## Implementation:

### Initial settings:

The program now consists of six folders, plus the requirements.txt file, which contains all the dependencies; manage.py which is an indispensable configuration file for Django, and html\_frontend.html which contains the HTML, javascript, and CSS code to add to the page of Kiroteca and which will be analyzed later.

One folder contains Tint, which as we will see, is used for the grammatical analysis of the text, another contains Mallet, used for the LDA; the third, djangoTesi, contains the project settings files while the backend contains all the code to manage and analyze the data.

The last two folders are not present in the repository because inserted in gitignore, they must be created, one with the name "Credentials" and must contain all the access files used by the program, while the other "Media" containing three subfolders: "Video", “Audio” and “YoutubeCaptions” which will be used as folders to temporarily store data still to be processed, if any files remain inside them it is because they have caused some kind of error and must be analyzed.

To run the program, after downloading it and creating a python virtual environment (tested on python 3.6, 3.7, 3.8) and installing all the dependencies contained in requirements.txt on it, the database must be set. To do this, download and install Postgresql, then add a server named "djangotesi" and as user "admin", these names can be changed in the file "djangoTesi/settings.py" in the DATABASE section; for the password instead, you need to create a "Credentials/db\_password.txt" file containing only the latter.

For the other credentials you will need to: create a “Credentials/django\_password.txt” file containing a string of about twenty random characters including letters, numbers, and special characters; after which you need to add the "Credentials/vimeo\_credential.txt" file containing a token that can be used to authenticate to the Unipv Vimeo account; also you need to enter the credentials to be used to download subtitles from YouTube such as "Credentials/client\_secret\_youtube.json" and those to access Google Cloud "Credentials/credentials\_googleCloud.json". The latter two must be generated from the Google Cloud console by creating a project and granting it permissions to use the youtube API and the Speech-to-text service.

The database and Django credentials are used in the “djangoTesi/settings.py” file while the other three in the “backend / AggregateData / parseVideo.py” file in the “AnalyzeVideo” class.

Being both Tint and Mallet written in java it is essential to have an updated version of the same on the machine.

In Google Cloud you need to create a bucket called “video-lessons” in the “europe-west1” region; in the class "Speech2text" I left the method "create\_bucket()" which does it automatically, but you have to run it once.

To create the database tables you need to run the commands "python manage.py makemigrations" and "python manage.py migrate", after which you need to create a superuser with "python manage.py createsuperuser" indicate the required data such as username and password that then they will be used to access the server administration panel.

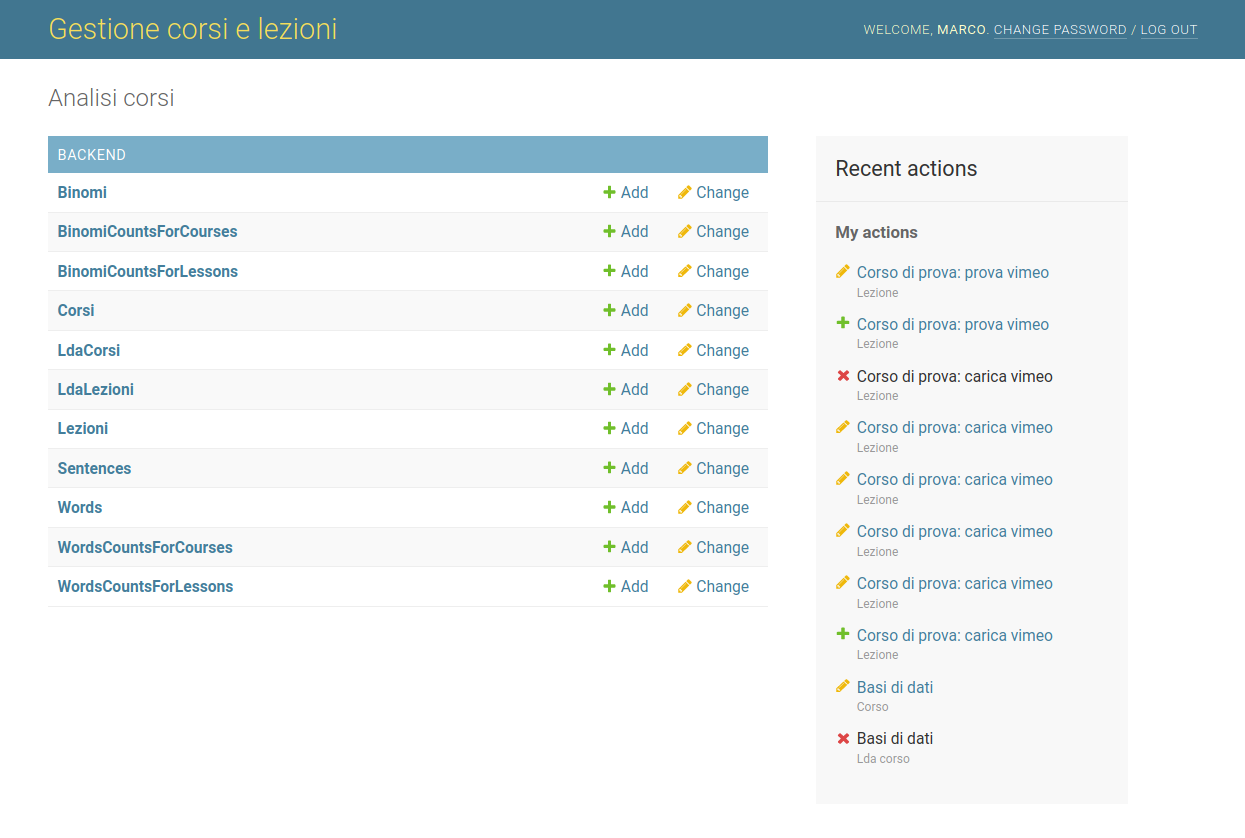
To launch the server just run "python manage.py runserver"

### Program structure:

The program consists of three fundamental parts, the backand server which includes the API and the database, its administrative site, and the page on Kiroteca.

To better understand the various parts of the program and their operation, I proceed with an example of a use case.

First, you need to enter the data of the courses and lessons in the backend, to do this we go to the server/admin site and log in with the data entered when creating the Django superuser.



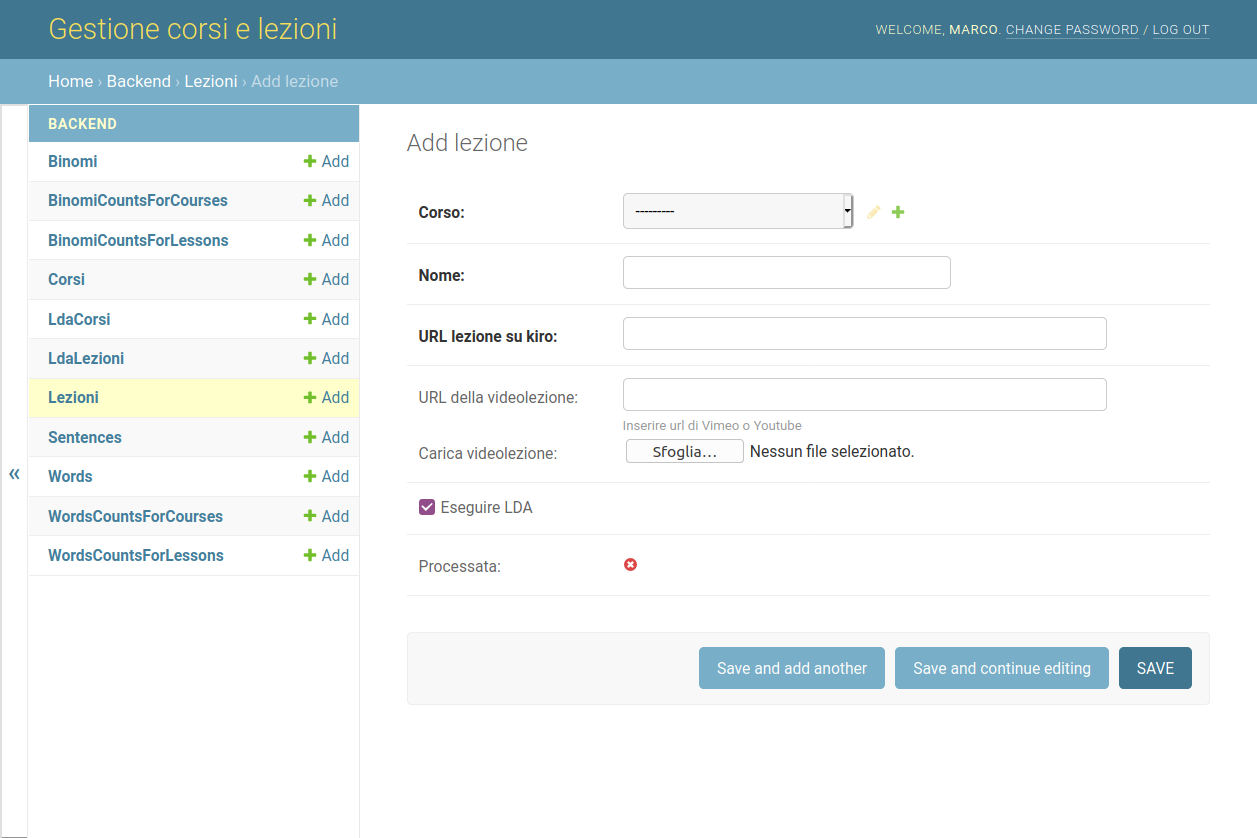
The administrative site of the backend will appear.

This site is automatically created by Django and managed in the "backend/admin.py" file,

At this point you need to add a new course, then go to "Corsi" and press "add"

Immagine che contiene testo

Descrizione generata automaticamenteFill in the form and save the course, but not checking the "process course" box, in this way we will only add the data entered in the database. After that we proceed to add the various lessons, to do so we need to go to the lessons section and press add.



Among the various data requested, we note Kiro's URL that contains the video lesson, which will be used to redirect students who want to view it; while the YouTube or Vimeo links are used only to download the subtitles or the video itself to be able to analyze them and will not be accessible by students.

You can see on admin.py that the form has been modified, first I check the data entered, in fact, you can either directly upload the video of the lesson or insert a link, which however must be from YouTube or Vimeo.  
You can also note that after saving the lesson, the “AnalyzeVideo(obj).start()” function is started, where obj represents the data entered in the form, therefore the lesson itself.  
Let's move on to the heart of the project to “backend/AggregateData/parseVideo.py” and we notice that the function just launched has started a new thread that will analyze the inserted lesson.  
First of all, it distinguishes how the lesson has been entered, whether via file or link and, in that case, whether by Vimeo or YouTube; it then starts the appropriate process to extrapolate the data, in this case:

Local video:

If a file has been downloaded directly, it will be temporarily saved in the "Media/Video" folder. To process it, the audio is first extracted from the video and saved in the "flac" format in the "Media/Audio" folder. the “Video2audio” class takes care of it, which uses the Python moviepy library; the audio, to be analyzed on Google Cloud, must necessarily be in flac format and with a single track.

At this point, the audio track must be uploaded to a Google Cloud bucket. This is done through the "Speech2Text" class which also analyzes the uploaded audio, also saving the timestamps of the words and adding punctuation automatically; after analyzing it, it deletes the audio from the bucket and brings the data in a standard format that is the one obtained with the subtitles from YouTube, it is one or two words separated by a timestamp enclosed in <> with the format hh:mm:ss.fff as in the following example

*allora<00:00:19.910> eccoci<00:00:20.360> qua<00:00:20.660> a<00:00:20.750> iniziare<00:00:21.380>*

At that point it will be analyzed with a process that we will see later and eventually the generated video and audio files will be deleted.

Vimeo link:

If a Vimeo link is inserted, the video ID is first extracted and, through the VimeoDownload class, the video file is downloaded to the "Media/Video" folder, to do so it needs two calls to the Vimeo API, the first to find out all the video formats available and the related download links, then it chooses the least weighted video and downloads it.

At what point we find ourselves in the same situation as the local video and we proceed in the same way.

YouTube link:

As for YouTube, first, the video ID is extracted from the link, credentials are then generated through the youtubeCredentials class which, if not present or expired, requires a token from YouTube, is then passed to the class captionDownload that downloads the automatically generated subtitles in Italian of the videos which are finally passed to the cropCaption class which eliminates redundant or useless elements and generates the text in the standard format.  
We have chosen to download the subtitles directly and not the entire video in order to download a greater number of them per day, but the problem remains that YouTube subtitles do not have punctuation so it will not be possible to perform the LDA as we will see later, it is instead possible to indexing words as described above.

### Data structure:

To store the data produced by the program it was necessary to implement a database.  
To do this we took advantage of the automation of Django and used the file "backend/models.py".

For the data structure, first of all, a table containing the courses was thought of, which contain the name and URL of Kiro of the same; it was then necessary to add a boolean to show whether the course had already been processed as a whole or not.  
In addition, the name of the course has been made unique to allow a unique search for it by a student; if the same course is added in different years, the previous courses can either be deleted, or the year in which it was held can be added to the name.  
To make a faster query the names in the database have been indexed.  
Finally, help text and names in Italian have been added to allow for better user experience when entering data on the administrative site.

For the lessons, on the other hand, in addition to the name Kiro's link, containing the video lesson, has been added other than the course to which it belongs as a foreign key, the URL of the original video which can be null, the video as a file, it can also be null, a boolean to decide whether or not to execute the LDA on the lesson and finally, an automatically created boolean that identifies if the lesson has already been processed.  
The check if at least one of the two fields between the video URL or the video itself has been filled in is done during the validation of the form to add a lesson, as well as if the URL is from YouTube or Vimeo.  
The video is uploaded directly to the “Media/Video” folder.  
It has been specified that if the course to which a lesson belongs with a foreign key should be deleted, the lesson itself will also be deleted.  
Finally, help text and names in Italian have been added to allow for better user experience when entering data on the administrative site.

To save the keywords of each lesson, a table was created including the lemmatized words themselves, the lesson they belong to as a foreign key, and the timestamp of when it was said.  
The word has been indexed in the database to allow a faster search, it has been specified that in case of deletion of the lesson it belongs to, the tuple with the keyword must also be deleted.

For the binomials a solution similar to that for words was adopted, saving the two words that make up the binomial separately but indexing them together in the database.  
Lesson and timestamp as for words.

For the counting of words and binomials, tables have been created on purpose, in fact, although it is possible to obtain them from the binomials and words tables through specific queries it would be a slow and useless solution, it was therefore decided that once the lesson is entered words and binomials are counted in it, and once you have finished inserting the lessons in a course, those of the same are generated.

For the counting of words and binomials in the lessons the tables are very similar but, in one there is a word, while in the other a binomial; in this case, they have not been lemmatized but are used that as a reference the first case found within a lesson.  
The lesson is added as a foreign key, specifying that in case of elimination this data must also be deleted, the word count that is indexed in the database and the tf (term frequency) i.e. the number of times that word appears divided by the number of total words of the lesson.

For the counting in the courses the situation is similar, with the difference that this time the foreign key is of a course and there is no tf.

To subsequently perform the LDA of an entire course it is necessary to save the sentences of each lesson. To do this, a structure has been created including the sentence as text, the number of the sentence in the lesson, and the lesson itself as a foreign key.

Two distinct tables have been created for the LDA, one for the lessons and one for the courses but they are similar, unlike the foreign key which can be of a lesson or a course, also there is an array of strings to contain the various topics found by the LDA.  
This last field is possible only in PostgreSQL and this is one of the reasons for its choice.

BISOGNA AGGIUNGERE IDF E TF-IDF

## Implementazione:

### Settaggi iniziali

Il programma ora è composto da sei cartelle, più il file requirements.txt che racchiude tutte le dipendenze, manage.py che è un file di configurazione indispensabile per Django e html\_frontend.html che contiene il codice html, javascript e css da aggiungere alla pagina di Kiroteca e che sarà analizzato più avanti.

Le sei cartelle contengono una Tint, che come vedremo è utilizzato per l’analisi grammaticale del testo, un’ altra contiene Mallet, utilizzato per l’LDA; la terza , djangoTesi, contiene i file di settaggio del progetto mentre backend contiene tutto il codice per gestire ed analizzare i dati.  
Le ultime due cartelle non sono presenti nel repository perchè inserite nel gitignore, vanno create, una con il nome “Credentials” e dovrà contenere tutti i file di accesso utilizzati dal programma, mentre l’altra “Media” contenente tre sottocartelle: “Video”, “Audio” e “YoutubeCaptions” che saranno utilizzati come cartelle per immagazzinare temporaneamente dati ancora da processare, se rimangono dei file al loro interno è perché hanno causato qualche tipo di errore e vanno analizzati.

Per eseguire il programma, dopo averlo scaricato e creato un virtual environment di python (testato su python 3.6, 3.7, 3.8) e installato su di esso tutte le dipendenze contenute in requirements.txt bisogna settare il database. Per farlo scaricate e installate Postgresql, aggiungeteci quindi un server di nome “djangotesi” e come user “admin”, questi nomi possono essere cambiati nel file “djangoTesi/settings.py” nella sezione DATABASE; per la password invece bisogna creare un file “Credentials/db\_password.txt” contenente unicamente quest’ultima.

Per le altre credenziali bisognerà: creare un file “Credentials/django\_password.txt” contenente una stringa di una ventina caratteri casuali compresi lettere, numeri e caratteri speciali; dopo di che bisogna aggiungere il file “Credentials/vimeo\_credential.txt” contenente un token utilizzabile per autenticarsi sull’account Vimeo dell’unipv; inoltre bisogna inserire le credenziali da utilizzare per scaricare i sottotitoli da YouTube come “Credentials/client\_secret\_youtube.json” e quelle per accedere a Google Cloud “Credentials/credentials\_googleCloud.json”. Queste ultime due vanno generate dalla console di Google Cloud creando un progetto e garantendogli i permessi per utilizzare le API di youtube e del servizio Speech-to-text.

Le credenziali del database e di django sono utilizzate nel file “djangoTesi/settings.py” mentre le altre tre nel file “backend/AggregateData/parseVideo.py” nella classe “AnalyzeVideo”.

Essendo sia Tint che Mallet scritti in java è indispensabile avere una versione aggiornata dello stesso sulla macchina.

In Google Cloud bisogna creare un bucket di nome “video-lezioni” nella regione “europe-west1”; volendo nella classe “Speech2text” ho lasciato il metodo “create\_bucket()” che lo fa in automatico, ma bisogna eseguirlo una volta.

Per creare le tabelle del database bisogna lanciare i comandi “python manage.py makemigrations” e “python manage.py migrate”, dopo di che bisogna creare un superuser con “python manage.py createsuperuser” indicare i dati richiesti come username e password che poi serviranno per accedere al pannello amministratore del server.

Per lanciare il server basta eseguire “python manage.py runserver”

### Struttura del programma

Il programma è composto da tre parti fondamentali, il server di backand che comprende le API e il database, il suo sito amministrativo e la pagina su Kiroteca.

Per comprendere al meglio le varie parti del programma e il loro funzionamento procedo con un esempio di caso d’uso.

Per prima cosa bisogna inserire i dati dei corsi e delle lezioni nel backend, per farlo andiamo sul sito del server /admin e ci logghiamo con i dati inseriti alla creazione del superuser di Django.  
//foto login  
Ci apparirà il sito amministrativo del backend.  
Questo sito è creato automaticamente da Django e gestito nel file “backend/admin.py”,   
A questo punto bisogna aggiungere un nuovo corso, quindi andare su “corsi” e premere “add”  
//foto form aggiunta corso  
Compilare il form e salvare il corso non spuntando la casella “processa corso”, in questo modo andremo solamente ad aggiungere i dati inseriti nel database. Dopo di che procediamo ad aggiungere le varie lezioni, per farlo bisognerà andare nella sezione lezioni e premere add.  
// foto form lezione  
Fra i vari dati richiesti notiamo l’url di Kiro che contiene la videolezione, servirà a reindirizzare gli studenti che vorranno visionarla; mentre invece i link di YouTube o Vimeo servono unicamente per scaricare i sottotitoli o il filmato stesso per poterli analizzare e non saranno accessibili dagli studenti.  
Possiamo notare su admin.py che il form è stato modificato, per prima cosa faccio un controllo sui dati inseriti, infatti si può o caricare direttamente il video della lezione o inserire un link, che però deve essere di YouTube o di Vimeo. Notiamo inoltre che dopo aver salvato la lezione viene fatta partire la funzione “AnalyzeVideo(obj).start()” dove obj rappresenta i dati inseriti nel form, quindi la lezione stessa.

Passiamo quindi al cuore del progetto, spostiamoci su “backend/AggregateData/parseVideo.py” e notiamo che la funzione appena lanciata ha fatto partire un nuovo thread che si occuperà di analizzare la lezione inserita.  
Per prima cosa distingue in che modo è stata inserita la lezione, se tramite file o link e, in quel caso, se Vimeo o YouTube; fa quindi partire il processo adeguato per estrapolarne i dati, nella fattispecie:

Video da locale:

Nel caso in cui sia stato scaricato un file direttamente verrà salvato momentaneamente nella cartella “Media/Video”, per elaborarlo viene per prima cosa estratto l’audio dal video e salvato in formato “flac” nella cartella “Media/Audio”, a farlo ci pensa la calasse “Video2audio” che sfrutta la libreria Python moviepy; l’audio per poter essere analizzato poi su Google Cloud deve necessariamente essere nel formato flac e con una singola traccia.  
A questo punto bisogna caricare in un bucket di Google Cloud la traccia audio, lo si fa tramite la classe “Speech2Text” che si occupa anche di analizzare l’audio caricato salvando anche i timestamp delle parole e aggiungendo della punteggiatura in modo automatico; dopo averlo analizzato elimina l ‘audio dal bucket e porta i dati in un formato standard che è quello ottenuto con i sottotitoli da YouTube, cioè una o due parole separate da un timestamp racchiuso fra <> con il formato hh:mm:ss.fff come nel seguente esempio

*allora<00:00:19.910> eccoci<00:00:20.360> qua<00:00:20.660> a<00:00:20.750> iniziare<00:00:21.380>*

*A quel punto verrà analizzato con un processo che vedremo in seguito e alla fine verranno eliminati i file video e audio generati.*

*Link di Vimeo*

Nel caso in cui viene inserito un link di Vimeo per prima cosa viene estrapolato l’id del video e, tramite la classe VimeoDownload, viene scaricato il file video nella cartella “Media/Video”, per farlo servono due chiamate alle api di Vimeo, la prima per sapere tutti i formati video disponibili e i relativi link di download, si sceglie quindi il video meno pesate e lo si scarica.  
A quanto punto ci ritroviamo nella stessa situazione del video da locale e si procede in egual modo.

Link di YouTube

Per quanto riguarda youtube la faccenda è differente, per prima cosa viene estratto l’id del video dal link, vengono poi generate delle credenziali tramite la classe youtubeCredentials che, se non presente o scaduto, richiede un token a youtube, viene quindi passata alla classe captionDownload che scarica i sottotitoli generati automaticamente in italiano del video che infine vengono passati alla classe cropCaption che va a eliminare elementi ridondanti o inutili e genera il testo nel formato standard.  
Si è scelto di scaricare direttamente i sottotitoli e non il video intero per poterne scaricare un numero maggiore al giorno, rimane però il problema che i sottotitoli di YouTube non hanno la punteggiatura quindi non sarà possibile eseguire l’LDA come vedremo in seguito, è invece possibile eseguire l’indexing delle parole come ora descritto.

### Indexing delle parole:

Ora che abbiamo il trascritto di una lezione con i timestamp in un formato a noi conosciuto possiamo passare all’indexing delle parole chiave.

Il primo passo è tokenizzare lo scritto, per fare questo usiamo la classe “Tokenize” che divide il testo in frasi, controlla che il server di Tint sia attivo e in caso contrario lo lancia, dopo di che gli manda una frase alla volta per essere elaborata.  
L’output generato è una lista di oggetti ognuno contenente la parola, il timestamp di quando è stata pronunciata e il PoS (part-of-speech), cioè l’analisi grammaticale delle frasi.  
Il PoS è una parte estremamente importante del programma perché permette di escludere dall’indentazione la maggior parte delle parole inutili per la ricerca degli argomenti come gli articoli, le preposizioni, i pronomi etc. È possibile trovare una lista di tutte le abbreviazioni per il PoS utilizzate da Tint con il loro significato a: <http://medialab.di.unipi.it/wiki/POS_and_morphology>

Avendo ottenuto i vari token si può procedere all’analisi del testo vera e propria, per prima cosa siamo andati a studiare quali tipologia di parole identificate dal PoS fossero le più utili per i nostri scopi, l’intero programma è stato concepito per poter modificare questa classificazione facilmente e permettere di fare futuri test e studi.  
Facendo diverse prove su 4 lezioni del corso “Basi di dati”, in particolare le 1, 2, 3 e 12, abbiamo constatato tramite revisione manuale che i nomi e gli aggettivi portavano assieme il maggior numero di informazioni utili per i nostri scopi. Nelle abbreviazioni di Tint sono classificate rispettivamente come “S” e “A”.  
Ho quindi proceduto a creare una tabella nel database contenente tutti i nomi e aggettivi, salvando la lezione di appartenenza e il timestamp.

### Struttura dati:

Mentre all’inizio bastavano dei file csv per poter gestire e analizzare manualmente i dati, nel momento in cui abbiamo voluto spostare il programma dalla semplice analisi a un sistema in grado anche di recuperare i dati prodotti abbiamo dovuto implementare un database.  
Per farlo si sfruttato il funzionamento di Django e utilizzato il file “backend/models.py”.

Per la struttura dei dati si è pensato prima di tutto a una tabella contenete i corsi, che contenesse quindi il nome e l’URL di Kiro dello stesso; è stato poi necessario aggiungere un booleano per visualizzare se il corso era già stato processato nel suo insieme o meno.  
Inoltre il nome del corso è stato reso univoco per permettere una ricerca univoca dello stesso da parte di uno studente; in caso dell’aggiunta di uno stesso corso in diversi anni si potrà o eliminare i corsi precedenti, oppure aggiungere al nome del corso l’anno in cui è stato svolto.  
Per effettuare una più veloce query dei nomi nel database sono stati indexati.  
Sono stati infine aggiunti help text e nomi in italiano per consentire una migliore esperienza utente durante l’immissione dei dati nel sito amministrativo.

Per le lezioni invece oltre al nome si è aggiunto il link di Kiro contenente la videolezione, il corso di appartenenza come chiave esterna, l’URL del video originale che può essere nullo, il video come file, pure lui può essere nullo, un booleano per decidere se eseguire o meno l’LDA sulla lezione e infine un booleano creato automaticamente che identifica se la lezione è già stata processata.  
Il controllo se almeno uno dei due campi fra l’URL del video o il video stesso sono stati compilati vene fatto durante la validazione del form per aggiungere una lezione, così come se l’URL è di YouTube o di Vimeo.  
Il video viene caricato direttamente nella cartella “Media/Video/”.  
È stato specificato che nel caso in cui il corso a cui appartiene una lezione tramite chiave esterna dovesse essere eliminato anche la lezione stessa verrà eliminata.  
Sono stati infine aggiunti help text e nomi in italiano per consentire una migliore esperienza utente durante l’immissione dei dati nel sito amministrativo.

Per salvare le parole chiave di ogni lezione è stata creata una tabella comprendente le parole stesse lemmizzate, la lezione di appartenenza come chiave esterna e il timestamp di quando è stata detta.  
la parola è stata indexata nel database per consentire una più veloce ricerca, è stato specificato che in caso di eliminazione della lezione di appartenenza debba essere anche cancellata la tupla con la parola chiave.

Per i binomi si è adottata una soluzione simile a quella per le parole, salvando le due parole che compongono il binomio separatamente ma indexandole assieme nel datatbase.  
Lezione e timestamp come per le parole.

Per il conteggio delle parole e dei binomi sono state create delle tabelle apposta, infatti, nonostante sia possibile ricavarli dalle tabelle binomi e parole tramite delle specifiche query sarebbe una soluzione lenta e inutile, si è quindi deciso che una volta che si inserisce la lezione viene effettuato il conteggio delle parole e binomi nella stessa, e una volta finito di inserire le lezioni in un corso si generano quelle dello stesso.

Per il conteggio delle parole e dei binomi nelle lezioni le tabelle sono molto simili ma in una è contenuta una parola mentre nell’altra un binomio, in questo caso non sono stati lemmizati ma è stato semplicemente preso il primo caso trovato all’interno di una lezione e usato quello come riferimento.  
Si aggiunge la lezione come chiave esterna, specificando che in caso di eliminazione debba essere eliminato anche questo dato, il conteggio delle parole che viene indexato nel db e il tf (term frequency) cioè il numero di volte in cui compare quella parola diviso il numero di parole totali della lezione.

Per il conteggio nei corsi la situazione è simile con la differenza che la chiave esterna questa volta è di un corso e non è presente il tf.

Per eseguire successivamente l’LDA di un intero corso è necessario salvare le frasi di ogni lezione, per farlo è stata creata una struttura comprendente la frase come testo, il numero della frase nella lezione e la lezione stessa come chiave esterna.

Per l’LDA sono state create due tabelle distinte, una per le lezioni e una per i corsi ma di base sono simili a differenza della chiave esterna che può appunto essere di una lezione o di un corso, in aggiunta è presente un array di stringhe per contenere i vari topic trovati dall’LDA.  
Quest’ultimo campo è presente solo in PostgreSQL ed è questo uno dei motivi della sua scelta.

BISOGNA AGGIUNGERE IDF E TF-IDF