**Web Platform per il training di modelli di NLP: DOCUMENTO TECNICO**

Introduzione:

Questo progetto rappresenta una Web Platform realizzata in Python che offre diversi servizi, tra cui la gestione di Intents, Entities e Datasets salvati come collezioni di documenti in un Database NoSQL (MongoDB), l’addestramento, il download, la cancellazione di tre modelli di NLP (Natural Language Processing), così come la visualizzazione dei risultati degli addestramenti e il loro testing.

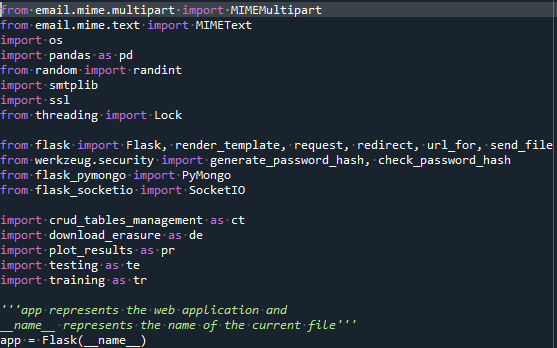
(Si ricorda che l’elenco completo delle librerie utilizzate per questo progetto è presente nel file requirements.txt)

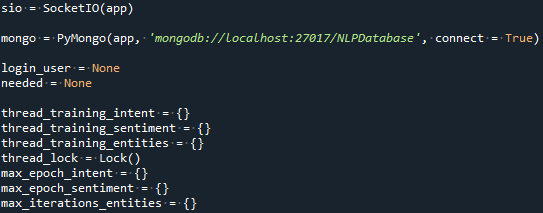
Implementazione del server:

Il server è realizzato in Python utilizzando:

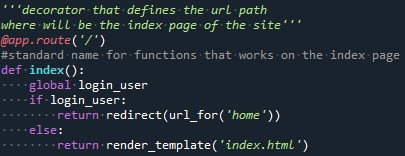
* Flask come WSGI Web App Framework, modulo Python che permette di sviluppare applicazioni web;
* Flask-PyMongo strettamente integrato con Flask e PyMongo; quest’ultimo rappresenta il Driver MongoDB ufficiale per Python che permette la gestione di Database NoSQL;
* Flask-SocketIO che fornisce alle applicazioni Flask l’accesso alle comunicazioni bidirezionali a bassa latenza tra il client e il server.

Per cui l’implementazione del server, cioè la gestione delle varie pagine che compongono il sito, è contenuta nei file **main.py**, **crud\_tables\_management.py**, **download\_erasure.py**, **plot\_results.py**, **testing.py**, **training.py**.



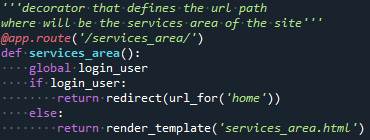


**index**



La funzione index gestisce la pagina che rappresenta l’indice del sito. In questa funzione e in ogni funzione dopo questa viene controllato se è stato eseguito o meno il login alla piattaforma, per evitare che utenti con intenzioni malevole possano entrare e accedere ugualmente ai servizi che questo sito offre.

**services\_area**



La funzione services\_area gestisce la pagina dove, a partire da essa, è possibile giungere alle pagine dove effettuare il login alla piattaforma o la registrazione.

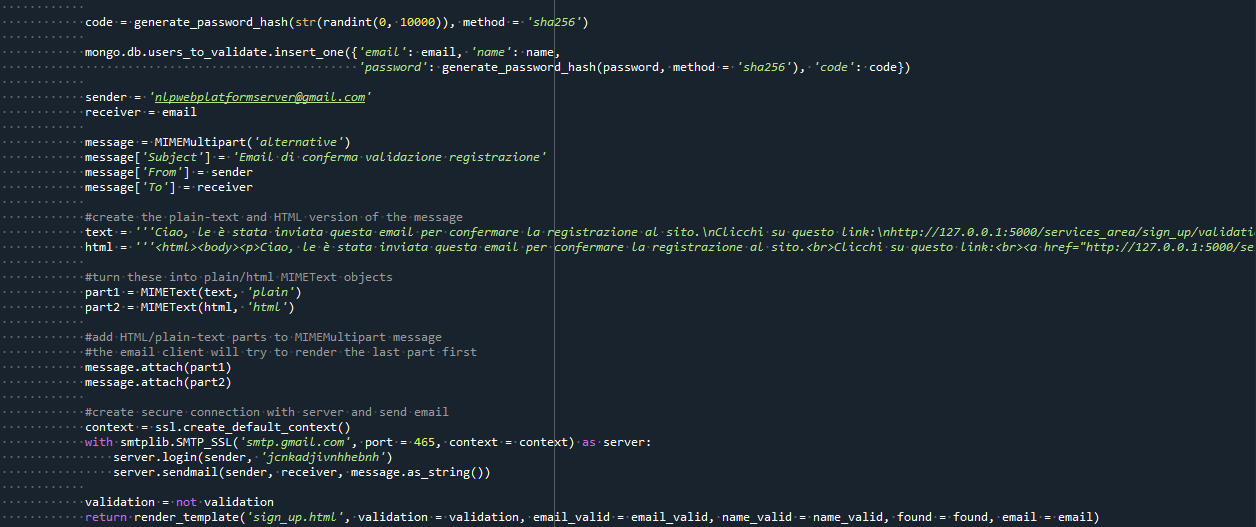
**login**



La funzione login gestisce la pagina dove è possibile effettuare il login al sito. Se sono state inserite delle credenziali, allora, la funzione controlla se nella collezione users del database MongoDB è effettivamente presente un utente con tali credenziali. In caso non sia stato trovato alcun utente, verrà mostrato un messaggio di errore nella pagina di login.

**sign\_up**

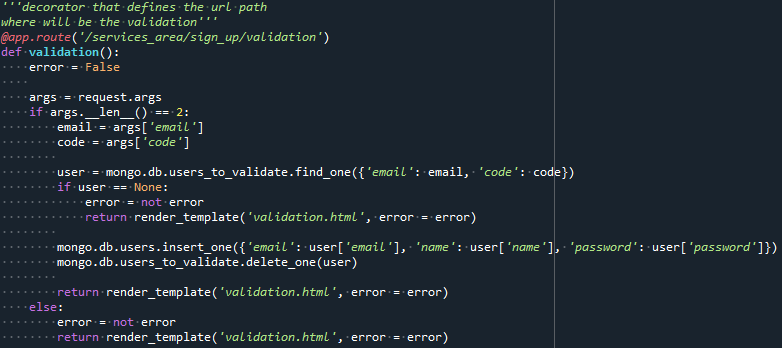






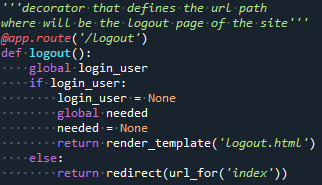
La funzione sign\_up gestisce la pagina dove è possibile registrarsi al sito. Se sono state inserite delle credenziali, la funzione controlla se un utente con tali credenziali è presente nelle collezioni users e users\_to\_validate del database MongoDB. Se trovato, allora un utente ha già richiesto di registrarsi o si è già registrato, quindi verrà visualizzato un messaggio di errore nella pagina. Altrimenti un utente verrà registrato: la password che ha inserito l’utente verrà crittografata con la funzione hash crittografica sha256; successivamente verrà inviata una email di conferma all’email che ha inserito l’utente utilizzando le funzioni delle librerie email, smtplib e ssl. Queste ultime librerie sono necessarie per l’invio sicuro delle email.

**validation**



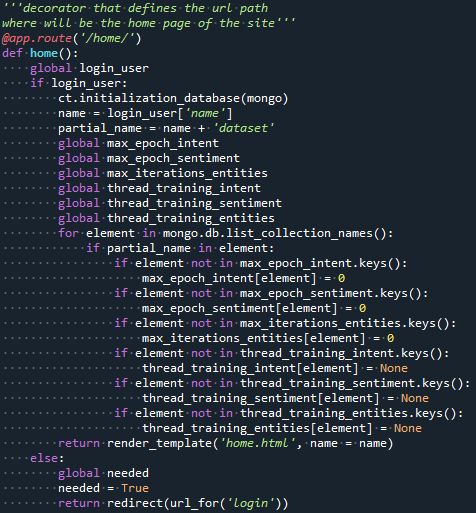
La funzione validation gestisce la pagina per la validazione al sito. Seguendo il link inviato tramite email si verrà indirizzati a questa pagina, dove avverrà la registrazione: si inserirà l’utente da validare nella collezione users, da users\_to\_validate, e si eliminerà conseguentemente l’utente da users\_to\_validate. Se l’utente non è presente in users\_to\_validate verrà visualizzato un messaggio di errore.

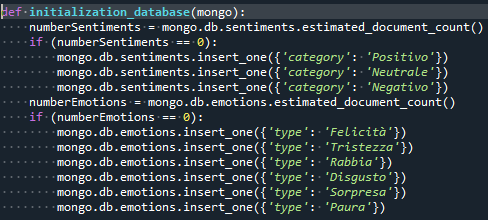
**logout**



La funzione logout effettua il logout dell’utente dalla piattaforma.

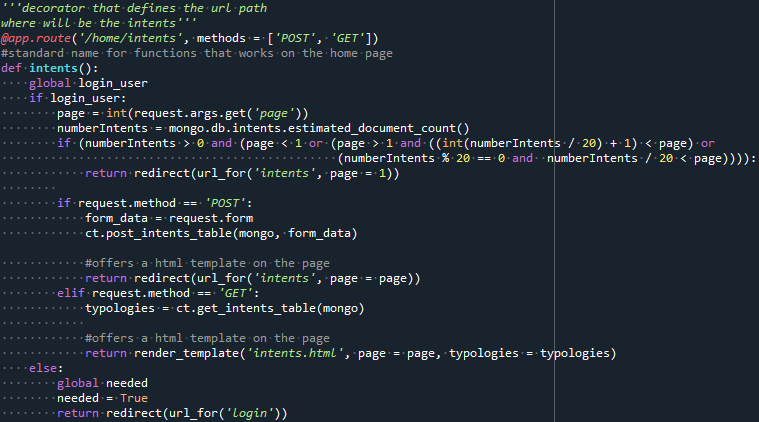
**home**

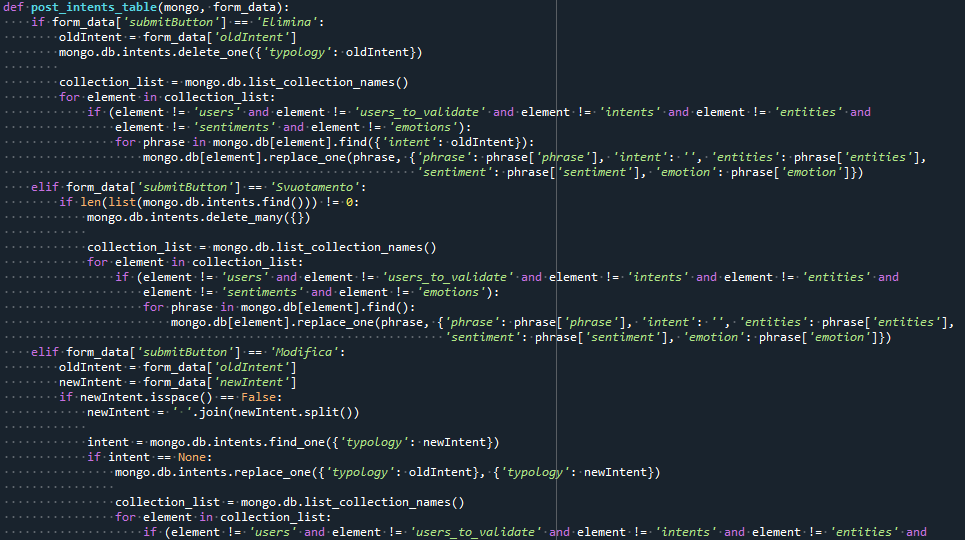


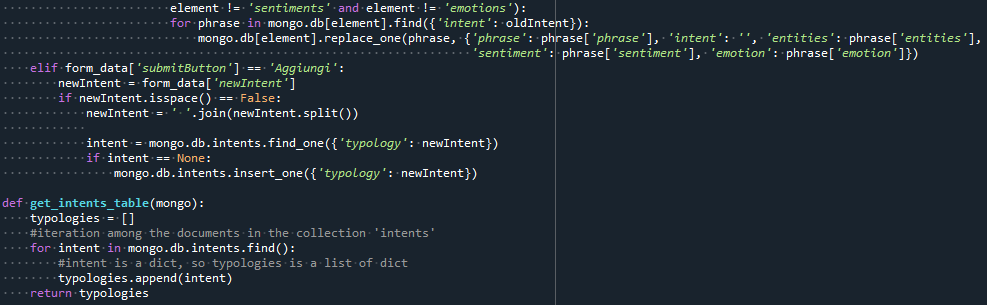


La funzione home gestisce la pagina che rappresenta l’homepage del sito. Questa funzione richiama la funzione initialization\_database del file **crud\_tables\_management.py** per inizializzare le collezioni sentiments ed emotions del database MongoDB, in caso queste collezioni siano vuote.

**intents**

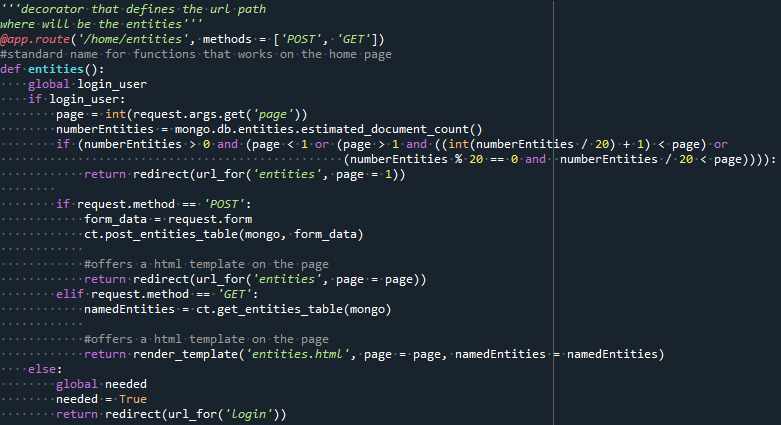


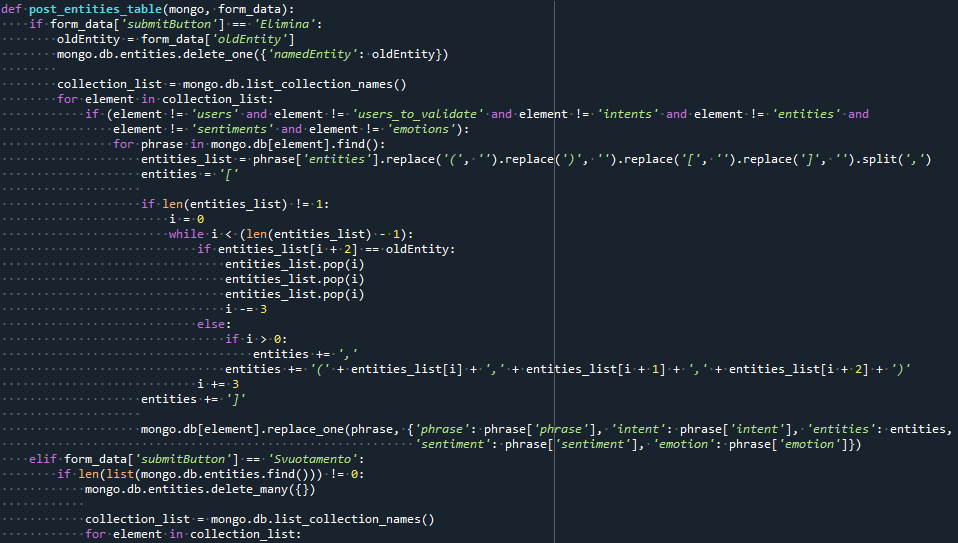


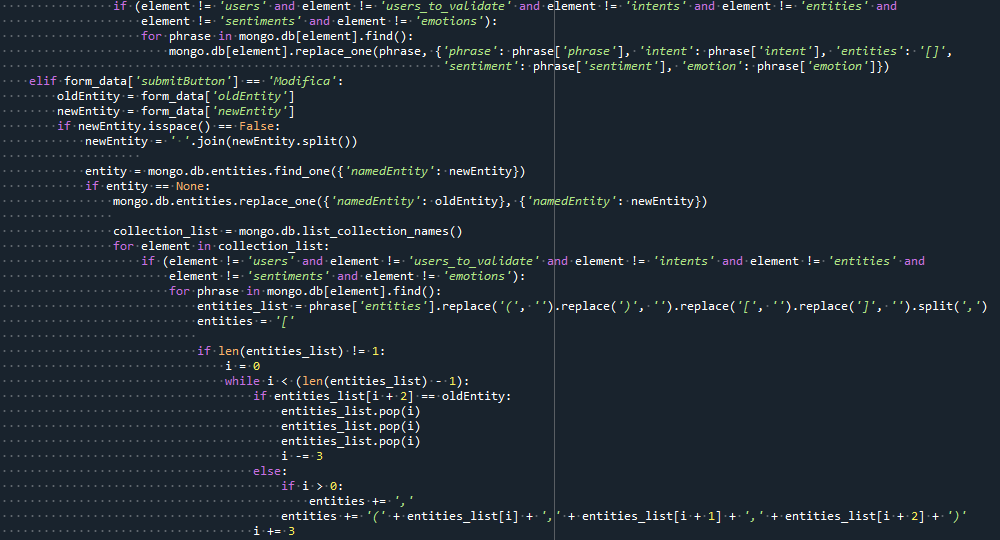


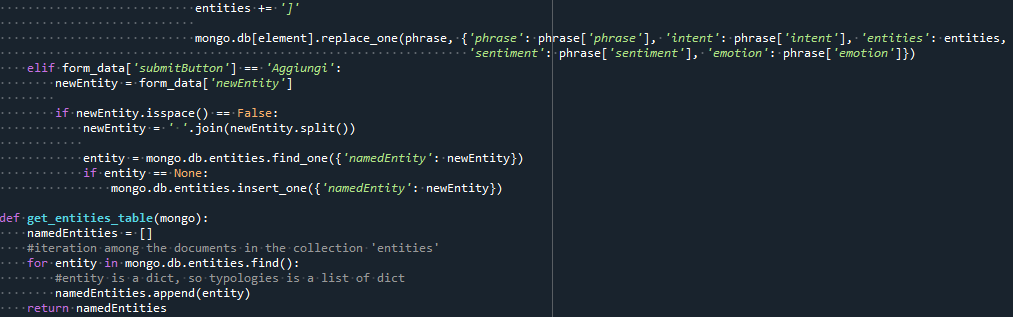
La funzione intents controlla la pagina dove è possibile gestire la collezione intents del database MongoDB. In questa funzione, così come nelle funzioni entities e training\_phrases, viene gestita efficientemente la paginazione. La gestione vera e propria degli intents, però, avviene nelle funzioni post\_intents\_table e get\_intents\_table, presenti nel file **crud\_tables\_management.py**. Infatti, tramite la funzione post\_intents\_table è possibile inserire, modificare, cancellare (uno o tutti) intents. Conseguentemente, se viene cancellato o modificato un intent, verrà rimosso dalle training phrases dei datasets in cui è presente, per evitare errori di qualsiasi tipo, così come per le entities. Inoltre non è possibile inserire o modificare intents con un nome uguale a quello di un altro intent presente nella collezione, ed è possibile inserire intents solo a partire da questi caratteri: [a-zA-Z0-9àèÈéÉìòù]. Queste osservazioni sono valide anche per le entities e per le training phrases, tranne per il fatto che queste ultime possono essere inserite solo a partire da questi caratteri: [a-zA-Z0-9àèÈéÉìòù .:,;!?].

**entities**



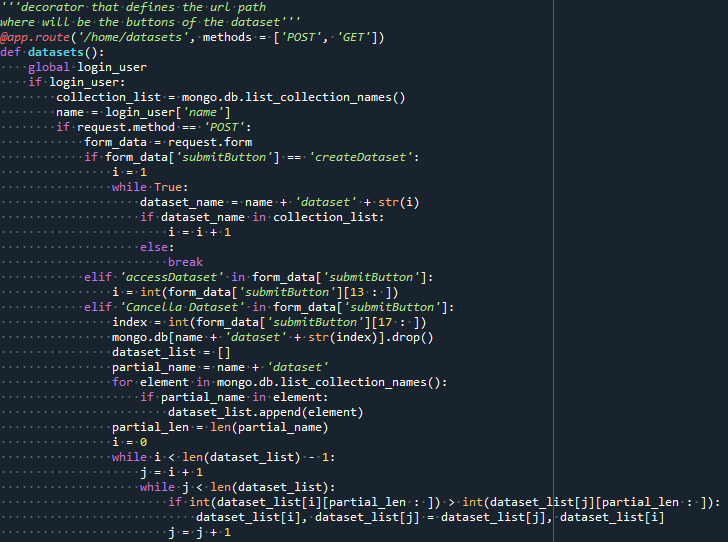


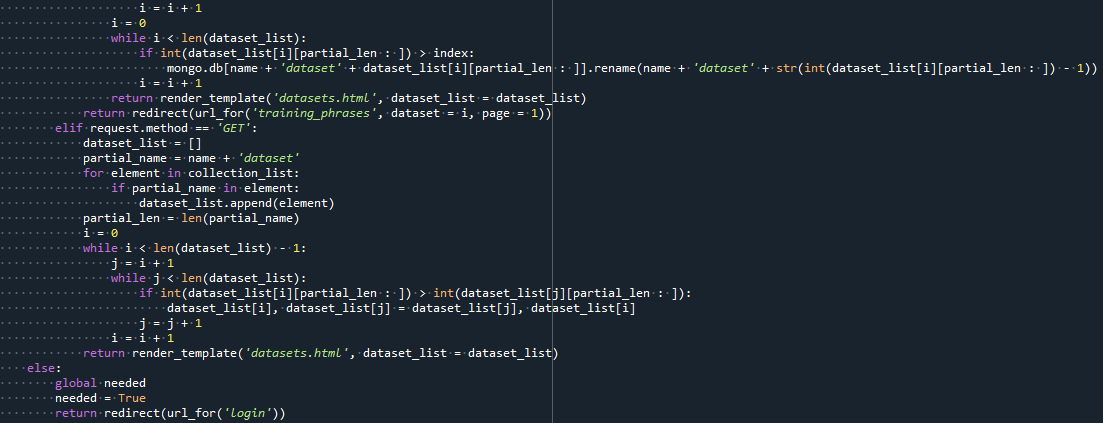




La funzione entities controlla la pagina dove è possibile gestire la collezione entities del database MongoDB. Così come per la funzione intents descritta precedentemente, la gestione vera e propria delle entities avviene nel file **crud\_tables\_management.py** tramite le funzioni post\_entities\_table e get\_entities\_table. Infatti, tramite la funzione post\_entities\_table è possibile inserire, modificare, cancellare (uno o tutti) entities.

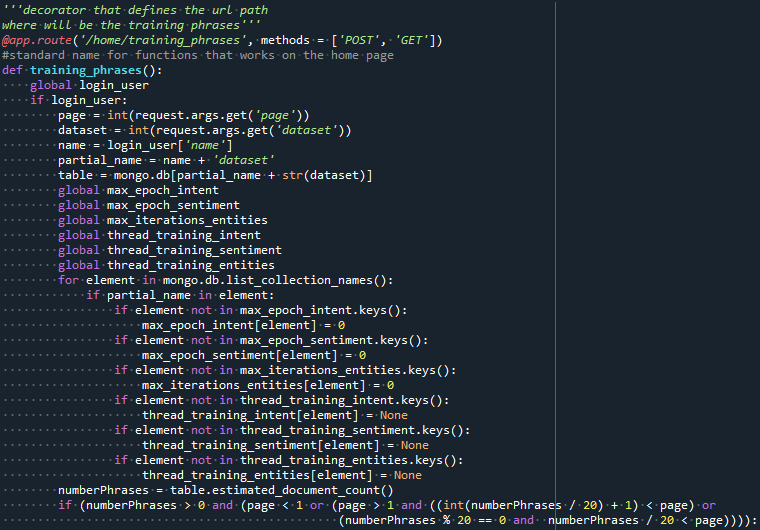
**datasets**

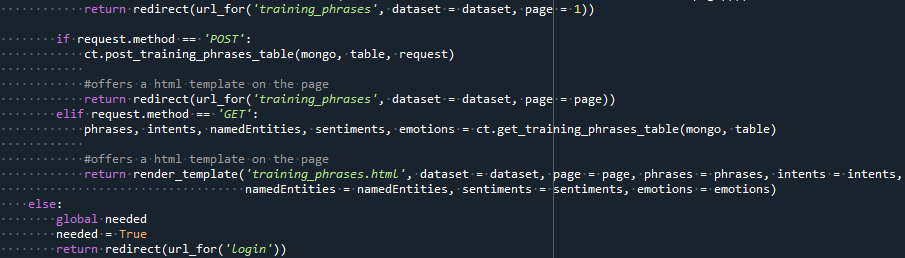
****



La funzione datasets controlla la pagina dove è possibile gestire i datasets disponibili per ogni utente, cioè è possibile creare nuovi datasets e accedere o cancellare datasets creati in precedenza. Tali datasets saranno disponibili solo per l’utente che li ha creati.

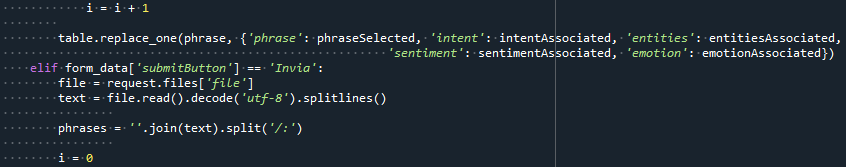
**training\_phrases**

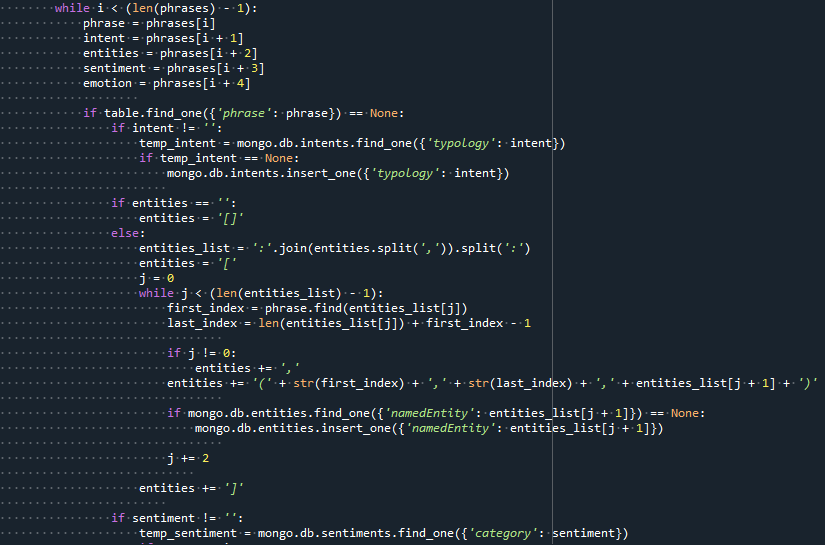


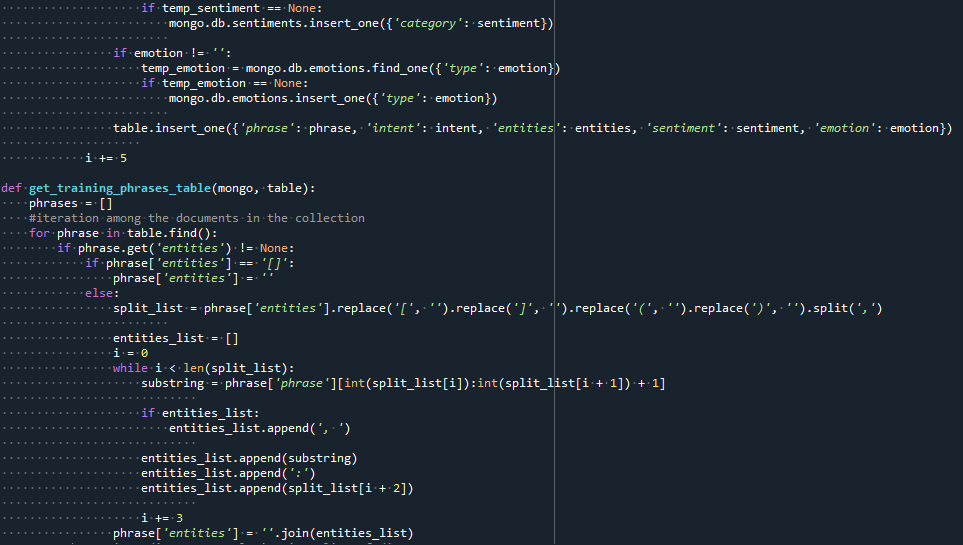


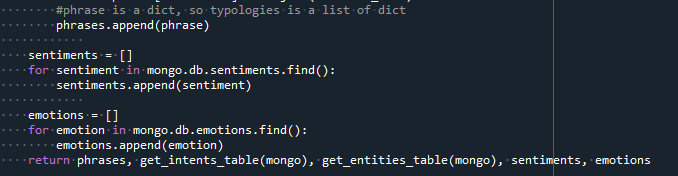






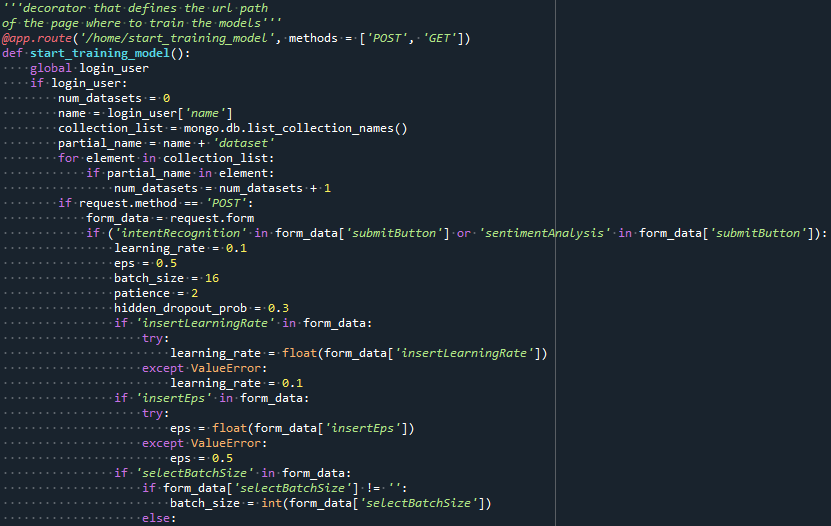


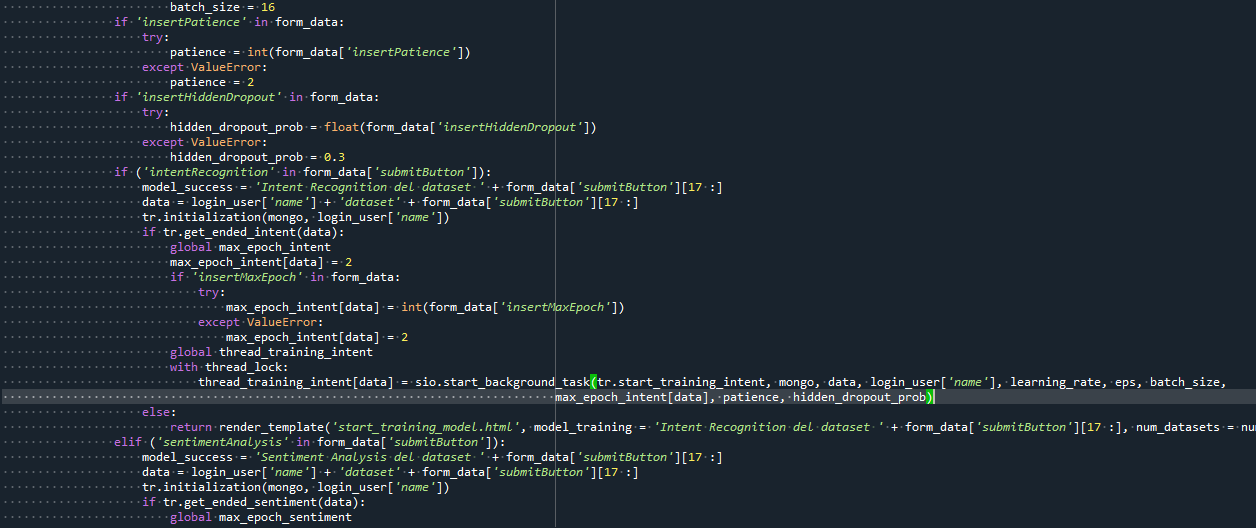




La funzione training\_phrases controlla la pagina dove è possibile gestire i dataset associati a ogni utente. Così come per le funzioni intents e entities descritte prima, la gestione vera e propria delle training\_phrases avviene nel file **crud\_tables\_management.py** tramite le funzioni post\_training\_phrases\_table e get\_training\_phrases\_table. In post\_training\_phrases\_table è possibile inserire, modificare, cancellare (uno o tutti) e annotare frasi di training per un dataset per volta.

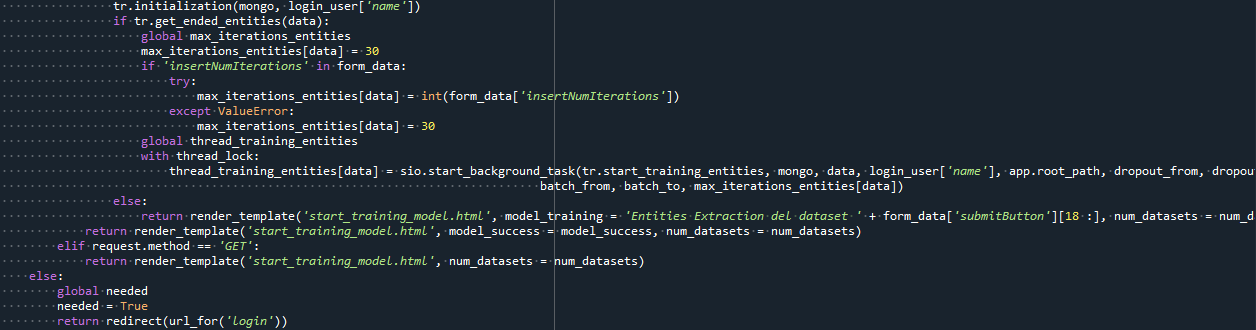
**start\_training\_model**





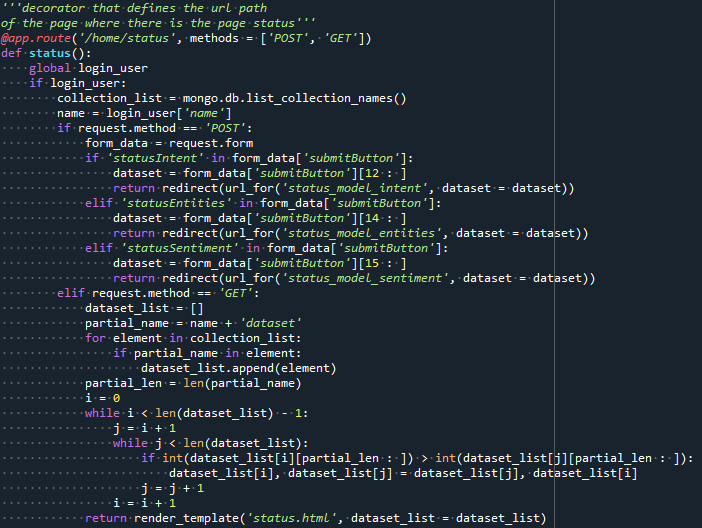


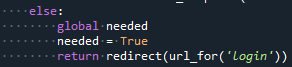


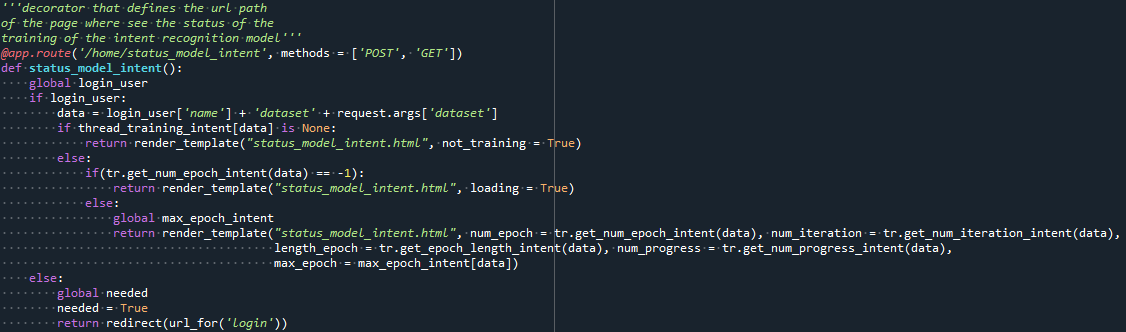


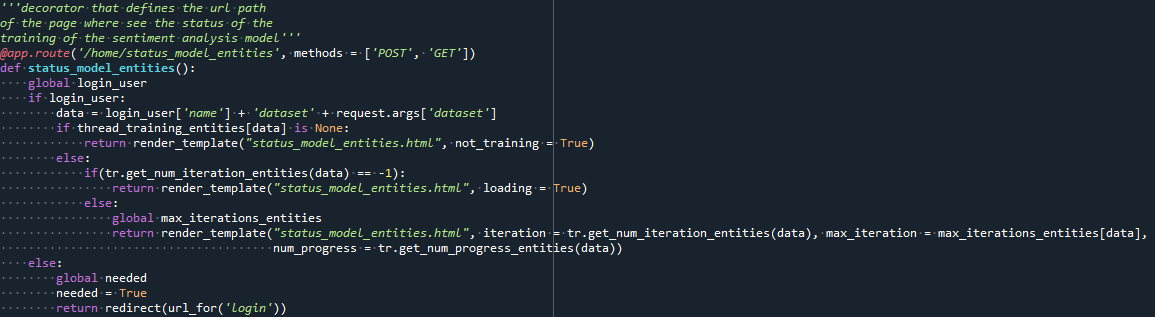
La funzione start\_training\_model permette di effettuare l’addestramento dei modelli attualmente presenti, cioè di IR, SA e EE. L’addestramento verrà spiegato meglio in delle sezioni successive di questo documento.

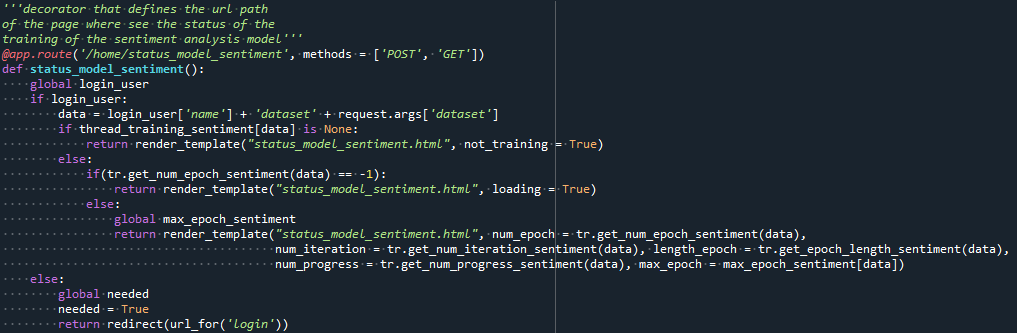
**status**

****

****

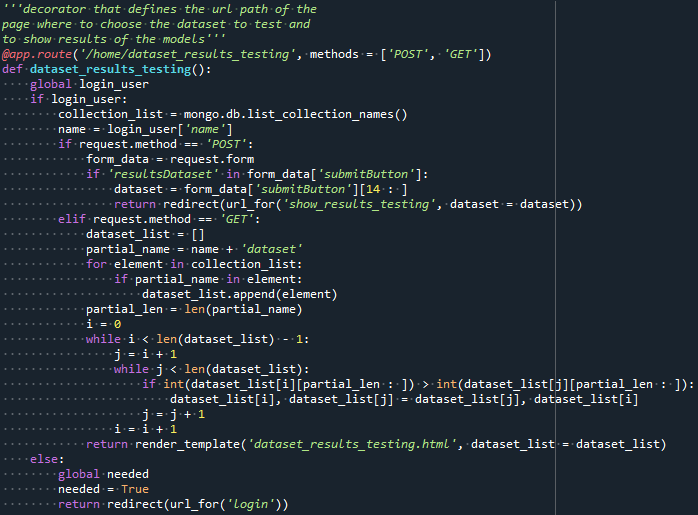
****

****

****

La funzione status gestisce la pagina dove a partire da essa si giunge alle vere e proprie pagine di status degli addestramenti, cioè le pagine gestite dalle funzioni status\_model\_intent, status\_model\_entities e status\_model\_sentiment. Se un addestramento è in esecuzione, che sia di IR, di SA, e/o di EE, vengono utilizzati gli iperparametri dell’addestramento per generare progress bar per visualizzare in tempo reale lo status dello/degli addestramento/i.

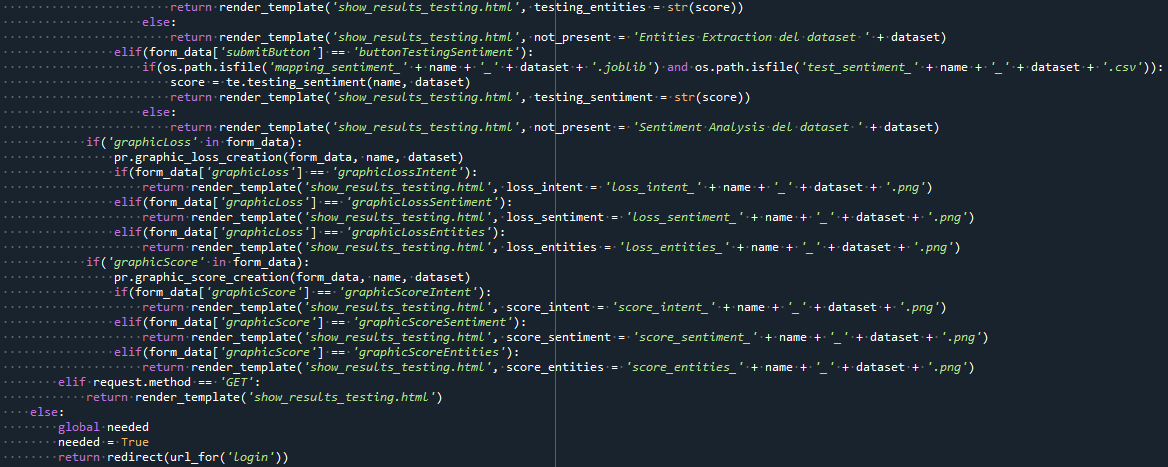
**dataset\_results\_testing**

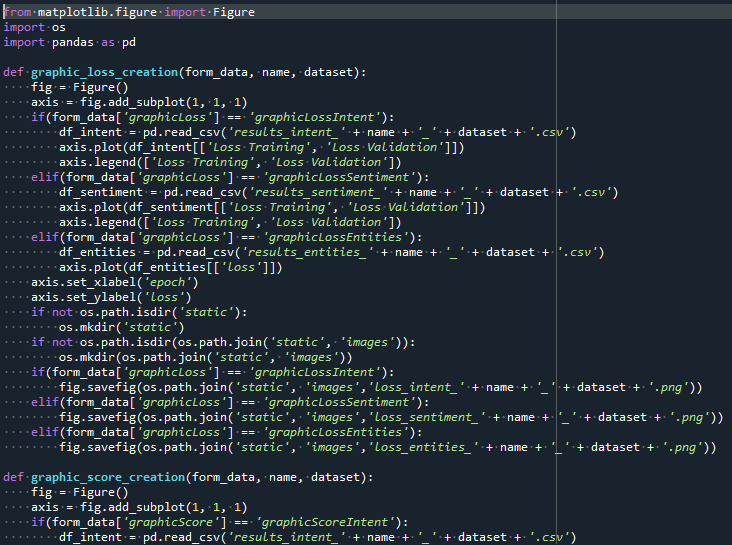
****

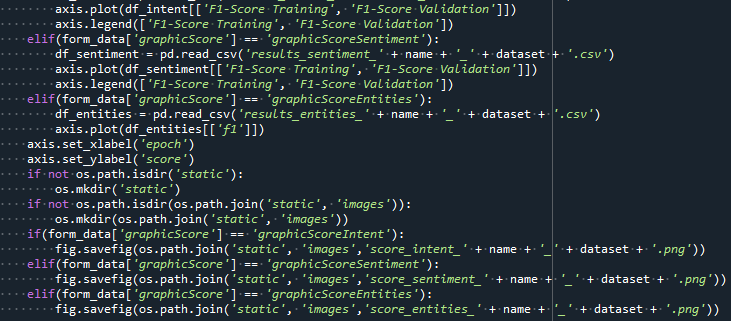
La funzione dataset\_results\_testing gestisce la pagina dove, a partire da essa, è possibile effettuare il testing e visualizzare i risultati per i modelli ottenuti dopo averli addestrati su singoli Dataset.

**show\_results\_testing**



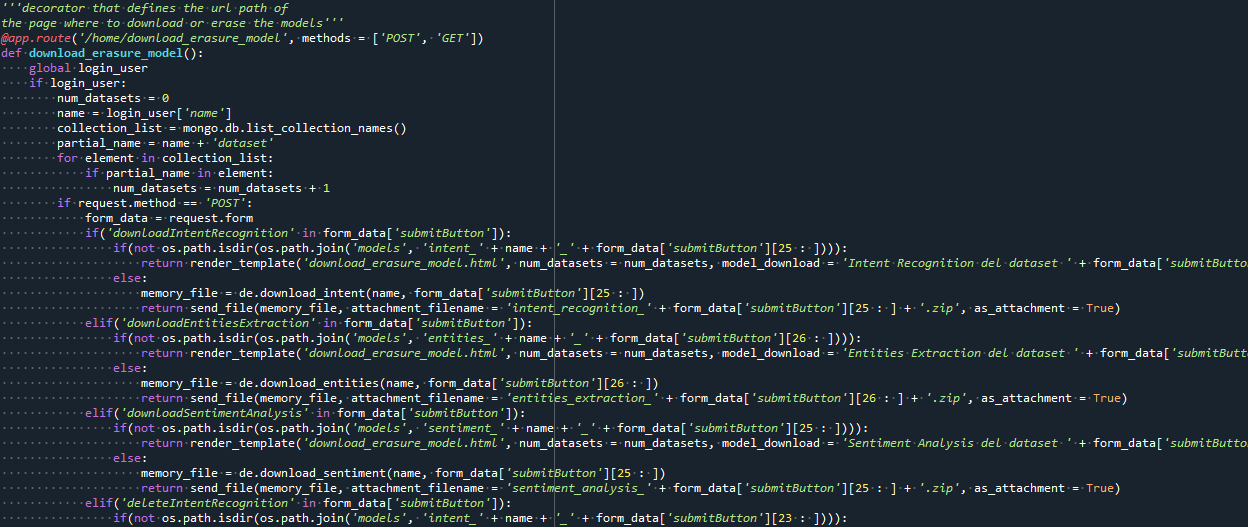


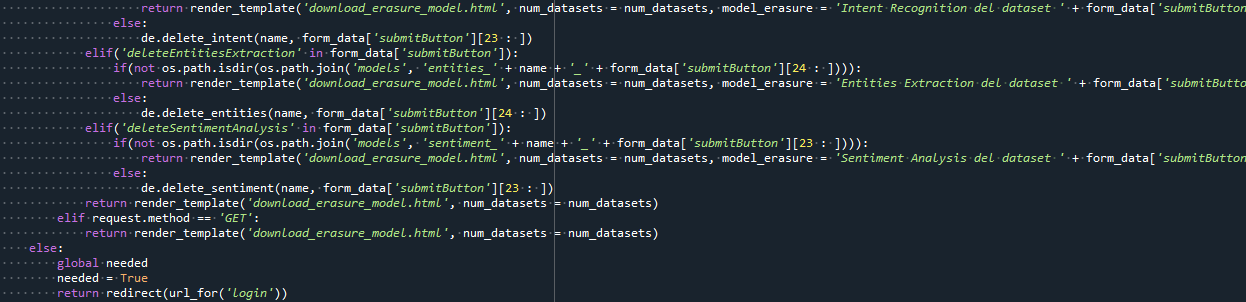


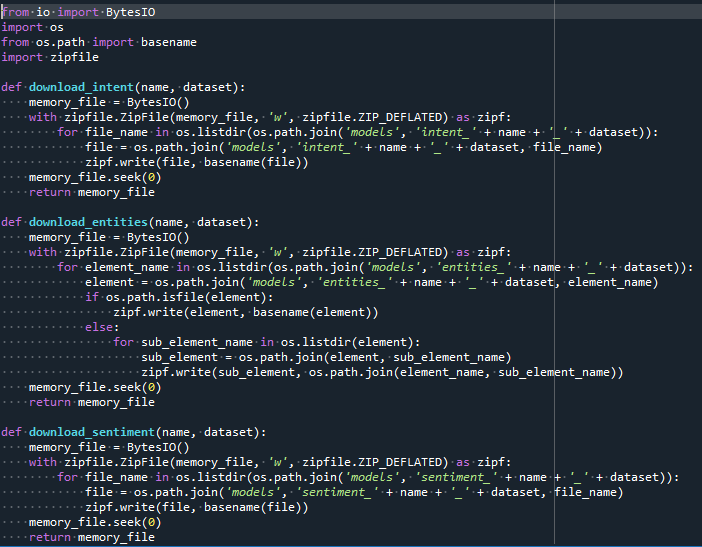


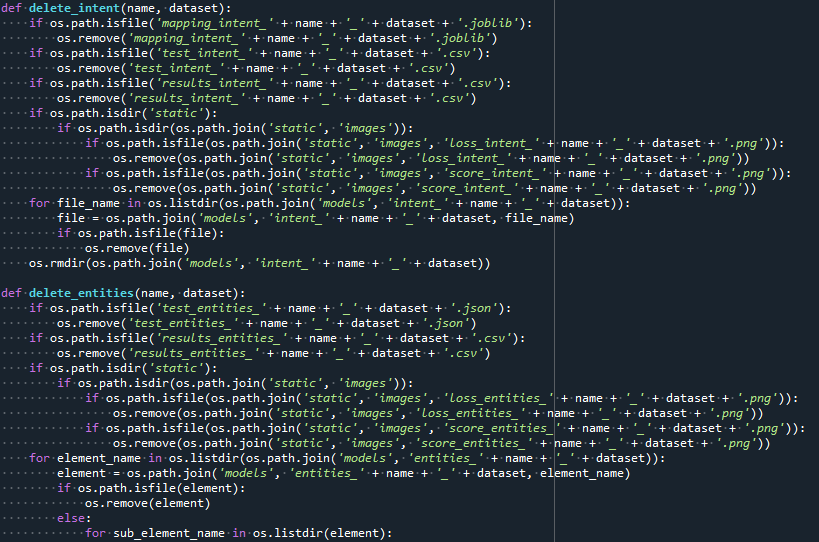
La funzione show\_results\_testing gestisce la pagina dove è possibile effettuare il testing dei modelli addestrati e mostrare i risultati degli addestramenti eseguiti. Il testing verrà spiegato meglio in delle sezioni successive di questo documento. La generazione dei grafici di Loss e F1-Score viene effettuata mediante le funzioni graphic\_loss\_creation e graphic\_score\_creation presenti nel file **plot\_results.py**, che utilizzano le libreriematplotlib, os e pandas.

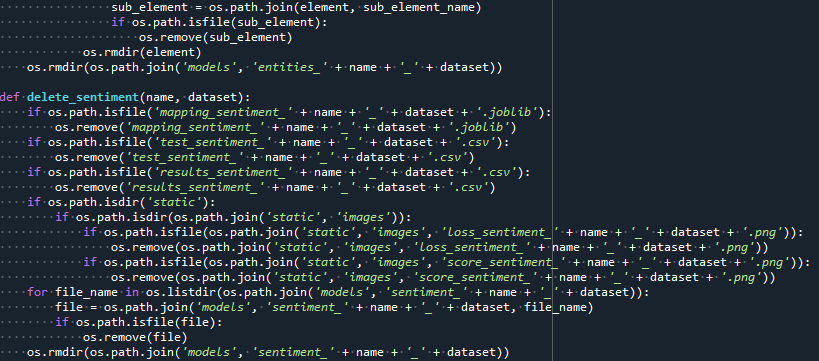
**download\_erasure\_model**











La funzione download\_erasure\_model gestisce la pagina dove è possibile scaricare o cancellare i modelli addestrati, se presenti. Queste operazioni vengono svolte tramite le funzioni del file **download\_erasure.py**, utilizzando le librerieio, os e zipfile.

Addestramento dei modelli:

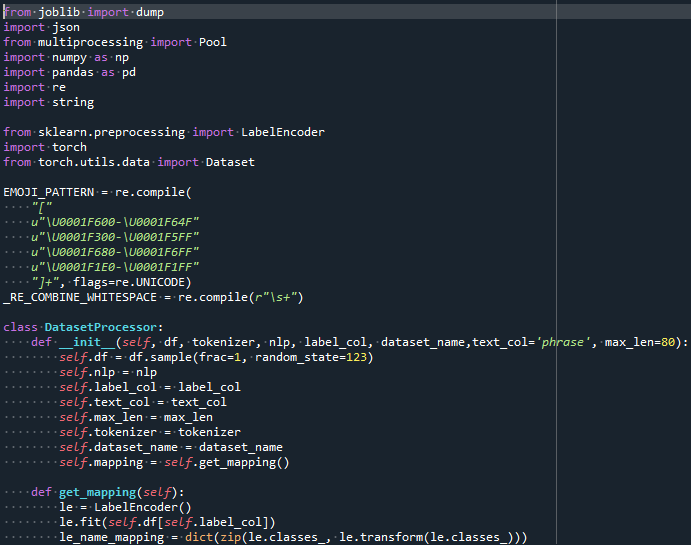
Il training di questi modelli è effettuato su thread paralleli (quindi è possibile avviare l’addestramento dei modelli in sequenza) in Python utilizzando principalmente:

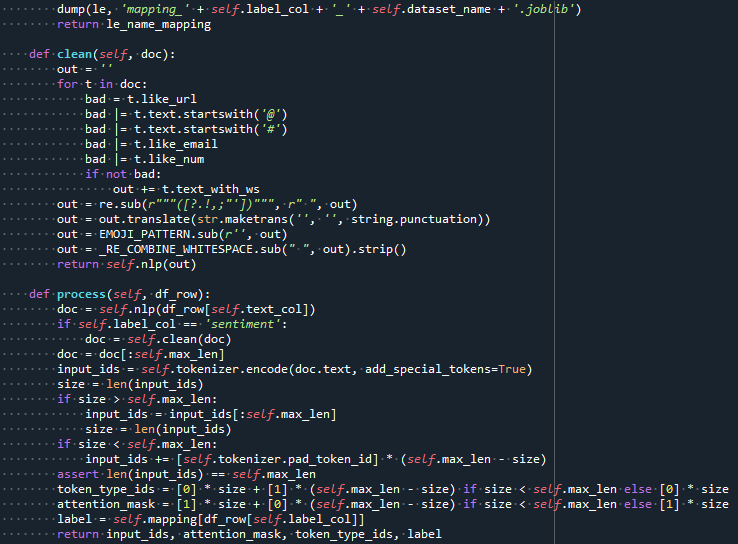
* spaCy, che rappresenta una libreria open-source progettata per lo sviluppo di applicazioni NLP;
* PyTorch, come pacchetto Python che fornisce Tensor Computation e Deep Neural Networks costruite su un sistema di autograd basato su nastro;
* Transformers che fornisce architetture di uso generale per NLU (Natural Language Understanding) e NLG (Natural Language Generation).

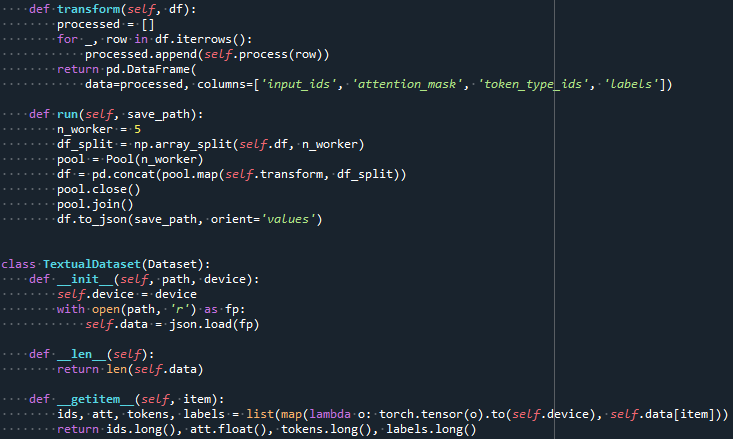
Intent Recognition e Sentiment Analysis:

L’addestramento dei modelli di IR e di SA avviene attraverso due classi e diversi file elencati di seguito:

**dataset.py:**

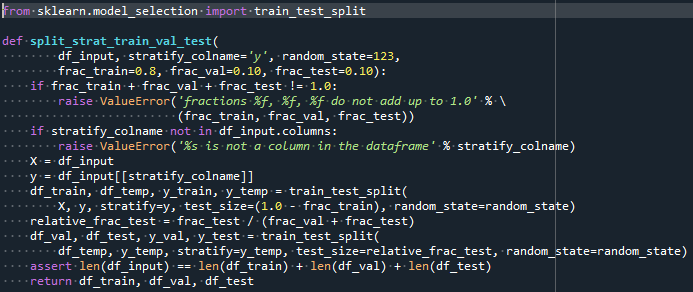






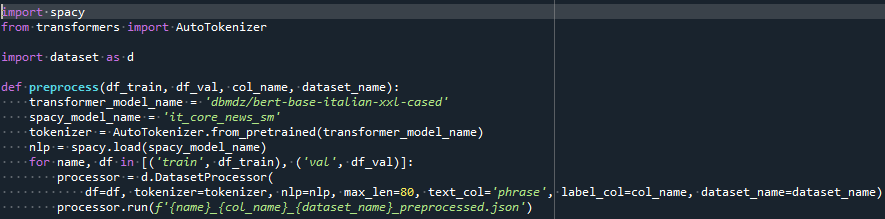
Nel file **dataset.py** ci sono le classi DatasetProcessor e TextualDataset che permettono il processing del dataset per l’addestramento dei modelli di IR (Intent Recognition) e di SA (Sentiment Analysis). Queste due classi vengono utilizzate principalmente in **preprocessing.py** e in **bert\_training.py**.

**splitting.py:**

****

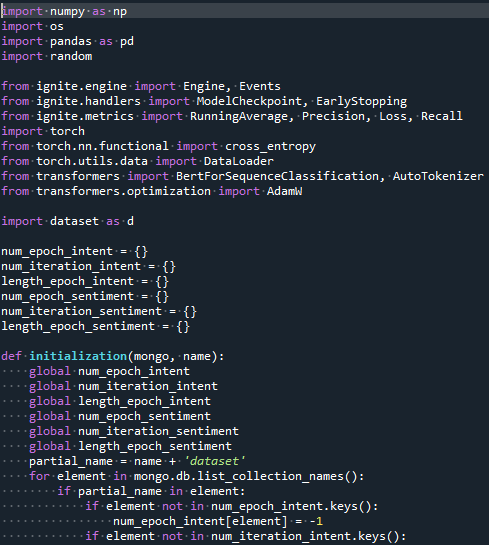
Nel file **splitting.py** avviene il processo dello splitting, cioè la divisione del dataset in frasi.

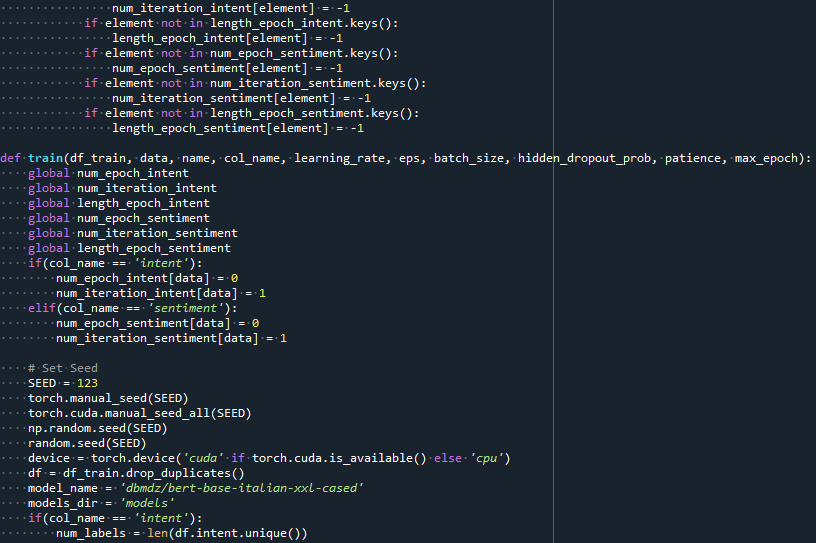
**preprocessing.py:**

****

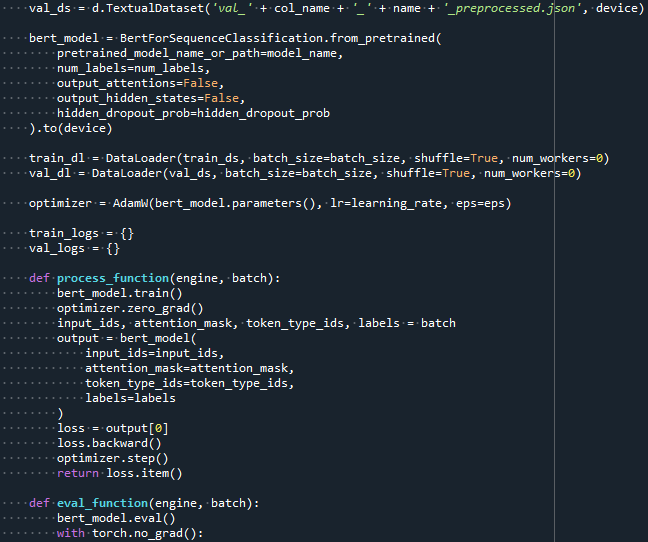
Nel file **preprocessing.py** avviene il preprocessamento del dataset, applicato trasversalmente sia al corpus sul quale è svolto l'addestramento, sia agli input da sottoporre al modello addestrato, il cui obiettivo è semplificare e trasformare il dato testuale a livello di singola parola o token*.*

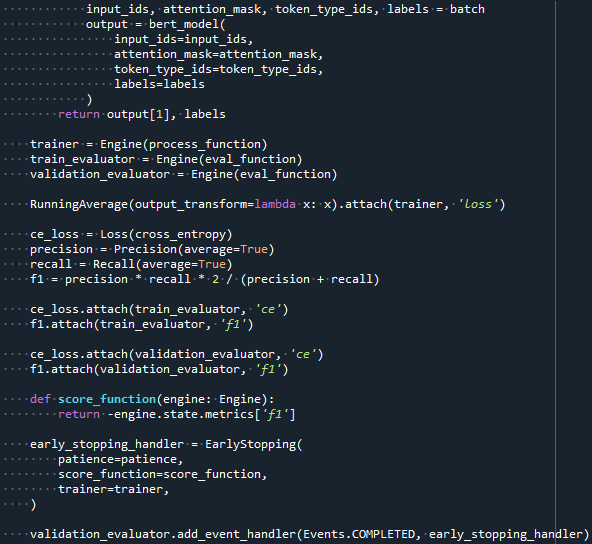
**bert\_training.py:**

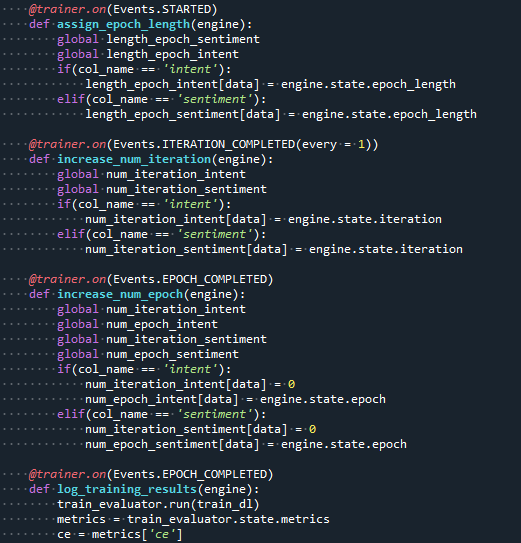
****

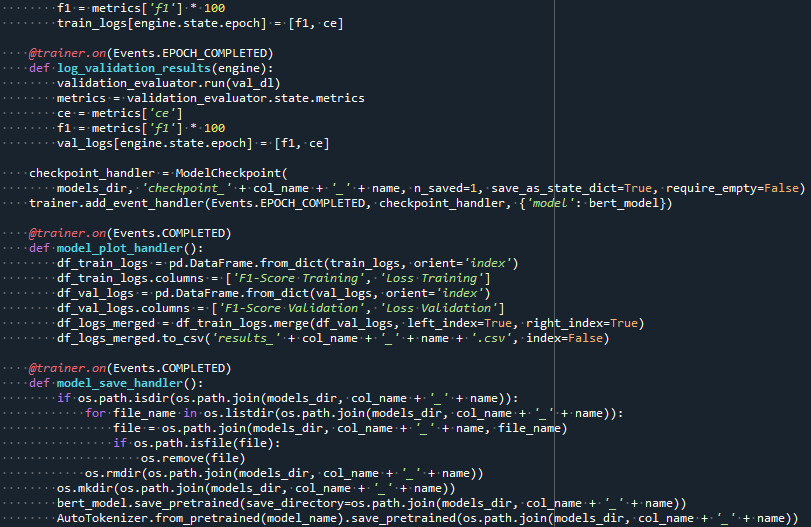
****

****

****

****

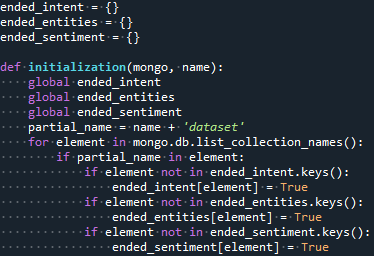
****

****

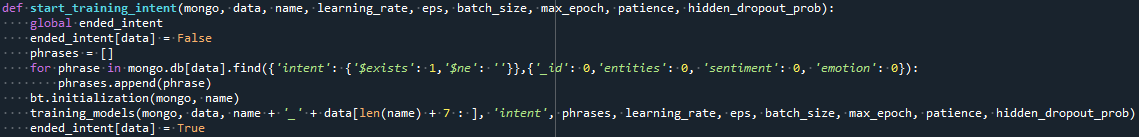
****

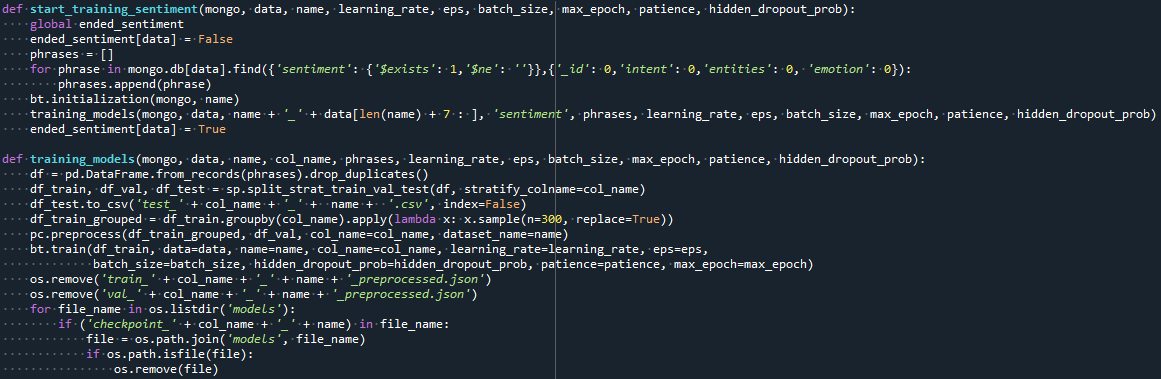
Nel file **bert\_training.py** la funzione principale è la funzione di train, che utilizza un algoritmo open source attualmente utilizzato da Google nel proprio motore di ricerca, cioè BERT(Bidirectional Encoder Representations from Transformers). BERT può comprendere parti di parlato come pronomi, fornendo un meccanismo per analizzare in maniera più efficace query scritte in linguaggio naturale.

**training.py**

****

(funzione che viene richiamata ogni volta che deve essere addestrato un modello)



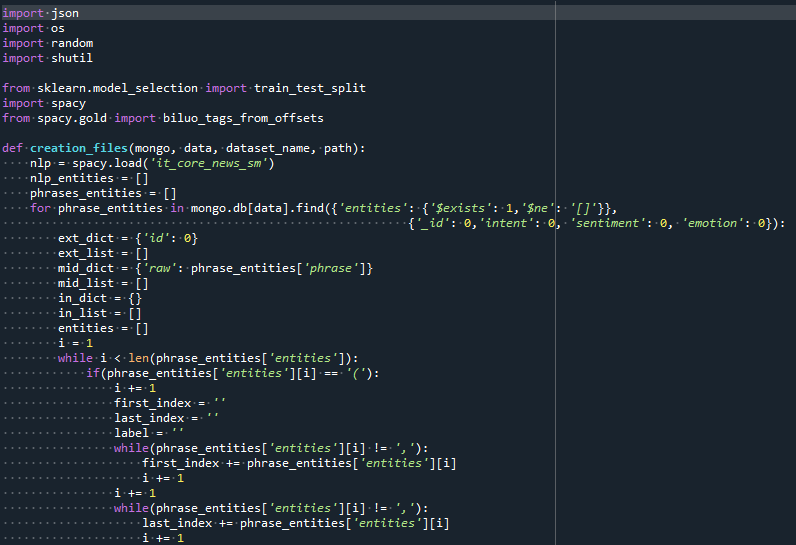


Nel file **training.py**, tra tutte le funzioni, ci sono le funzioni che iniziano l’addestramento del modello di IR o di SA su thread paralleli, cioè start\_training\_intent e start\_training\_sentiment. Entrambe richiamano la funzione training\_models, tra i cui parametri si ritrovano gli iperparametri utilizzati per l’addestramento del modello, cioè: learning rate, eps, batch size, max epoch, patience, hidden dropout. Essa si occupa dello splitting, del preprocessing e del bert training.

Entities Extraction:

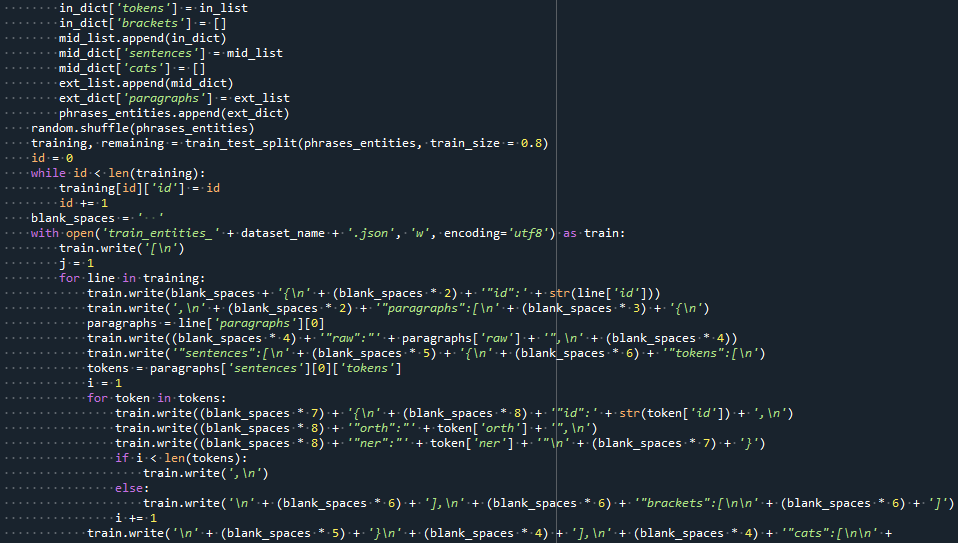
L’addestramento del modello di EE avviene attraverso due file elencati di seguito:

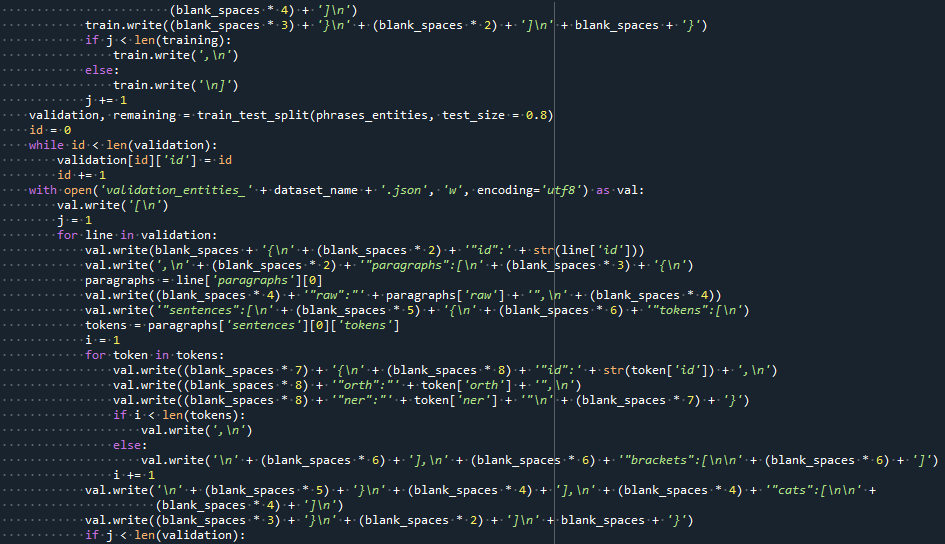
**preparation.py**

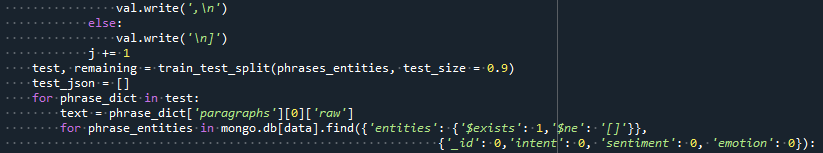


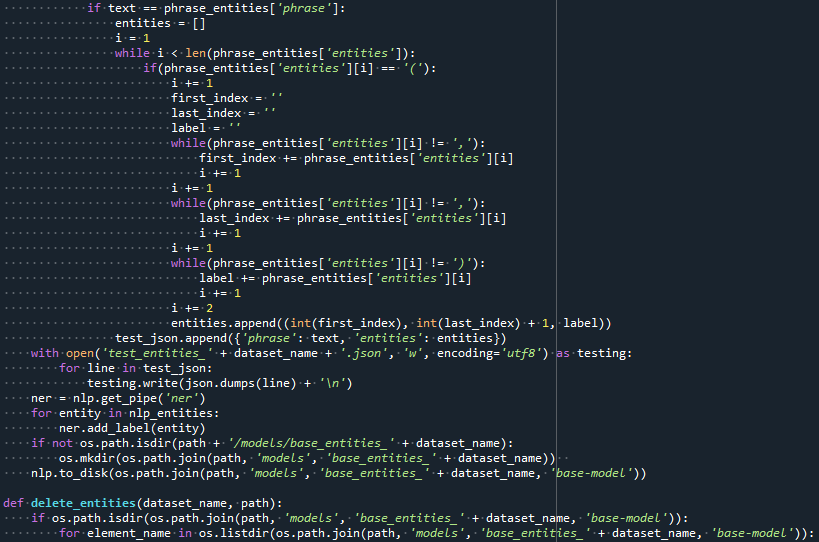


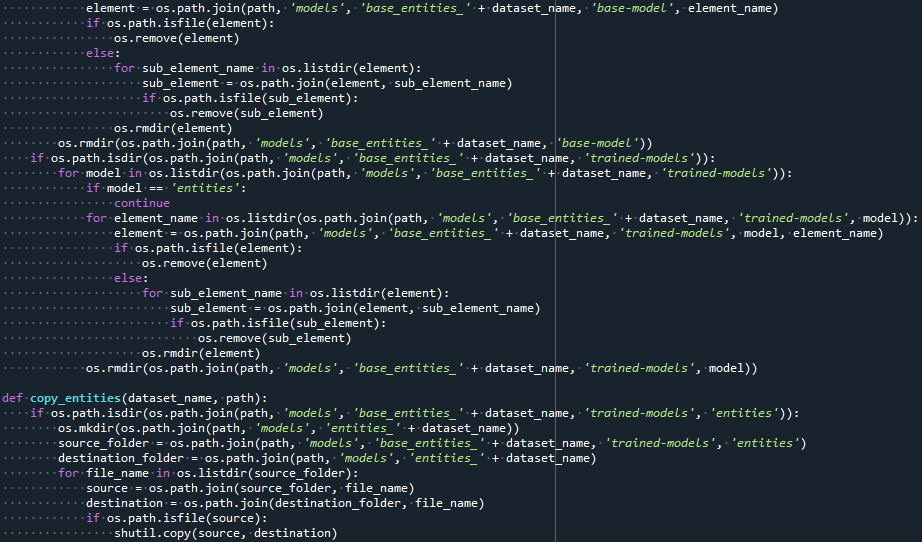


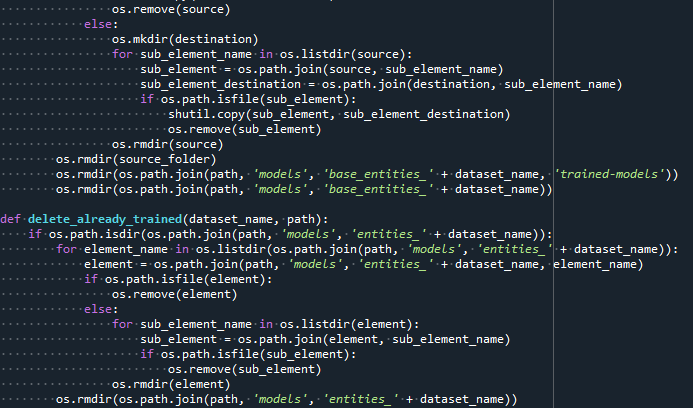






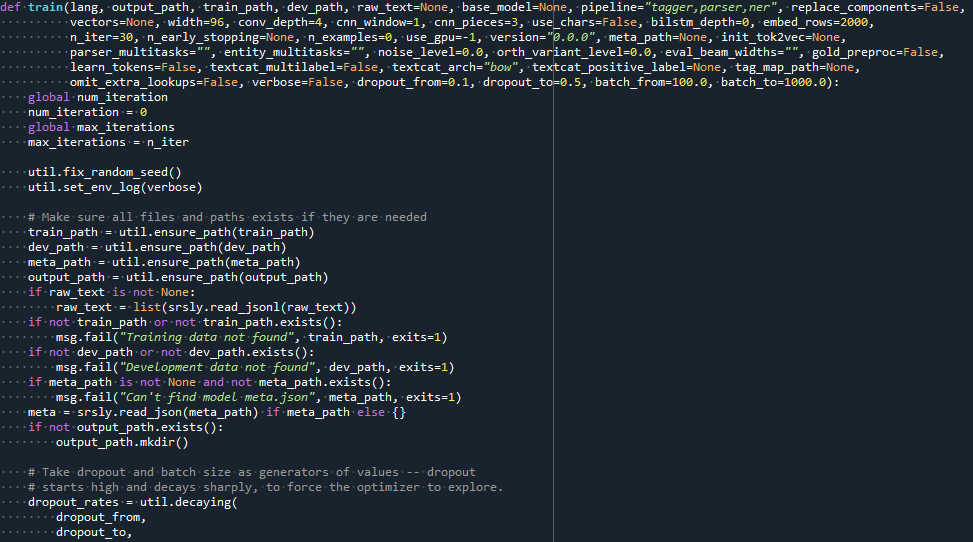






Nel file **preparation.py** ci sono le funzioni che si occupano di creare i file .json di training, validation e testing effettuando lo splitting del dataset, di cancellare i modelli addestrati lasciando il modello migliore chiamato entities, di copiare la cartella relativa al modello migliore all’interno della cartella models.

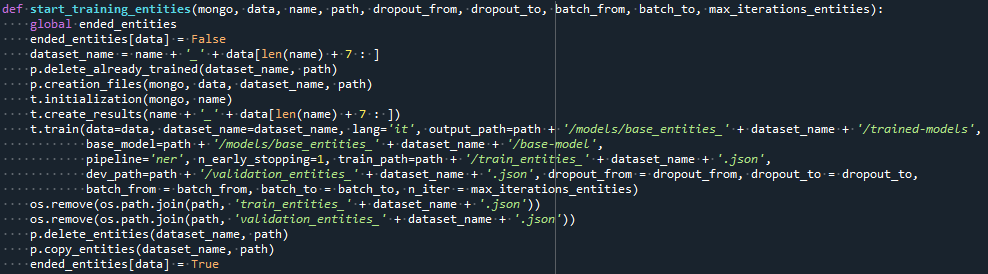
**training\_ee.py**



(immagine della funzione principale del training dell’Entities Extraction; si è deciso di inserire unicamente questa immagine per via della grandezza del file)

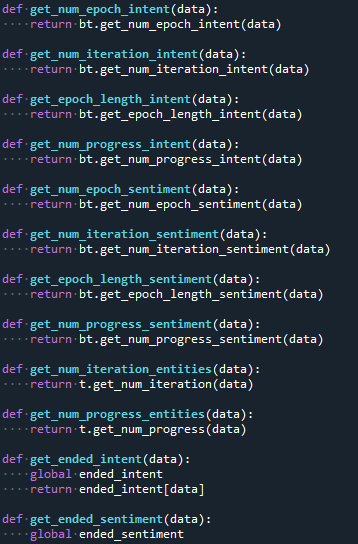
Nel file **training\_ee.py** è presente la funzione train, che permette di addestrare effettivamente il modello di EE. Essa si occupa di individuare e classificare le named entities menzionate in un dataset organizzato in categorie.

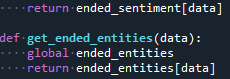
**training.py**



Nel file **training.py** c’è anche la funzione che inizia l’addestramento del modello di EE su un thread parallelo, cioè start\_training\_entities. Tra i parametri della funzione citata precedentemente si ritrovano gli iperparametri utilizzati per l’addestramento del modello, cioè: dropout from, dropout to, batch from, batch to, max iterations entities. Essa si occupa quindi di addestrare il modello utilizzando la funzione train presente nel file **training\_ee.py**.

Infine, nel file **training.py** ci sono anche funzioni get che permettono di visualizzare lo stato dell’addestramento in esecuzione nella pagina status relativa all’addestramento:





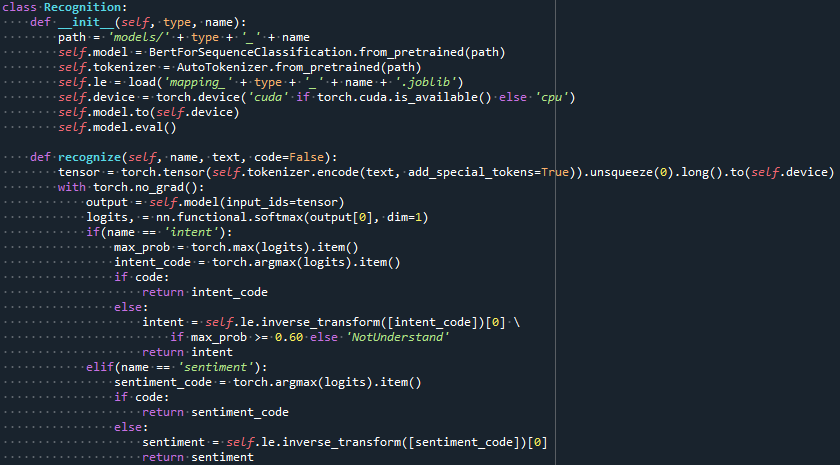
Testing dei modelli:

Le librerie rilevanti per il testing dei modelli sono spaCy, PyTorch e Transformers, descritte già precedentemente in questo documento.

Intent Recognition e Sentiment Analysis:

Il testing dei modelli di IR e di SA avviene attraverso una classe, cioè Recognition, e i file **inference.py** e **testing.py**:

**inference.py:**

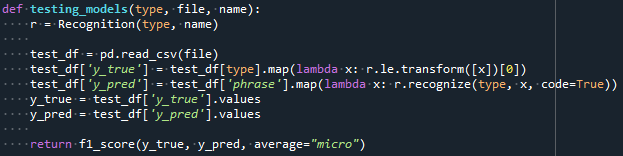


In **inference.py**, come accennato precedentemente, è presente la classe Recognition, che permette di effettuare la funzione di Natural Language Inference (NLI). Tale classe viene utilizzata per il testing dei modelli di IR e di SA nel file **testing.py**.

**testing.py:**





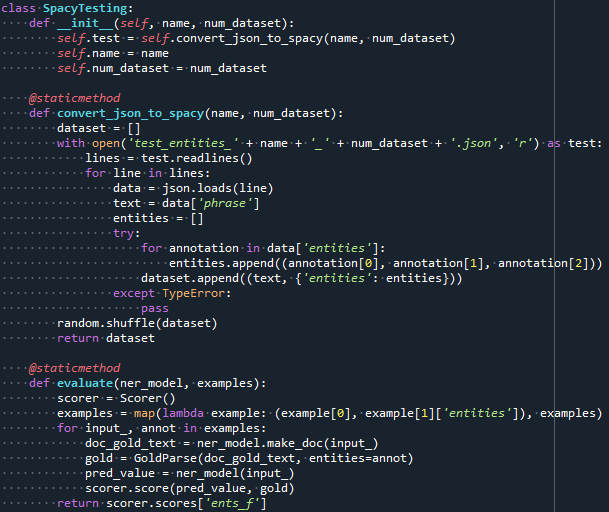


Nel file **testing.py** viene effettuato il testing del modello, rispettivamente modello di Intent Recognition (IR) tramite la funzione testing\_intent e modello di Sentiment Analysis (SA) tramite la funzione testing\_sentiment. Entrambe queste funzioni richiamano la funzione testing\_models, che ha come parametri name che rappresenta se il testing è per l’intent o per il sentiment, e file che rappresenta il file passato per il testing.

Entities Extraction:

Il testing del modello di EE avviene attraverso una classe, cioè SpacyTesting, e i file **inference.py** e **testing.py**:

**inference.py:**





In **inference.py** è presente la classe SpacyTesting, che, come la classe Recognition, permette di effettuare la funzione di Natural Language Inference (NLI). Tale classe viene utilizzata per il testing del modello di EE nel file **testing.py**.

**testing.py:**



Nel file **testing.py** viene effettuato il testing del modello di EE tramite la funzione testing\_entities. Questa funzione richiama la funzione run presente nel file **inference.py** definita nella classe SpacyTesting.