Corso di Programmazione Web & Mobile

A.A. 2020-2021

Sentiment Analysis tramite un Naive-Bayes Classifier

Andrea Tinelli

Matricola: 941800

Analisi dei requisiti

Il progetto si pone come obiettivo quello di realizzare uno strumento per la classificazione di testi in grado di essere addestrato attraverso un'interfaccia grafica web.

La gestione di un classificatore di testi attraverso un'interfaccia web permette a qualsiasi tipo utente, anche privo di qualsivoglia fondamento riguardante il funzionamento del classificatore e/o delle tecnologie informatiche necessarie al suo funzionamento, di andare ad interagire con quest'ultimo in maniera semplice ed intuitiva.

L'interfaccia web è dotata di un design responsivo, creato con la popolare libreria Bootstrap e alcuni fogli di stile personalizzati, che adatta l'aspetto della pagina in modo tale da essere visualizzata al meglio sui vari media, permettendo quindi di utilizzare il progetto da qualsiasi device dotato di un web browser in possesso dell'utente.

Il sito permette quindi all'utente, attraverso interfaccia grafica, di andare ad istruire il classificatore inserendo nuove associazioni tra testo e la relativa classificazione, di andare a testare lo stato del classificatore ricevendo una proposta di classificazione per un dato testo, dando la possibilità di venire corretto, e di visionare lo stato del classificatore andando a visionare la classificazione scelta per un dato testo con un grado di affidabilità della decisione.

All'interno del progetto vi è quindi la necessità di avere un flusso costante di testi da poter sottoporre all'utente per poter istruire il classificatore o da sottoporre al classificatore per ottenere una associazione; come fonte di questi testi viene utilizzando Twitter, ottenendo tramite le relative api i tweet più recenti.

L'utente può inoltre reimpostare completamente il classificatore e cancellare i tweet ottenuti per aggiornarsi con quelli più recenti attraverso l'apposita opzione nella pagina delle impostazioni.

Le pagine sono servite da un server creato in Node.js tramite la popolare libreria express, esso, fornisce anche una serie di API per ottenere i tweet ed interagire con il classificatore che risiede lato server e che salva il suo stato all'interno di un file JSON.

Interfacce

L'applicazione è dotata di diverse interfacce tra le quali: una pagina con una breve descrizione del progetto, una pagina con i moduli utilizzati per lo sviluppo del progetto ed una pagina con le impostazioni del progetto, tuttavia, tutte le operazioni che possono essere effettuate per interagire con il classificatore sono contenute in un'unica interfaccia suddivisa in tab.

Questa interfaccia principale contiene quindi:

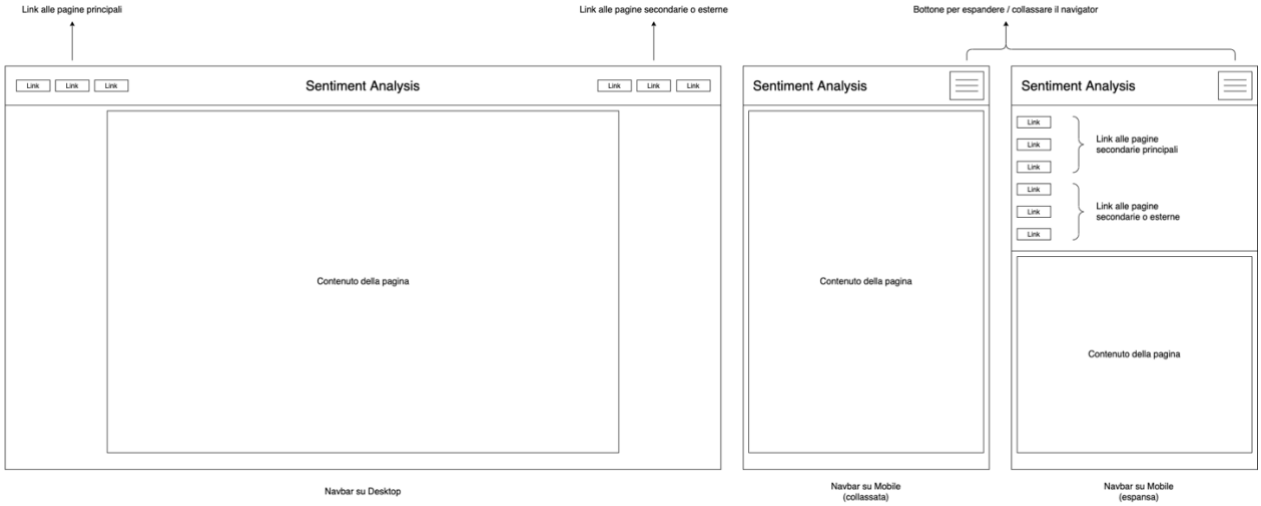
- Una tab per l'addestramento semplice del classificatore: in questa tab viene mostrata una card con un tweet ed una card con una serie di associazioni tra le quali scegliere.
  L'utente può quindi selezionare una porzione di testo del tweet ed associarlo ad una delle categorie disponibili utilizzando le checkbox e confermando con l'apposito bottone, istruendo così il classificatore.
  La tab è dotata di un tasto per saltare l'operazione e passare al tweet successivo.

- Una tab per l'addestramento guidato del classificatore: in questa tab viene mostrata una card con un tweet ed una card con l'associazione scelta dal classificatore per il testo contenuto nel tweet.
  Attraverso un bottone per la conferma e un dropdown l'utente può quindi confermare l'associazione scelta dal classificatore o modificarla.
  La tab è dotata di un tasto per saltare l'operazione e passare al tweet successivo.
- Una tab per la classificazione automatica: in questa tab viene mostrata una card con un tweet ed una card con l'associazione scelta dal classificatore insieme ad un grado di affidabilità di quest'ultima.
  La tab è dotata di un tasto per passare al tweet successivo.

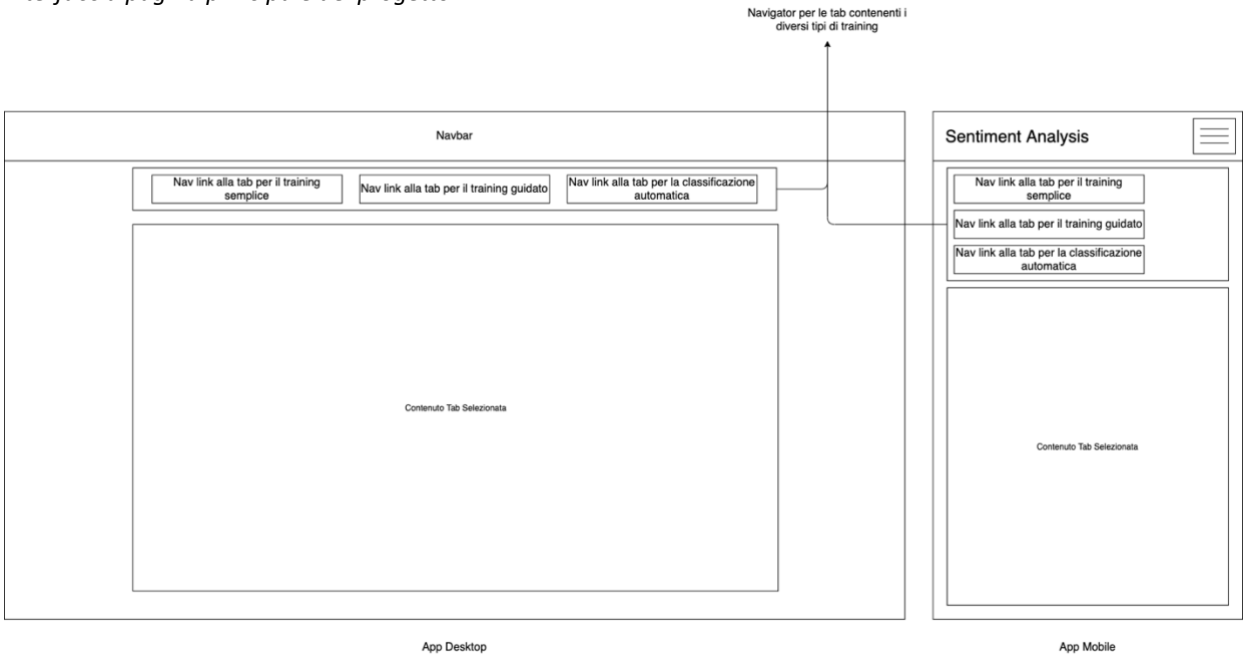
Le interfacce sono stilizzate attraverso la popolare libreria Bootstrap e dei file di CSS.

Progettazione delle interfacce principali:

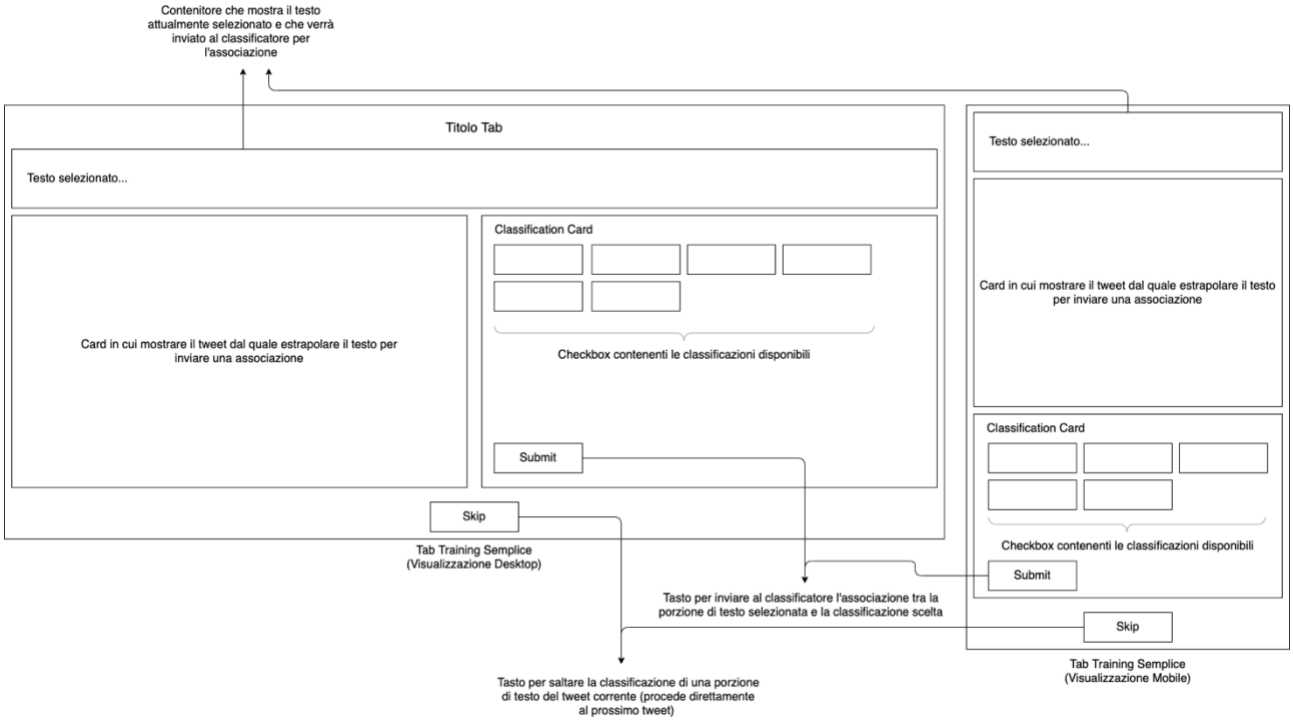
Interfaccia navbar:



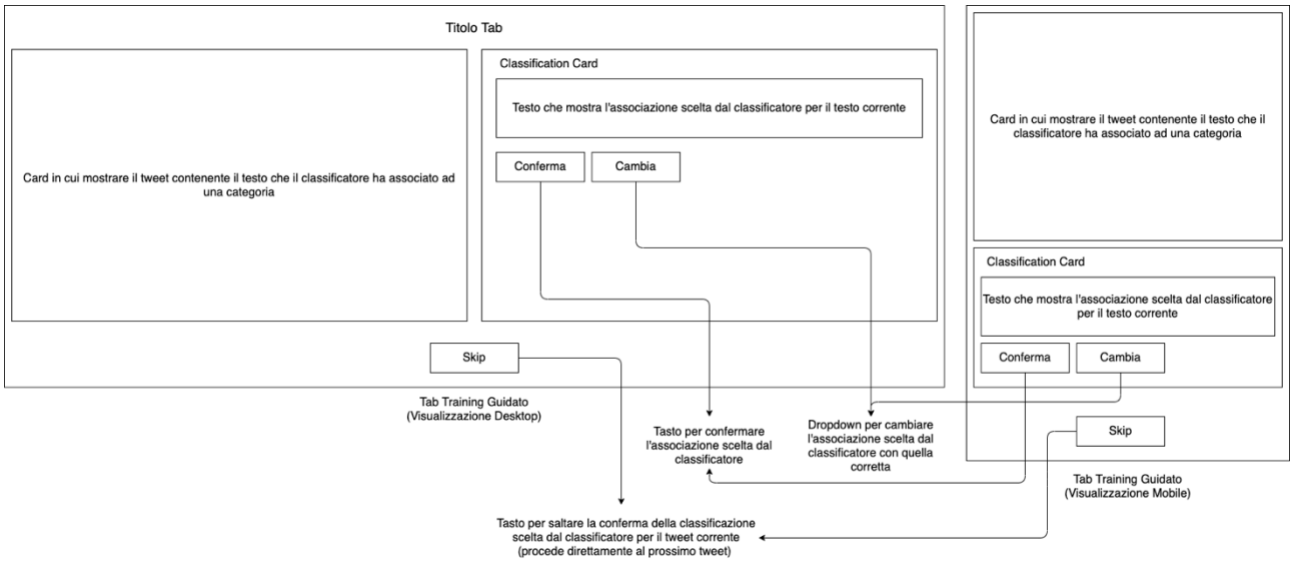
Interfaccia pagina principale del progetto:



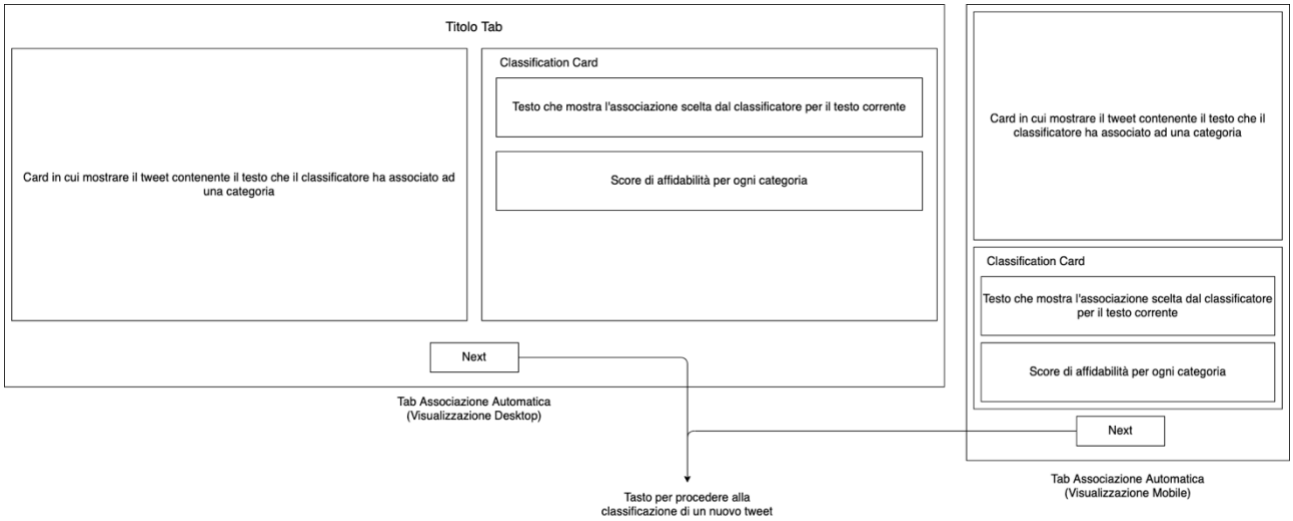
Interfaccia tab per il training semplice:



Interfaccia tab per il training guidato:

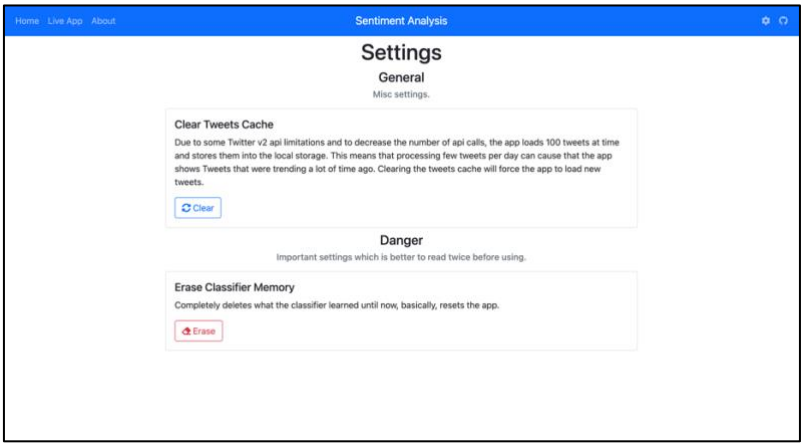


Interfaccia tab per l'associazione automatica:

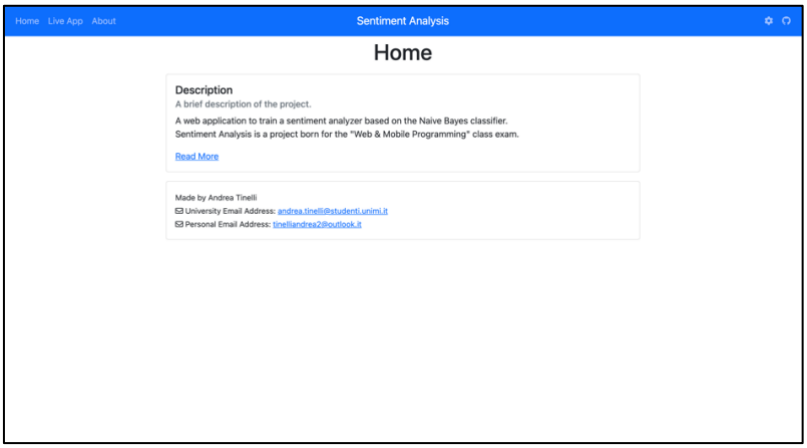


Interfacce finali:

Pagina delle impostazioni

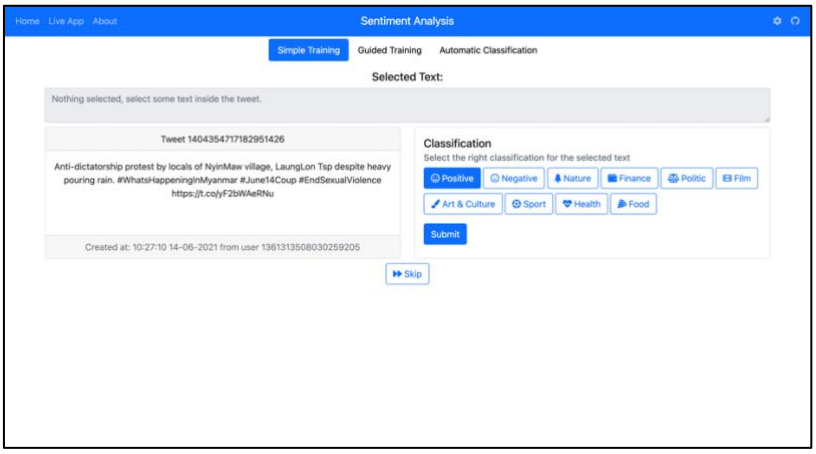


Pagina index

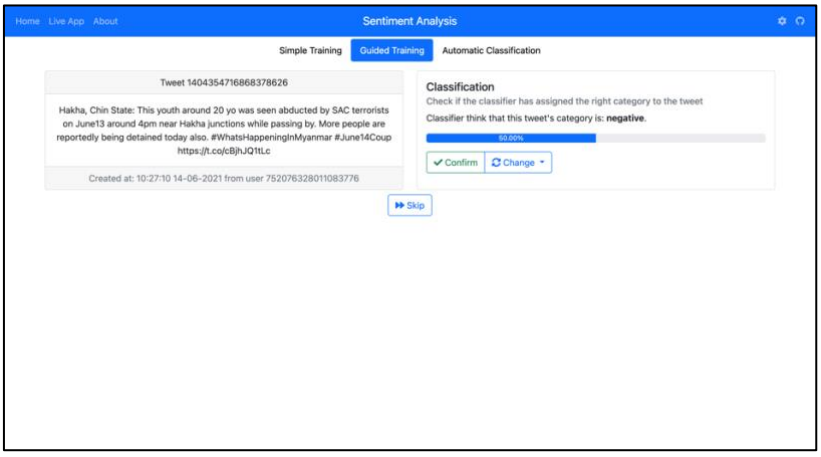


Pagina errore 404

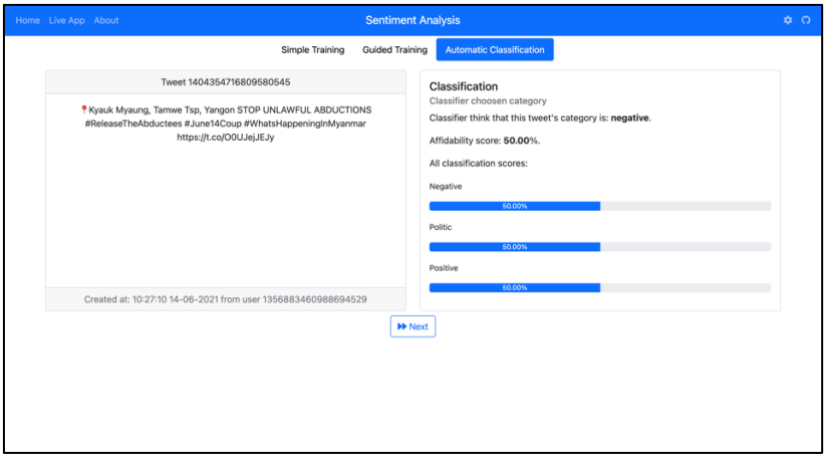
Pagina App con Tab per il training semplice



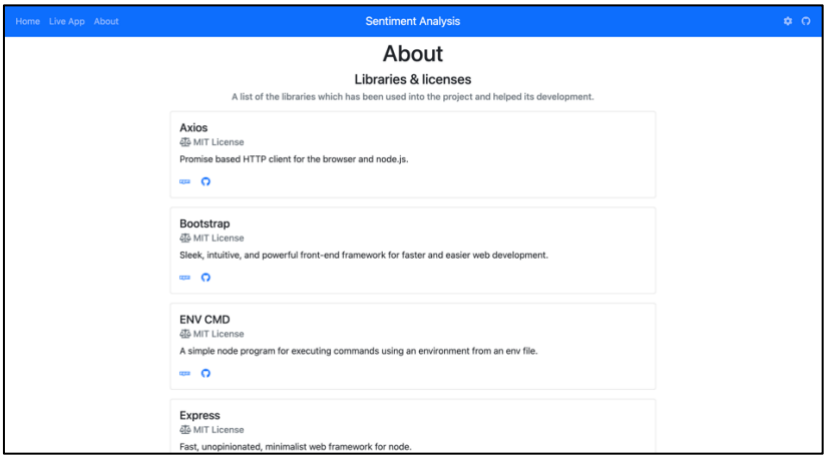
Pagina App con Tab per il training guidato



Pagina App con Tab per associazione automatica

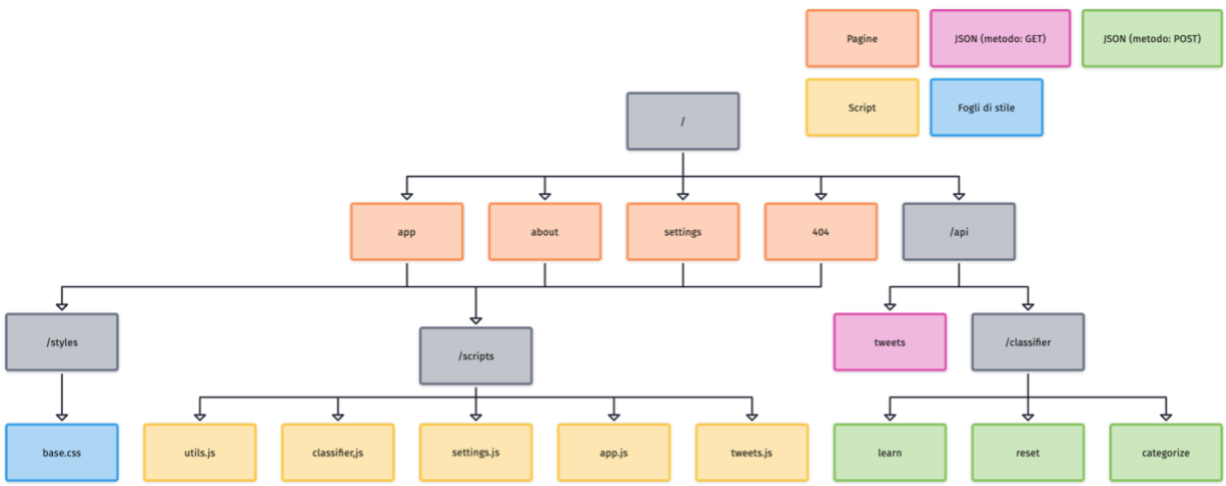


Pagina about



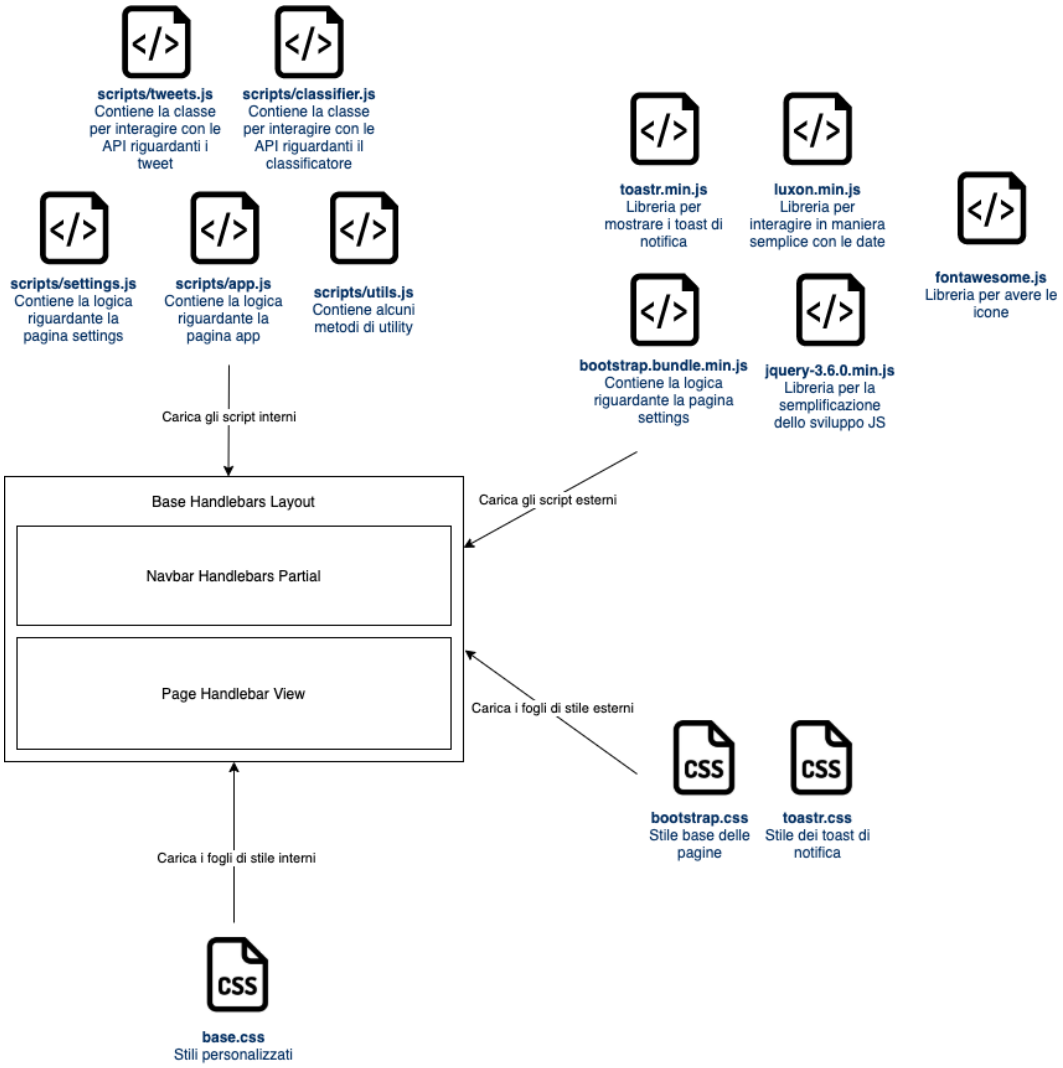
Architettura

Diagramma delle risorse accessibili tramite URI:



Descrizione delle risorse

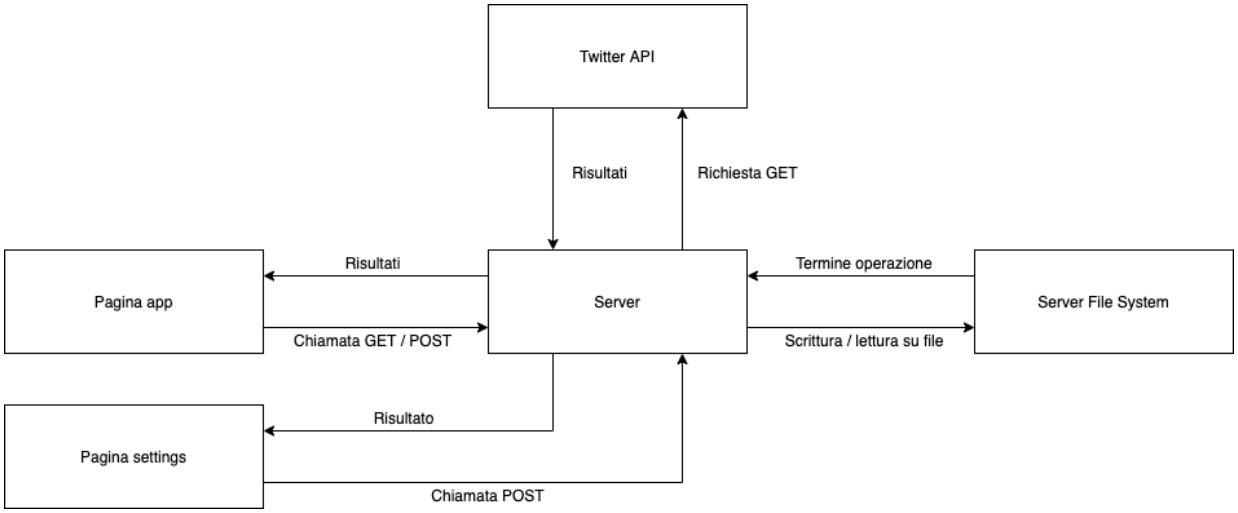
Ogni pagina viene renderizzata dinamicamente al momento della ricezione della chiamata attraverso il template engine Handlebars.
La struttura di ogni pagina ha quindi un layout base di Handlebars che si occupa di fornire una base di partenza uguale per tutte le pagine e di caricare tutto il necessario al loro corretto funzionamento (script, stili ed il partial che costituisce la navbar per spostarsi tra le pagine).
La pagina vera e propria è quindi una view di handlebars che viene passata come body del layout base.



Le api restituiscono dinamicamente dei valori in formato JSON, rispettivamente:

- /api/tweets restituisce i 100 tweets più recenti
- /api/classifier/learn istruisce il classificatore con una nuova associazione
- /api/classifier/reset reimposta il classificatore al suo stato iniziale
- /api/classifier/categorize resituisce le associazioni del classificatore per la frase fornita

Diagramma delle interazioni delle pagine con il server:



Frammenti di codice

Classe per interagire lato client con le API del classificatore:

```
/**
 * OVERVIEW: Classifier Class provides useful methods to iteract with the classifier api
 */
class Classifier {

    /**
     * Calls the api to instruct the classifier with a new text-category association
     *
     * @param {String} text
     * @param {String} category
     * @returns {Promise}
     */
    static learn(text, category) {

        return new Promise((resolve, reject) => {

            // Sends the request to the api to let the classifier learn a new association
            fetch('/api/classifier/learn', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ text, category })
            })
            // Computes the response
            .then(async (response) => {

                // Reads the response content as a JSON
                let data = await response.json();
                // Checks if the request has been completed without errors
                if(response.status >= 300) return reject(new Error(data.message));
                // Returns the data
                resolve(data.message);

            })
            // Error Handling
            .catch((error) => reject(new Error("Unknown error.")));

        })

    }

    /**
     * Calls the api to retrieve the classifier classification of the provided text
     *
     * @param {String} text
     * @returns {Promise}
     */
    static categorize(text) {

        return new Promise((resolve, reject) => {

            // Sends the request to the api to let the classifier learn a new association
```

```
            fetch('/api/classifier/categorize', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ text })
            })
            // Computes the response
            .then(async (response) => {

                // Reads the response content as a JSON
                let data = await response.json();
                // Checks if the request has been completed without errors
                if(response.status >= 300) return reject(new Error(data.message));
                // Returns the data
                resolve(data);

            })
            // Error Handling
            .catch((error) => reject(new Error("Unknown error.")));

        })

    }

    /**
     * Calls the api to reset the classifier
     *
     * @returns {Promise}
     */
    static reset() {

        return new Promise((resolve, reject) => {

            // Sends the request to the api to let the classifier learn a new association
            fetch('/api/classifier/reset', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' }
            })
            // Computes the response
            .then(async (response) => {

                // Reads the response content as a JSON
                let data = await response.json();
                // Checks if the request has been completed without errors
                if(response.status >= 300) return reject(new Error(data.message));
                // Returns the data
                resolve(data);

            })
            // Error Handling
            .catch((error) => reject(new Error("Unknown error.")));

        })

    }

}
```

Route del server per istruire il classificatore con una nuova associazione:

```
// Instructs the Bayes classifier with the submitted text associated to the provided classification
router.post('/api/classifier/learn', async (req, res) => {

    // Accepted classifications
    const acceptedClassifications = [
        'positive',
        'negative',
        'nature',
        'finance',
        'politic',
        'film',
        'art',
        'sport',
        'health',
        'food'
    ];

    // Checks if all data have been correctly provided
    if(
        typeof(req.body.text) !== 'string' ||
        req.body.text.length < 4 ||
        typeof(req.body.category) !== 'string' ||
        !acceptedClassifications.includes(req.body.category)
    ) return res.status(400).send({ message: "One or more needed data are missing or invalid." });

    // Creates the bayes
    let classifier = new BayesClassifier();
    if(fs.existsSync(bayesConfigPath)) classifier
        .restore(JSON.parse(fs.readFileSync(bayesConfigPath)));
```

```
    // Instructs the bayes classifier with the received data
    classifier.addDocument(req.body.text, req.body.category);
    classifier.train();

    // Saves the classifier state
    if(!fs.existsSync(dataPath)) fs.mkdirSync(dataPath);
    fs.writeFileSync(bayesConfigPath, JSON.stringify(classifier), { flag: "w" });

    // Returns the success of the operation
    return res.send({ message: "Succesfully learned." });

})
```

CSS personalizzato:

```
/* App Loading Div */
.app-loader {
    background-color: white;
    z-index: 99999;
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    display: flex;
    align-items: center;
    justify-content: center;
}

/* Page Content Container */
#page-content {
    margin-top: 60px;
}

/* Navbar */
.navbar {
    position: fixed;
    right: 0;
    top: 0;
    left: 0;
    z-index: 1;
}

@media (min-width: 768px) {

    /* Centered Navbar Text */
    .navbar-brand.abs {
        position: absolute;
        left: 50%;
        transform: translate(-50%, 0);
        text-align: center;
    }

}
```

Handlebars partial che renderizza la navbar:

```
<!-- Top Navbar -->
<nav class="navbar navbar-expand-md navbar-dark bg-primary">
    <div class="container-fluid">

        <!-- Project Title -->
        <a class="navbar-brand abs" href="#">Sentiment Analysis</a>

        <!-- Responsive Button -->
        <button class="navbar-toggler ms-auto" type="button" data-bs-toggle="collapse" data-bs-
target="#collapseNavbar">
            <span class="navbar-toggler-icon"></span>
        </button>

        <!-- Nav Options -->
        <div class="navbar-collapse collapse" id="collapseNavbar">

            <!-- Left Options -->
            <ul class="navbar-nav">

                <!-- Home -->
                <li class="nav-item">
                    <a class="nav-link" href="/">Home</a>
                </li>

                <!-- Live App -->
                <li class="nav-item">
                    <a class="nav-link" href="/app">Live App</a>
                </li>

                <!-- About -->
```

```
                <li class="nav-item">
                    <a class="nav-link" href="/about">About</a>
                </li>

            </ul>

            <!-- Right Options -->
            <ul class="navbar-nav ms-auto">

                <!-- Settings -->
                <li class="nav-item">
                    <a class="nav-link" href="/settings"><i class="fas fa-cog"></i></a>
                </li>

                <!-- GitHub -->
                <li class="nav-item">
                    <a class="nav-link" target="_blank" href="https://github.com/initrm/sentiment-
analysis"><i class="fab fa-github"></i></a>
                </li>

            </ul>
        </div>
    </div>
</nav>
```

Conclusioni

La realizzazione del progetto è stata suddivisa in diverse fasi, come prima ho analizzato la richiesta e scelto le tecnologie adatte allo sviluppo di quest'ultima.
Ho scelto Bootstrap come libreria per lo stile in quanto molto popolare e semplice da utilizzare in modo tale da avere un design accattivante e responsivo unito alla facilità di sviluppo per concentrarmi più sul funzionamento effettivo del sito.
Ho scelto jQuery come libreria per il supporto dello sviluppo del codice in quanto semplifica molto lo sviluppo di script JS lato client, specialmente nella manipolazione del DOM.
La prima cosa che ho iniziato a sviluppare è stato il server, ho definito le route necessarie e ho iniziato a lavorarci studiando anche le fonti esterne come, ad esempio, il funzionamento della libreria utilizzata per il classificatore ed il funzionamento delle API di Twitter.
Sono poi passato alla progettazione delle pagine e alla creazione di queste ultime, ho scelto di utilizzare un Template Engine (Handlebars) al posto di scrivere pagine in "plain" html in modo tale da riutilizzare alcuni componenti come la Navbar e il layout di base rendendo anche le pagine più modulari e creando file più snelli.
Ho scritto poi i vari script lato client per interagire con il server attraverso chiamate AJAX (e quindi in maniera asincrona).
Come ultima cosa ho collegato e sistemato tutti componenti, diciamo in una fase di "rifinitura", assicurandomi di risolvere tutti i piccoli errori di cui non mi ero accorto durante la fase di sviluppo più "grossolano".
Un'interessante miglioria che mi è venuta in mente durante lo sviluppo e che potrebbe essere implementata in una ipotetica versione 2 potrebbe essere quella di rendere l'applicazione disponibile a tutti online dove, gli utenti loggati, potranno trainare il proprio classificatore e salvare lo stato, ad esempio, all'interno di un database, per poi ritrovarlo in una futura sessione e continuare da dove erano rimasti.

L'intero progetto è disponibile su GitHub: https://github.com/initrm/sentiment-analysis