BrainJobs

UniShare

Davide Cozzi
@dlcgold

Gabriele De Rosa
@derogab

Indice

Capitolo 1

Introduzione

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.

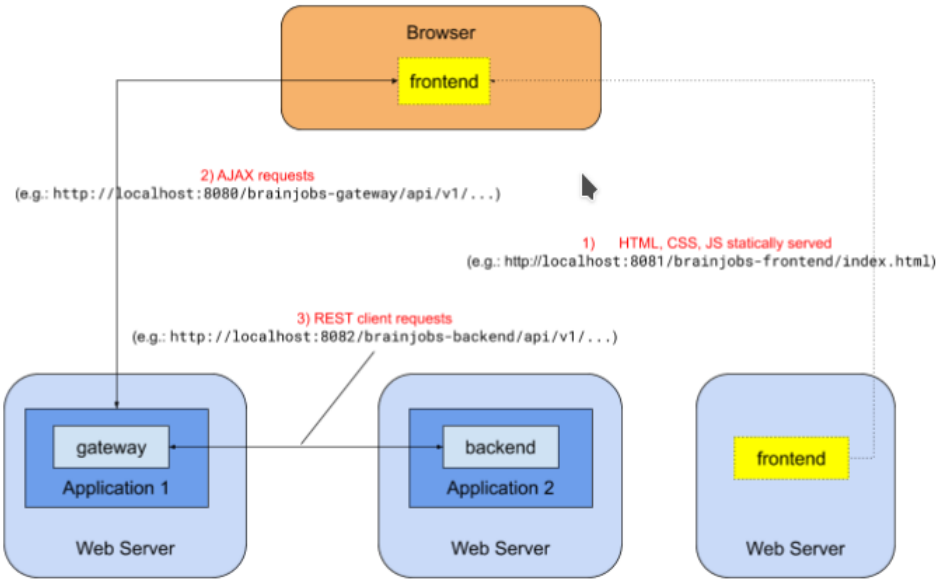

Figura 1.1: struttura finale del progetto

Capitolo 2

FrontEnd

Iniziamo parlando del frontend. Per quanto riguarda l'aspetto estetico è stato usato un tema di Bootstrap per poter rappresentare più semplicemente componenti come il menù presente nella parte alta della pagina, contenente le informazioni del progetto, e la navbar con il bottone per richiamare il menù. Entrambi componenti sono nel tag *header*.

Figura 2.1: navbar

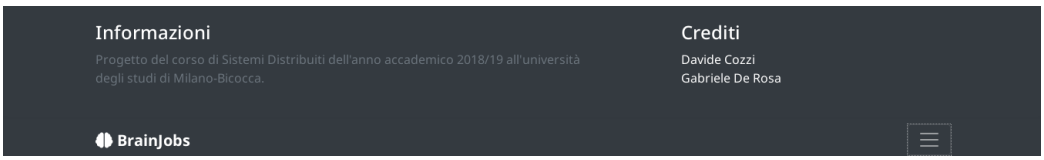

Figura 2.2: menù a scomparsa

navbar:

```
<div class="navbar navbar-dark bg-dark shadow-sm">
 <div class="container d-flex justify-content-between">
  <a href="#" class="navbar-brand d-flex align-items-center">
   <strong><i class="fas fa-brain"></i> BrainJobs</strong>
   </a>
```

```
  <button class="navbar-toggler" type="button"
    data-toggle="collapse"
    data-target="#navbarHeader" aria-controls="navbarHeader"
    aria-expanded="false" aria-label="Toggle navigation">
   <span class="navbar-toggler-icon"></span>
  </button>
 </div>
</div>
```

Menù:

```
<div class="collapse bg-dark" id="navbarHeader">
 <div class="container">
  <div class="row">
   <div class="col-sm-8 col-md-7 py-4">
    <h4 class="text-white">Informazioni</h4>
    <p class="text-muted">
     Progetto del corso di Sistemi Distribuiti dell'anno
      accademico 2018/19 all'università degli studi di
       Milano-Bicocca.
    </p>
   </div>
   <div class="col-sm-4 offset-md-1 py-4">
    <h4 class="text-white">Crediti</h4>
     <ul class="list-unstyled">
      <li><a class="text-white"
       href="https://www.github.com/dlcgold">
        Davide Cozzi</a></li>
      <li><a class="text-white"
       href="https://www.github.com/derogab">
        Gabriele De Rosa</a></li>
     </ul>
    </div>
   </div>
  </div>
</div>
```

Si passa poi alla sezione con il titolo della pagina, con l'icona presa da quella messe a disposizione sul sito `https://fontawesome.com/icons/`:

Figura 2.3: titolo della pagina

```
<section class="jumbotron text-center">
 <div class="container">
  <h1 class="jumbotron-heading"><i class="fas fa-brain"></i>
   BrainJobs</h1>
 </div>
</section>
```

Analizziamo ora una delle parti principali della pagina: **il form di inserimento dati**. Qui si è fatto uso della classe *form-group* per impostare i vari campi del form, con titolo e *form-control* per l'inserimento, e della classe *custom-select* per quei campi con selezione obbligatoria, dove quindi è stato aggiunto un selettore:

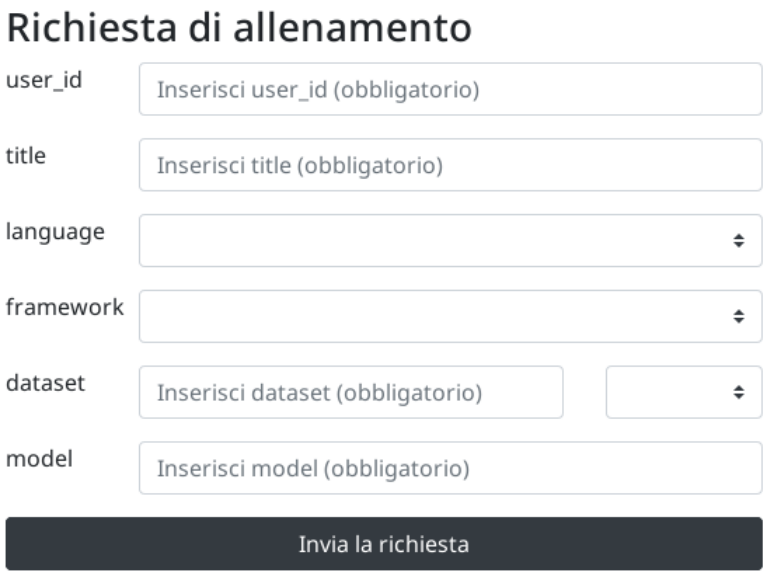

Figura 2.4: form per l'inserimento della richiesta

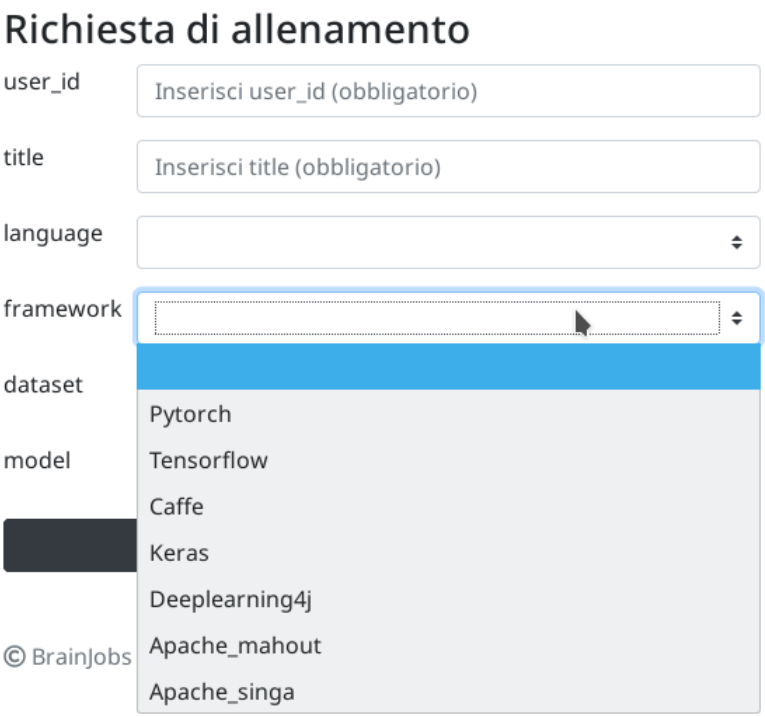

Figura 2.5: esempio di selettore

Vediamo quindi, per esempio, la parte nell'*index.html* dedicata all'inseriemnto di *title*, ovvero un inserimento manuale senza selettore:

```
<div class="row">
 <div class="col-md-2 col-sm-12">
  <label for="title">title</label>
 </div>
 <div class="col-md-10 col-sm-12">
  <input type="text" class="form-control" id="title"
   placeholder="Inserisci title (obbligatorio)">
 </div>
</div>
```

Dove notiamo come le classi *col-md-n* e *col-sm-n* permettono di rendere *responsive* gli elementi (la stringa e il box di inserimento).

Passiamo ora a vedere l'esempio di un inserimento mediante selettore, prendendo come esempio l'inserimento del framework:

```
<div class="form-group">
 <div class="row">
  <div class="col-md-2 col-sm-12">
   <label for="framework">framework</label>
  </div>
  <div class="col-md-10 col-sm-12">
   <select class="custom-select" class="form-control"
     id="framework">
    <option selected></option>
    <option value="pytorch">Pytorch</option>
    <option value="tensorflow">Tensorflow</option>
    <option value="caffe">Caffe</option>
    <option value="keras">Keras</option>
    <option value="deeplearning4j">Deeplearning4j</option>
    <option value="apache_mahout">Apache_mahout</option>
    <option value="apache_singa">Apache_singa</option>
   </select>
  </div>
 </div>
</div>
```

Analizziamo ora la seconda parte fondamentale della pagina, dove l'utente può inserire uno *user_id* o un *job_id* per effettuare una query nel database delle richieste:

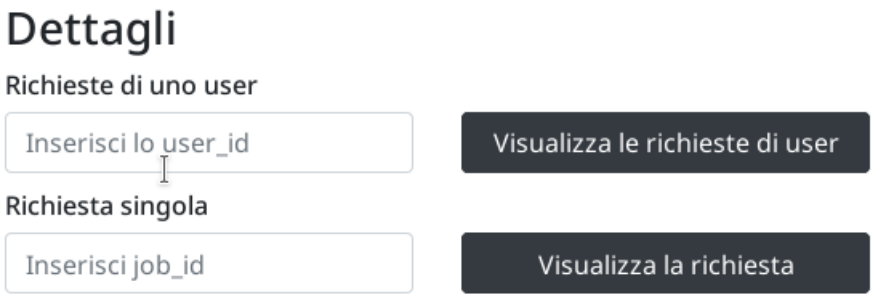

Figura 2.6: campi per l'inserimento di query

Nell'*index.html* si ha quindi:

```
<h3>Dettagli</h3>

<h6>Richieste di uno user</h6>
 <div class="row">
  <div class="col-md-6">
   <input type="text" class="form-control"
     id="user_id_search"
      placeholder="Inserisci lo user_id">
  </div>
  <div class="col-md-6">
   <button id="get-all-requests"
     class="btn btn-dark btn-block">
      Visualizza le richieste di user</button>
  </div>
 </div>

 <h6 style="margin-top: 10px;">Richiesta singola</h6>
 <div class="row">
  <div class="col-md-6">
   <input type="text" class="form-control"
     id="job_id" placeholder="Inserisci job_id">
  </div>
  <div class="col-md-6">
   <button id="get-single-request"
    class="btn btn-dark btn-block">
     Visualizza la richiesta</button>
  </div>
 </div>
```

infine i risulati della query verranno visualizzati mediante:

```
<div id="results"></div>
```

Infine una parola per tutta quella parte del file dedicata al permettere l'uso di *bootstrap*, *jquery* e del *custom.js* mediante il quale, con l'uso di **ajax**, sono state fatte le *POST* e le *GET*:

```
<!-- nell'HEAD -->

<!-- Bootstrap core CSS -->
<link href="css/bootstrap.min.css" rel="stylesheet">

<!-- FA -->
<link href="css/fa.css" rel="stylesheet">

<!-- Custom styles for this template -->
<link href="css/custom.css" rel="stylesheet">

...


<!-- alla fine del file -->

<!-- jQuery -->
<script src="js/jquery.min.js"></script>
<script>window.jQuery ||
 document.write('<script src="js/jquery.min.js">
  <\/script>')</script>

<!-- Bootstrap bundle JS -->
<script src="js/bootstrap.bundle.min.js"></script>

<!-- Custom javascript script w/ ajax requests-->
<script src="js/custom.js"></script>
```

Abbiamo visto il file *HTML* ma questo non basta in quanto serve il *custom.js* per interagire, mediante jquery e ajax, con il backend. Si ha quindi la seguente struttura per il **frontend**:

```
.
   css:
       bootstrap.css
       bootstrap.css.map
       bootstrap-grid.css
       bootstrap-grid.css.map
       bootstrap-grid.min.css
       bootstrap-grid.min.css.map
       bootstrap.min.css
       bootstrap.min.css.map
       bootstrap-reboot.css
       bootstrap-reboot.css.map
       bootstrap-reboot.min.css
       bootstrap-reboot.min.css.map
       custom.css
       fa.css
       fa.min.css
   index.html
   js:
       bootstrap.bundle.js
       bootstrap.bundle.js.map
       bootstrap.bundle.min.js
       bootstrap.bundle.min.js.map
       bootstrap.js
       bootstrap.js.map
       bootstrap.min.js
       bootstrap.min.js.map
       custom.js
       fa.js
       fa.min.js
       jquery.js
       jquery.min.js
   webfonts:
        fa-brands-400.eot
        fa-brands-400.svg
        fa-brands-400.ttf
```

```
fa-brands-400.woff
fa-brands-400.woff2
fa-regular-400.eot
fa-regular-400.svg
fa-regular-400.ttf
fa-regular-400.woff
fa-regular-400.woff2
fa-solid-900.eot
fa-solid-900.svg
fa-solid-900.ttf
fa-solid-900.woff
fa-solid-900.woff2
```

con tutti i file per la parte di CSS, tutto il necessario per jquery e bootstrap, i fonts etc...
Ci concentriamo ovviamente sul *custom.js* che abbiamo scritto per interfacciare frontend e backend.