Traccia B: Web Applications Development

Titolo

BrainJobs

Contesto

BrainJobs è un (ipotetico) servizio cloud di tipo Software-as-a-Service (SaaS) che offre ai suoi utenti la possibilità di “allenare” modelli di apprendimento automatico, di valutarne le prestazioni ed (eventualmente) riutilizzarli per effettuare simulazioni.

Il sistema permette agli utenti di effettuare richieste di allenamento o simulazione caricando i dati insieme al modello o utilizzandone uno già precedentemente allenato e salvato nel proprio archivio. In base al linguaggio o al framework utilizzato per il codice del modello, BrainJobs lancia la computazione in un particolare ambiente di esecuzione che verrà istanziato “on-the-fly” in un’altra piattaforma cloud di tipo Serverless basata su containers (es: Apache OpenWhisk, Knative, …).

Gli utenti possono sottomettere più richieste consecutive. Esse verranno gestite in parallelo in un sistema a coda. Ogni richiesta di un utente corrisponde ad un task di lavoro (job).

Gli utenti possono controllare lo stato delle loro richieste dalla dashboard di BrainJobs, ed una volta terminate, visualizzarne i risultati. Successivamente, il sistema permette di scartare o salvare il modello per utilizzi futuri.

L’architettura del servizio BrainJobs è suddivisa in tanti servizi e componenti, ognuno con un compito ben specifico. Al vostro team, è richiesta la creazione di due componenti:

1. un componente di frontend implementato utilizzando HTML, CSS e JavaScript che utilizza il paradigma AJAX per inviare/ricevere dati
2. un componente di backend che espone una HTTP API REST

Il frontend deve permettere ad un utente di creare una nuova richiesta di allenamento, visualizzare la lista delle sue richieste e visualizzare le informazioni di dettaglio di ogni richiesta.

Il backend deve essere in grado di salvare una nuova richiesta, fornire la lista delle richieste di un utente e restituire informazioni di dettaglio di ogni richiesta.

Una volta che il backend ha salvato una nuova richiesta, altri servizi di BrainJobs si occuperanno di lanciare la computazione, aggiornare lo stato del job ed aggiungere i risultati. Il compito del vostro team è **esclusivamente** quello fornire un frontend ed un backend con le funzionalità sopra indicate.

Descrizione

Versione base - fino ad un massimo di 2 punti

Realizzare un'applicazione web dinamica costituita da:

- HTTP API REST che offra le seguenti funzionalità
  - ricevere in ingresso una richiesta di allenamento in formato JSON (vedi schema più avanti) e restituire la risorsa creata con status code *201 Created* e *Location* header con valore pari all'identificativo del job (*job_id*)
  - restituire in formato JSON la lista di tutte le richieste di un utente
  - restituire in formato JSON una singola richiesta utilizzando il suo id (*job_id*)
- un client web (HTML/CSS/JS) che utilizzando il paradigma AJAX offra le seguenti funzionalità:
  - inviare una richiesta di allenamento in formato JSON
  - visualizzare la lista delle richieste di un utente mostrando i campi *job_id, title, language, framework, created_at, status*
  - visualizzare i dettagli di una richiesta mostrando tutti i campi

Una richiesta di allenamento è costituita dai seguenti campi:

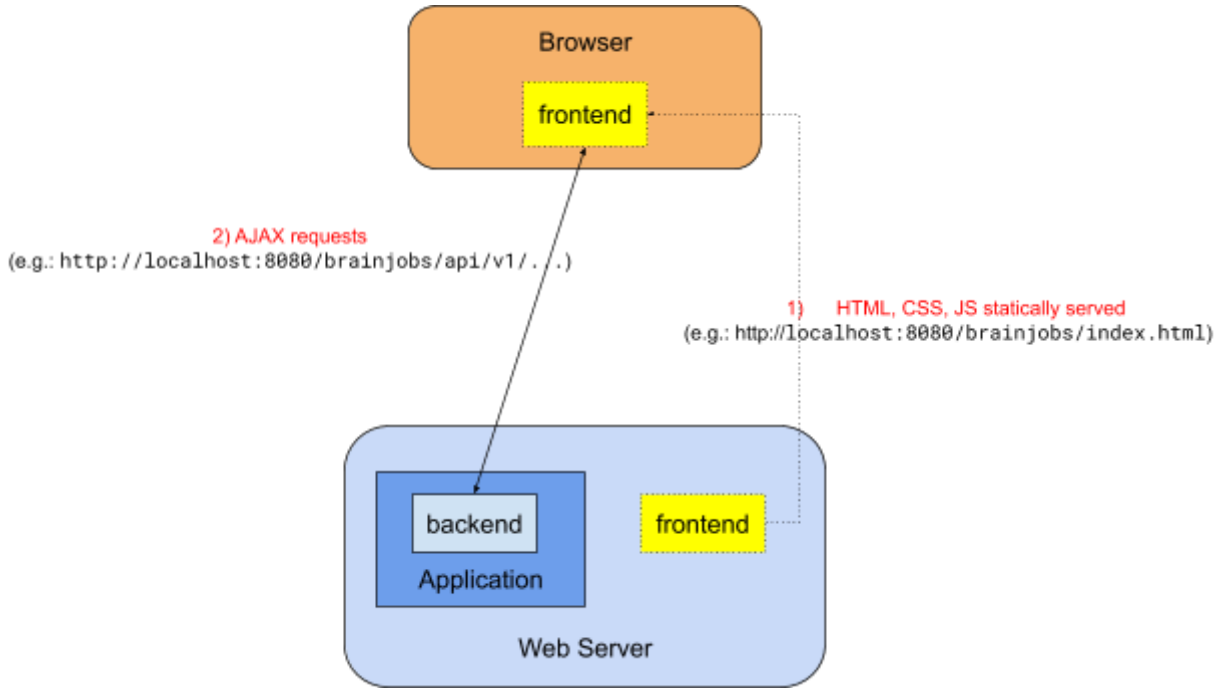
- **user_id** *[string, required]*
- **title** *[string, required]*
- **language** *[string, required, values_in: python, java, scala, r, c++, julia]*
- **framework** *[string, optional, values_in: pytorch, tensorflow, caffe, keras, deeplearning4j, apache_mahout, apache_singa]*
- **dataset** *[string, required]*
- **dataset_datatype** *[string, required, values_in: csv, avro, json]*
- **model** *[string, required]*
- **job_id** *[string, required]*
- **created_at** *[datetime, required]*
- **status** *[string, required, values_in: created, enqueued, building, running, terminated, failed, unknown]*

I campi in **rosso** sono aggiunti dal backend durante la creazione della richiesta, i campi in **nero** sono passati in ingresso al backend.

Il campo *model* per semplicità è stato indicato come stringa, supponete che contenga pezzi di codice sorgente nel linguaggio/framework indicato rispettivamente nei campi *language* e *framework*.

I valori riportati per i campi *language*, *framework*, *dataset_datatype* e *status* possono essere implementati utilizzando una *Enumeration* e sono gli unici valori accettati dal sistema.

Di seguito viene fornito un diagramma per esplicitare l'architettura dell'applicazione.



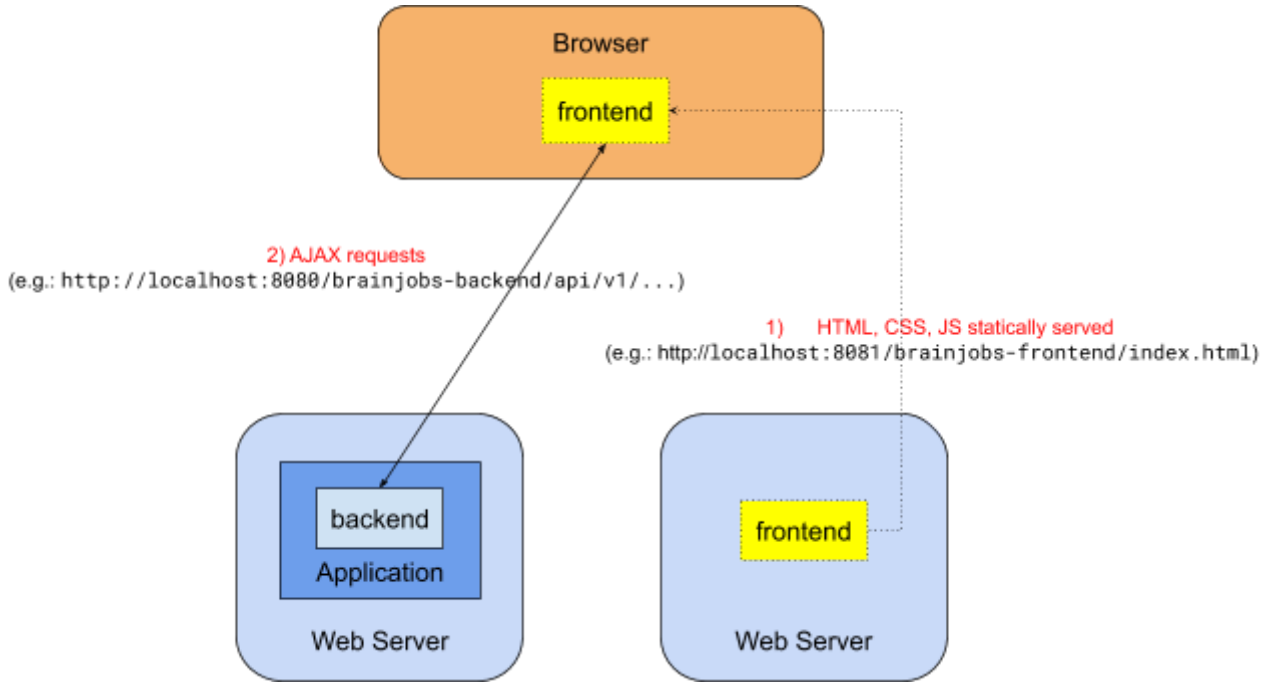
Variante 1 - fino ad un massimo di 3 punti

Le funzionalità del sistema da implementare sono le stesse della versione base, ma cambia la visione architetturale.

È richiesta una netta separazione del componente di frontend da quello di backend. Questo implica che devono essere creati due differenti progetti. Essi possono essere eseguiti sullo stesso server, ma avranno due percorsi (application path) differenti.

Il progetto dell'applicazione di frontend può essere un progetto web statico mentre quello dell'applicazione di backend deve essere un progetto web dinamico.

Il seguente diagramma esplicita questa prima variante architetturale.



Variante 2 - fino ad un massimo di 4 punti

Come nella *Variante 1*, le funzionalità restano quelle della versione base, ma viene complicata ulteriormente l'architettura.

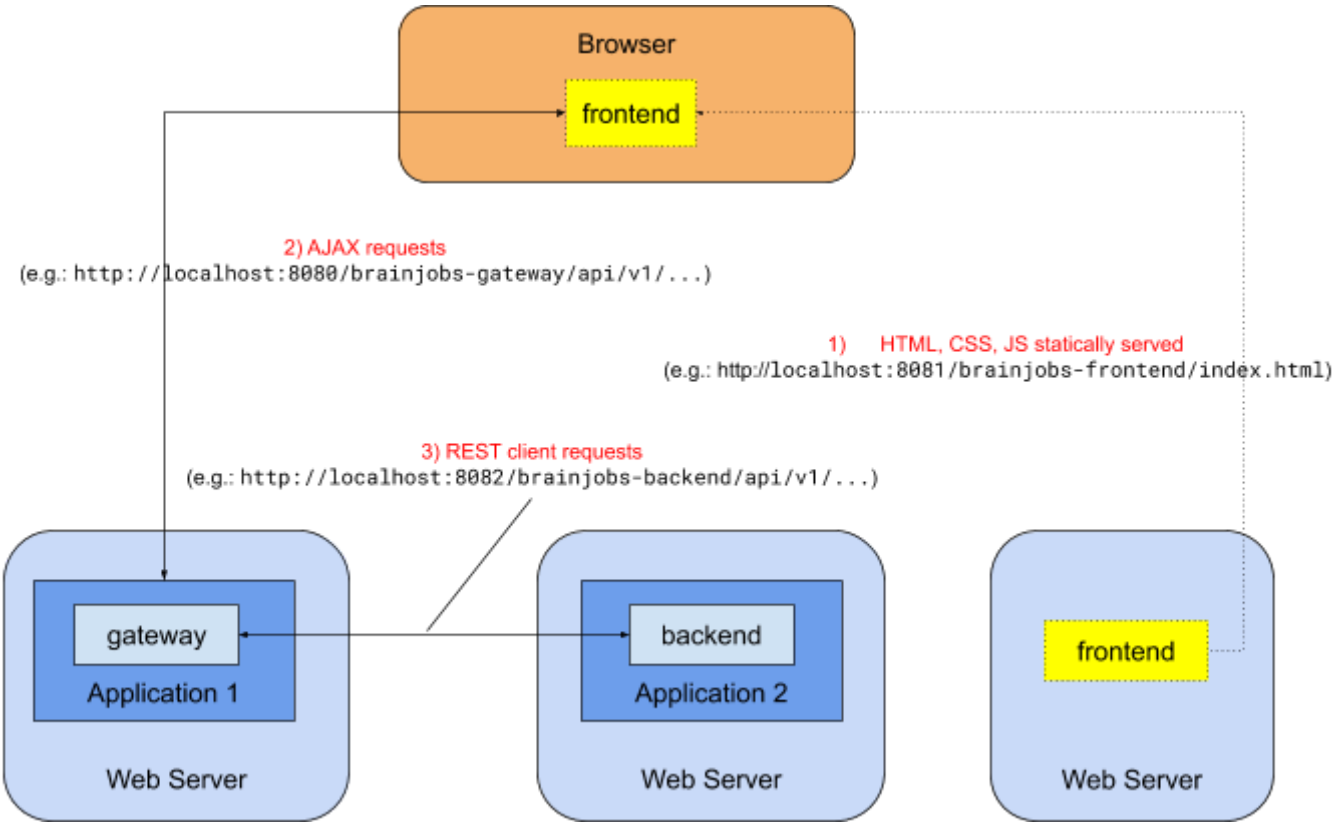
In questa variante viene introdotto il concetto di API Gateway. Viene richiesto quindi di creare 3 progetti:

- frontend
- api_gateway
- backend

L'API Gateway è un componente che, fra le sue funzionalità, maschera la presenza di uno o più backend alle applicazioni di frontend, occupandosi di esporre una API univoca e di aggregare, filtrare e comporre richieste/risposte per/da applicazioni di backend.

L'API Gateway, in questo caso, può replicare per semplicità la stessa API esposta dal backend ma il suo compito sarà quello di utilizzare internamente un REST Client per inoltrare le richieste del frontend al backend e restituirne le risposte.

Il seguente diagramma esplicita la seconda variante architetturale.



Tips&Tricks

Di seguito vengono fornite ulteriori informazioni, risorse, riferimenti e piccoli aiuti per lo svolgimento della traccia sotto forma di domanda/risposta.

1. **Posso utilizzare altri linguaggi/framework?** *Sì, è possibile sia utilizzare framework di frontend come Bootstrap sia implementare il sistema nel linguaggio che preferisci. Usa gli strumenti che conosci meglio, a patto di rispettare l'architettura e le funzionalità richieste.*

2. **Non ho capito bene cosa sia un API Gateway, dove posso reperire informazioni?** *Ottime risorse sono:*
    a. *https://microservices.io/patterns/apigateway.html*
    b. *https://docs.microsoft.com/it-it/dotnet/standard/microservices-architecture/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern*
    c. *https://www.nginx.com/learn/api-gateway/*

3. **Cos'è un REST Client?** *È un software in grado di effettuare richieste ad una API REST. Ad esempio, nel laboratorio dedicato a REST, è stato utilizzato il Jersey REST Client all'interno dei test.*

4. **Sono un po' perso nella definizione degli endpoint dell'API...qualche aiuto?** *Di seguito vengono elencate alcune possibili modalità di definizione degli endpoint:*
    a. *Job-centric:*

    i. `/api/v1/jobs?user_id={user_id}` *[GET: visualizza tutti i jobs di un utente]*
    ii. `/api/v1/jobs/{job_id}` *[GET: visualizza i dettagli di un job]*
    iii. `/api/v1/jobs` *[POST: aggiunge un job, l'id dell'utente al quale è associato è contenuto nella richiesta]*
  b. *User-centric:*
    i. `/api/v1/users/{user_id}/jobs` *[GET/POST: visualizza tutti i job di un utente o aggiunge un job, in questo caso l'id dell'utente nella richiesta è lo stesso nel path]*
    ii. `/api/v1/users/{user_id}/jobs/{job_id}` *[GET: visualizza i dettagli un job]*

5. **Dove salvo le richieste (jobs) nel componente di backend?** *Se hai le conoscenze per utilizzare un database puoi farlo. Altrimenti puoi utilizzare delle strutture dati temporanee (esempio: hash maps, dizionari, etc…) come visto negli esempi a lezione.*