtraggio per titolo e/o numero massimo di elementi da restituire<br>• Interazione con DB mySQL nel quale salvare i contenuti del feed a richiesta e possibilità di restituire i contenuti del DB quando richiesto |
| Documentazione e commenti nel sorgente |
| Caricare il materiale su GitHub |

L'applicazione espone 4 funzionalità invocabili riportate di seguito.

| | |
|---|---|
| GET /posts | Restituisce tutti i post |
| GET /posts-filtered | Restituisce solo i post che fanno match con i parametri "title" e "items" |
| GET /sync-db | Legge i contenuti del feed e li persiste su DB |
| GET /posts-db | Restituisce il contenuto del DB |

**SCELTE IMPLEMENTATIVE**

Per esporre le 4 funzionalità descritte sopra si ricorre alle API di **Express** che permettono la creazione dei 4 corrispettivi endpoint invocabili tramite chiamate GET.

Quanto riguarda la **lettura** dal feed si è utilizzato il pacchetto "*https*" di NodeJS che permette, tramite le sue API, di leggere i dati contenuti nel feed tramite l'URL specificata. In particolare, tramite l'endpoint "/posts" vengono restituiti in formato testuale tutti i post contenuti nel feed. Inoltre, è possibile effettuare un'operazione di filtraggio tramite l'endpoint "/post-filtered" e i suoi parametri *title* e *items*. Vengono restituiti i primi *items* post il cui titolo contiene la stringa specificata in *title*.

Quanto riguarda la parte di **persistenza** su data-store si è scelto di utilizzare un servizio cloud. In particolare, si è scelto per la sua semplicità di adottare AWS RDS che permette di istanziare rapidamente dei database mySQL. Tramite le API di AWS RDS si è creato un semplice DB usato nel progetto per la persistenza dei dati. Visto il formato JSON dei post, si è scelto ai fini dell'esempio di persistere per praticità solamente i campi ID, DATE, TITLE, CONTENT e LINK. Si è quindi creata un'opportuna tabella "posts" con queste colonne nel DB che poi verrà usata dall'applicazione per salvare e reperire i dati.

L'interazione tra l'applicazione e il DB avviene tramite richieste SQL e quindi si è scelto di adottare il pacchetto "*mysql*" di NodeJS che offre tutte le API necessarie. Tramite l'endpoint "/sync-db" l'applicazione legge i post dal feed e procede con la persistenza sul DB. Tramite l'endpoint "/post-db" l'applicazione legge i dati dalla tabella contenuta nel DB e li restituisce al cliente.

Si passa ora a descrivere il **deploy**. Il requisito inerente al deploy specifica solamente l'uso di un servizio di **cloud**, lasciando libertà sul provider specifico. Viste la semplicità dell'applicazione, le dimensioni ridotte e lo scopo meramente esemplificativo del progetto si è scelto di ricorrere ad una soluzione **FaaS** per avere un deploy estremamente rapido, agevole e facilitato. In particolare, si è scelto di ricorrere ancora alla suite AWS e quindi la scelta ricade sul loro servizio serverless chiamato Lambda che permette il deploy di applicazioni FaaS. Per fare ciò è necessario un involucro che incapsuli l'applicazione e ne permetta il deploy come FaaS. Ciò è possibile adottando il pacchetto "*serverless-http*" di NodeJS.

Per effettuare il deploy come FaaS su AWS Lambda è necessario caricare un file .zip contenente il sorgente e le dipendenze dei pacchetti NodeJS usati. Fatto ciò, l'applicazione serverless è già funzionante e invocabile tramite l'URL fornita da AWS Lambda.

Infine, si è proceduto a caricare il materiale su GitHub e a redìgere tale documentazione di progetto.

**SVILUPPI FUTURI**

- Dividere l'app in moduli indipendenti per ciascuna funzionalità (lettura feed, filtraggio, persistenza,…)
- Introdurre meccanismo di login al DB non hard-coded
- Aggiunta chiavi primarie DB
- Rendere l'app SOA-compliant