TECNOLOGIE DI CRIMEMINER

* FRONTEND
* NodeJs

Strumento che permette di installare tutte le librerie necessarie per la costruzione e visualizzazione del frontend, attraverso il comando **npm**.

* Bootstrap

Libreria base contenente template fondamentali per gli stili del sito.

* SASS

Potente libreria che permette la compilazione di file con estensione **.scss** in file con estensione **.css.map** e **.css**, così da poter scrivere gli stili in modo più performante, evitando ridondanze per i vari browser, permettendo di inserire variabili e organizzare meglio il file.

* Cytoscape.js

Libreria che permette la visualizzazione e gestione di grafici orientati all’interno di una pagina web.

* Cytoscape-dagre.js

Modulo di Cytoscape.js che permette la visualizzazione **dagre** del grafo (per poter funzionare ha bisogno della libreria “dagre.js”)

* Cytoscape-fcose.js

Modulo di Cytoscape.js che permette la visualizzazione **fcose** (Force Atlas Layout) del grafo (per poter funzionare ha bisogno delle librerie “layout-base.js” e “cose-base.js”)

* BACKEND
* Neo4j

È il software che gestisce il **database** relativo alle informazioni sui nodi e gli archi. È un database non relazionale che per comunicare con il backend fa uso di **query** della tipologia **Cypher**.

* Django

È un framework di **Python** che viene usato per ricevere richieste dal frontend ed effettuare operazioni sul database gestito da Neo4j.

* django-neomodel

È un modulo di Django che permette la comunicazione tra Django e Neo4j in modo da gestire le operazioni sul database. Le operazioni riguardano soprattutto i metodi CRUD di inserimento e rimozione vista la corrispondenza 1 ad 1 delle tabelle con il model.

* django-request-mapping

È un modulo di Django che permette di accedere alle View dell’applicativo tramite delle url.

STRUTTURA FRONTEND DI CRIMEMINER

* DASHBOARD

È la pagina principale che contiene informazioni comuni a tutte le pagine dell’applicativo. Per questo scopo, contiene al suo interno lo stile e gli script che permettono una buona visualizzazione e il corretto funzionamento del sito.

In particolare, contiene:

* L’**header,** che permette di visualizzare il nome della pagina corrente (presente in “app/templates/dashboard.html”).
* La **sidebar**, che visualizza il logo dell’applicativo e permette di navigare tra le varie **view** sotto richiesta dell’utente (presente in “app/templates/dashboard.html”).
  + Le icone utilizzate sono presenti nella directory “app/static/img/dashboardHeader/”.
  + La rotazione delle frecce dei sottomenù è gestita dallo script presente nel file “app/static/js/dashboard.js”.
* Lo **stile** dell’header e della sidebar (presente in “app/static/js/dashboard.scss”).
* LAYOUT PAGE GRAPH

È l’insieme delle componenti delle varie view che non si diversificano. Questa componente non ha una sua vera e propria pagina html.

In particolare, contiene:

* Lo **stile** degli accordion (proprietà e impostazioni contenute nelle varie view) delle varie view (presente in“app/static/js/templates/layoutGraphPage.scss”).
* La rotazione delle frecce dei gli è gestita dallo script presente nel file “app/static/js/templates/layoutGraphPage.js”.
* INDIVIDUAL WIRETAPS

È la view che contiene il grafo degli individui collegati dalla relazione “HaChiamato”, permette l’inserimento, la modifica e la cancellazione degli individui e delle chiamate. In più permette l’anonimizzazione dei dati dell’individuo (eccetto l’identificativo) che verrà mantenuta tramite i cookie per le varie view.

In particolare, contiene:

* Il grafo della view, presente all’interno del div con classe “cyContent” che verrà caricato tramite una chiamata effettuata al backend e dopo la sua creazione verranno gestiti tutti i comportamenti sui click dei nodi, degli archi e sul background.
* Le modali di inserimento, modifica e cancellazione che avranno opportuni metodi per l’apertura delle modali stesse e per l’invio delle informazioni.

I fogli di markup, stile e script saranno tutti identificati col nome “individualWiretaps”, rispettivamente:

* markup: “app/templates/features/individualWiretaps/individualWiretaps.html”
* stile: “app/static/css/individualWiretaps/individualWiretaps.scss”
* script: “app/static/js/individualWiretaps/individualWiretaps.js”
* INDIVIDUAL CRIMES

È la view che contiene il grafo degli individui e dei reati collegati dalle relazioni “ImputatoDi” e “Condannato”, permette l’inserimento, la modifica e la cancellazione degli individui e delle chiamate. In più permette l’anonimizzazione dei dati dell’individuo (eccetto l’identificativo) che verrà mantenuta tramite i cookie per le varie view.

In particolare, contiene:

* Il grafo della view, presente all’interno del div con classe “cyContent” che verrà caricato tramite una chiamata effettuata al backend e dopo la sua creazione verranno gestiti tutti i comportamenti sui click dei nodi, degli archi e sul background.
* Le modali di inserimento, modifica e cancellazione che avranno opportuni metodi per l’apertura delle modali stesse e per l’invio delle informazioni.

I fogli di markup, stile e script saranno tutti identificati col nome “individualCrimes”, rispettivamente:

* markup: “app/templates/features/individualCrimes/individualCrimes.html”
* stile: “app/static/css/individualCrimes/individualCrimes.scss”
* script: “app/static/js/individualCrimes/individualCrimes.js”
* INDIVIDUAL ENVIROMENTAL TAPPING

È la view che contiene il grafo degli individui e delle intercettazioni ambientali collegati dalla relazione “Presente”, permette l’inserimento, la modifica e la cancellazione degli individui e delle chiamate. In più permette l’anonimizzazione dei dati dell’individuo (eccetto l’identificativo) che verrà mantenuta tramite i cookie per le varie view.

In particolare, contiene:

* Il grafo della view, presente all’interno del div con classe “cyContent” che verrà caricato tramite una chiamata effettuata al backend e dopo la sua creazione verranno gestiti tutti i comportamenti sui click dei nodi, degli archi e sul background.
* Le modali di inserimento, modifica e cancellazione che avranno opportuni metodi per l’apertura delle modali stesse e per l’invio delle informazioni.

I fogli di markup, stile e script saranno tutti identificati col nome “individualEnviromentalTapping”, rispettivamente:

* markup: “app/templates/features/individualEnviromentalTapping/individualEnviromentalTapping.html”
* stile: “app/static/css/individualEnviromentalTapping/individualEnviromentalTapping.scss”
* script: “app/static/js/individualEnviromentalTapping/individualEnviromentalTapping.js”
* INDIVIDUAL CRIMES ENVIROMENTAL TAPPING

È la view che contiene il grafo avente le informazioni delle view precedenti. In più permette l’anonimizzazione dei dati dell’individuo (eccetto l’identificativo) che verrà mantenuta tramite i cookie per le varie view.

In particolare, contiene:

* Il grafo della view, presente all’interno del div con classe “cyContent” che verrà caricato tramite una chiamata effettuata al backend e dopo la sua creazione verranno gestiti tutti i comportamenti sui click dei nodi, degli archi e sul background.

I fogli di markup, stile e script saranno tutti identificati col nome “individualCrimesEnviromentalTapping”, rispettivamente:

* markup: “app/templates/features/individualCrimesEnviromentalTapping/individualCrimesEnviromentalTapping.html”
* stile: “app/static/css/individualCrimesEnviromentalTapping/individualCrimesEnviromentalTapping.scss”
* script: “app/static/js/individualCrimesEnviromentalTapping/individualCrimesEnviromentalTapping.js”
* INTERLOCUTORE CRIMES

È la view che contiene il grafo degli interlocutori collegati dalle relazioni etichettate “HaConversato” e che consente di effettuare un recupero dei dati degli interlocutori e delle conversazioni che essi hanno avuto tra di loro. In più è permessa l’anonimizzazione dei dati dell’interlocutore (eccetto l’identificativo) che verrà mantenuta tramite i cookie per le varie view.

In particolare, contiene:

* Il grafo della view, presente all’interno del div con classe “cyContent” che verrà caricato tramite una chiamata effettuata al backend e dopo la sua creazione verranno gestiti tutti i comportamenti sui click dei nodi, degli archi e sul background.
* Sono stati posti tre filtri di ricerca: un filtro che permette di selezionare l’emozione di interesse, che restituisce i nodi che hanno quella emozione e poi cliccando su un nodo di interesse restituito dal filtro, è possibile vedere in che conversazione quel nodo ha provato l’emozione indicata e con chi ha avuto quella conversazione;   
  un filtro per nome, ovvero l’utente può inserire il nome di interesse e visualizzare le informazioni di quell’interlocutore e anche le relazioni che esso ha avuto con altri nodi; ultimo filtro permette, invece, scegliere una data di inizio e una data di fine che permettono di definire un range temporale di interesse e quindi visualizzare un grafo che interessa solo quel range, restituendo nodi e archi di quel periodo.  
  Inoltre, è possibile combinare i filtri tra di loro, attraverso un apposito tasto, effettuando combinazioni da tre o da due.

I fogli di markup, stile e script saranno tutti identificati col nome “interlocutoreCrimes”, rispettivamente:

* markup: “app/templates/features/interlocutoreCrimes.html”
* stile: “app/static/css/interlocutoreCrimes/interlocutoreCrimes.scss”
* script: “app/static/js/features/interlocutoreCrimes.js”
* MODULI AUSILIARI
* Cookie

Modulo custom per l’utilizzo dei cookie all’interno della piattaforma.

Per usarli bisogna usare i seguenti comandi:

* **document.cookie** che è standard per Javascript per la creazione del cookie
* **setCookie**(“valore1”, ”valore2”) che è una funzione custom che permette di modificare il valore di un cookie mettendo in “valore1” il nome del cookie, in “valore2” il valore da associargli. Sarebbe possibile inserire anche delle option aggiungendo un terzo campo che è opzionale.
* **getCookie**(“valore1”) che è una funzione custom che permette di ricevere il valore del cookie “valore1”

I metodi custom è possibile trovarli nel file “CrimeMiner/static/js/cookie.js”

* Toast

Modulo custom che usa i toast di Bootstrap e modifica il loro colore in base alla tipologia. Bisognerà inserire il comando **viewToastMessage**(title, content, tipology) dove verranno inseriti titolo e contenuto e in tipology, potranno essere inseriti:

* “success”: per un’operazione andata a buon fine (il colore del toast sarà verde)
* “alert”: per una segnalazione (il colore del toast sarà giallo)
* “error”: per un’operazione non riuscita (il colore del toast sarà rosso)

Il metodo viewToastMessage è presente all’interno del file toast.js (presente in “CrimeMiner/static/js/toast.js”)

* Load

Modulo di Bootstrap che permette la visualizzazione del caricamento nelle pagine, si usa tramite il comando **loadPage**(valore) e il valore dovrà essere un intero in millisecondi per inserire la durata stessa del caricamento

In particolare, abbiamo:

* Lo **stile** del caricamento (presente in “CrimeMiner/static/css/load.scss”)
* La funzione che permette il suo funzionamento (presente in “CrimeMiner/static/js/load.js”)

STRUTTURA BACKEND DI CRIMEMINER

* ROUTING

La Gestione degli url per la visualizzazione delle pagine viene fatta utilizzando due metodi, primo il metodo di routing è quello già offerto da Django, all’interno del file “CrimeMiner/app/urls.py”. Si utilizza una variabile **urlpattern[]** in cui vengono inseriti i path delle pagine a cui sono associati i file html. Per la visualizzazione delle view che prevedono l’utilizzo di metodi Get/Post e dunque lo scambio di dati, si è visto necessario utilizzare allo stesso tempo un altro metodo di routing offerto da Django Request Mapping. Questo metodo svolge sempre la stessa funzionalità, ma prevede solo la registrazione delle view con il metodo **urlpattern.register()** dal momento che i path sono già assegnati all’interno delle pagine stesse lato backend.

* CONNESSIONE AL DB

La connessione al database viene realizzata nel file “CrimeMiner/app/Neo4jConnection.py”.

In questo file sono state riportate anche le query che hanno permesso il caricamento dei nuovi dati del file csv presente nella cartella “CrimeMiner/sentiment\_analysis”. Le query in questione sono state commentate per permettere una comprensione futura o gestione migliore dei dati.   
Qui vengono realizzati due tipi di connessione: prima di tutto viene definito il **Neo4jDriver**  con i dati di accesso al db (**uri,user,password**) per l’utilizzo delle query cypher. Nello specifico per utilizzare il driver nelle repository, viene istanziata una sessione di connessione, a cui viene passata la query come parametro e su cui poi viene invocato il metodo .**run().** Il secondo tipo di connessione si è visto necessario per realizzare la comunicazione con Neo4j attraverso Django Neomodel e l’utilizzo delle operazioni crud, in quanto prevedeva una diversa sintassi e doveva essere inserito nel metodo **inizialize**() della connessione per essere eseguito un’unica volta con il comando **config.DATABASE\_URL**. Nel file Neo4jConnection.py vengono inoltre realizzate le **Projection** dei vari grafi per l’implementazione delle **metriche** all’interno dell’applicativo, in quanto è necessario utilizzare il comando **gds.projection** solamente all’avvio di CrimeMiner per poi richiamare e calcolare la specifica metrica nella pagina assegnata a quel determinato grafo.

NOTA: per carica un file csv su neo4j e rispettare il path di default da esso voluto, inserire i file all’interno della cartella import di neo4j.

* INDIVIDUAL WIRETAPS

Per quanto riguarda il lato back-end della pagina Individual Wiretaps, questa è strutturata nel seguente modo:

1. Nel del file “CrimeMiner/app/Models/Entity/IndividuoModel.py” è stata definito il modello con corrispondenza 1 ad 1 alla tabella **Individuo** in Neo4j con i campi e le relazioni in cui è coinvolto. Una volta definito il modello, le informazioni relative agli individui vengono ricavate attraverso le query al db nel file “CrimeMiner/app/repositories/IndividuoRepository.py” e richiamate all’interno della view “../Views/IndividuoView.py”.
2. Le informazioni relative alla intercettazione tra due individui sono presenti all’interno della relazione “**HaChiamato**” a cui corrisponde il model “../Models/Relationship/HaChiamatoModel.py”. Analogamente all’individuo, le informazioni relative ad un intercettazione, insieme alle metriche, vengono ricavate nella repository “../../IndividuoIntercettazioneRepository.py” e richiamate all’interno della view “../../IndividuoIntercettazioneView.py”.
3. I metodi della view presentano l’annotazione @**requestMapping** così da facilitare il trasferimento di dati tra back-end e front-end.

* INDIVIDUAL CRIMES

Per quanto riguarda il lato back-end della pagina Individual Crimes, questa è strutturata nel seguente modo:

1. Nel del file “CrimeMiner/app/Models/Entity/ReatoModel.py” è stata definito il modello con corrispondenza 1 ad 1 alla tabella **Reato** in Neo4j con i campi e le relazioni in cui è coinvolto. Una volta definito il modello, le informazioni relative agli reati vengono ricavate attraverso le query al db nel file “CrimeMiner/app/repositories/ReatoRepository.py”.
2. Le informazioni relative alla condanna o un’imputazione di un individuo verso un reato, sono presenti all’interno delle relazioni “**ImputatoDi**” e “**Condannato**” a cui corrispondono i model “../Models/Relationship/ImputatoDiModel.py”, “../Models/Relationship/CondannatoModel.py”. Le informazioni relative ad un individuo e il modo in cui è collegato ad un determinato reato, insieme alle metriche, vengono ricavate nella repository “../../IndividuoReatoRepository.py” dove ogni volta viene fatta distinzione tra un’imputazione o una condanna per fare la query. I metodi che nella repository effettuano queste query vengono poi richiamati all’interno della view “../../IndividuoReatoView.py”.
3. I metodi della view presentano l’annotazione @requestMapping così da facilitare il trasferimento di dati tra back-end e front-end.

* INDIVIDUAL ENVIROMENTAL TAPPING

Per quanto riguarda il lato back-end della pagina Individual Enviromental Tapping, questa è strutturata nel seguente modo:

1. Nel del file “CrimeMiner/app/Models/Entity/IntercettazioneAmbModel.py” è stata definito il modello con corrispondenza 1 ad 1 alla tabella **IntercettazioneAmb** in Neo4j con i campi e le relazioni in cui è coinvolto. Una volta definito il modello, le informazioni relative alle intercettazioni ambientali vengono ricavate attraverso le query al db nel file “CrimeMiner/app/repositories/IntercettazioneAmbRepository.py” e richiamate all’interno della view “../Views/IntercettazioneAmbView.py”.
2. Le informazioni relative collegamento di un individui in un intercettazione ambientale sono presenti all’interno della relazione “**Presente**” a cui corrisponde il model “../Models/Relationship/PresenteModel.py”. Le informazioni relative ad un intercettazione insieme alle metriche, vengono ricavate nella repository “../../IndividuoIntercettazioneAmbRepository.py” e richiamate all’interno della view “../../IndividuoIntercettazioneAmbView.py”.
3. I metodi della view presentano l’annotazione @**requestMapping** così da facilitare il trasferimento di dati tra back-end e front-end.

* INDIVIDUAL CRIMES ENVIROMENTAL TAPPING

Per quanto riguarda il lato back-end della pagina Individual Crimes Enviromental Tapping, questa è strutturata nel seguente modo:

1. È una tabella ricavata dall’unione di tutte le precedenti così da avere una panoramica generale nel grafo in tutta la sua interezza. Di conseguenza non è presente un model ma bensì utilizza i singoli model per la costruzione del grafo. Le informazioni relative alle intercettazioni tra individui, i reati associati, e il collegamento con un’intercettazione ambientale, insieme alle metriche, vengono ricavate nella repository “../../IndividuoReatoIntercettazioneAmbRepository.py” e richiamate all’interno della view “../../IndividuoReatoIntercettazioneAmbView.py”.
2. I metodi della view presentano l’annotazione @**requestMapping** così da facilitare il trasferimento di dati tra back-end e front-end.

* INTERLOCUTORE CRIMES

Per quanto riguarda il lato back-end della pagina Interlocutore Crimes, questa è strutturata nel seguente modo:

1. Nel file “CrimeMiner/app/Models/Entity/InterlocutoreModel.py” è stata definito il modello con corrispondenza 1 a 1 alla tabella **Interlocutore** in Neo4j con i campi e le relazioni in cui è coinvolto. Si ha anche “CrimeMiner/app/Models/Entity/FraseModel.py” per le proprietà di Frase.  
   Una volta definito il modello, le informazioni relative agli individui vengono ricavate attraverso le query al db nel file “CrimeMiner/app/repositories/InterlocutoreRepository.py” e richiamate all’interno della view “../Views/InterlocutoreCrimesView.py”.
2. Le informazioni relative alla conversazioni/telefonate/intercettazioni tra due o più interlocutori sono presenti all’interno della relazione “**HaConversato**” a cui corrisponde il model “../Models/Relationship/HaConversatoModel.py”. Analogamente all’interlocutore, le informazioni relative ad una conversazione, insieme alle metriche, vengono ricavate nella repository “../../InterlocutoreInfoRelazioneRepository.py” e richiamate all’interno della view “../../InterlocutoreCrimesView.py”.
3. I metodi della view presentano l’annotazione @**requestMapping** così da facilitare il trasferimento di dati tra back-end e front-end.

SENTIMENT ANALYSIS E EMOTION CLASSIFICATION

La sentiment analysis e l’emotion classification sono presenti nella directory “CrimeMiner/sentiment\_analysis”. Il database iniziale è il file “database-prova.xlsx” in formato excel. Di seguito gli script in ordine di esecuzione:

1. **modifica-dataset.py**: ha permesso la lettura del file excel e diverse modifiche e pulizie. Inoltre, è stato creato un nuovo file csv che ha diviso ogni conversazione in singole frasi.
2. **sentiment\_analysis.py**: sul file csv ottenuto dallo script precedente, e chiamato “frase\_dataset.csv”, è stato applicato il neuraly-italian bert sentiment model (etichette: positive, negative, neutral) per la sentiment analysis, creando una nuova colonna chiamata “sentimento” e un nuovo file csv.
3. **feel\_emotion.py**: sul file csv ottenuto dallo script precedente, e chiamato “frase\_sentiment\_analysis.csv”, è stato applicato il modello feel-it per l’emotion classification (joy, anger, fear, sadness). Questo ha portato alla creazione di un nuovo file csv, che presenta due nuove colonne: la colonna “emozione” in cui è riportata l’emozione predominante assegnata ad ogni frase e la colonna “lista\_emozioni” in cui vengono riportati tutti i valori di predizione assegnati alle varie emozioni.
4. **Sostituzione\_null.py**: questo file ha permesso la lettura dell’ultimo file csv ottenuto e l’inserimento della stringa “dato non fornito” per ogni cella del dataset che presentava valore null. Le colonne che presentavano valori null sono indicati nello script. Questo è stato effettuato per non perdere dati importanti.

Il file finale ottenuto è “**frase\_sentiment\_emotion\_modificato.csv**” ed è quello utilizzato per il caricamento dei dati all’interno del database gestito da neo4j.