Alma Mater Studiorum
Università di Bologna
Campus di Cesena

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

PIATTAFORMA DI SOCIAL MEDIA SENSING NEL CONTESTO DELLA RIGENERAZIONE URBANA: IL PROGETTO DARE-SAM

Elaborato in

Tecnologie Web

Relatore:
Prof.ssa
SILVIA MIRRI

Presentata da:
GIOVANNI GOBBI

Correlatore:
Dott.ssa
CATIA PRANDI

Terza sessione di Laurea
Anno Accademico 2019-2020

Introduzione

Negli ultimi anni il mondo intero ha assistito ad una crescita esponenziale dell'utilizzo di Internet [1]. Questo è riconducibile in gran parte all'esplosione di popolarità di una specifica tipologia di servizi: i social media. Ad esempio Facebook, social network più popolare al mondo, ha 2,7 miliardi di utenti attivi [2]; Youtube, piattaforma di condivisione video, ha più di 2 miliardi di utenti attivi [3]. In generale, secondo la fonte *Our world in data* [4], nel 2019 circa un terzo della popolazione mondiale faceva regolare uso di piattaforme social, ovvero circa due terzi di tutti gli utilizzatori di Internet.
La recente pandemia globale ha ulteriormente incrementato questi numeri ma soprattutto il tempo che ciascuno di noi trascorre sui social media e con ciò, secondo uno studio svolto da Suzan Wold su 4.500 persone in Nord America [5], la quantità di contenuti pubblicati da ciascun utente. Lo studio evidenzia anche come sempre più persone utilizzano questi strumenti come mezzi di informazione ed e-commerce.
Questo numero enorme di utilizzatori non può fare altro che generare un'altrettanto enorme quantità di dati [6]. Essi sono distinguibili in due macro categorie: contenuti pubblicati e interazioni da parte di utenti con essi.
Infatti, sebbene parte importante del funzionamento dei social media sia quella di permettere la creazione e la condivisione di contenuti agli utenti, la componente dell'interazione e visualizzazione di essi è fondamentale (senza nessuna visualizzazione, non ci sarebbe nessun interesse a creare nuovi contenuti).
Per evidenziare l'interesse da parte delle aziende realizzatrici di social media

nel migliorare continuamente le modalità con cui è possibile interagire con i contenuti, è bene descrivere brevemente il modello di business che queste aziende hanno adottato negli ultimi anni. Il guadagno primario derivante dai social media (e per molti dei quali, tra cui Facebook [7], anche l'unico) è dovuto alla vendita di pubblicità e inserzioni da mostrare agli utenti finali mentre essi interagiscono con i contenuti sulla piattaforma. Dalla rivoluzione dei social media è nata la necessità di fornire inserzioni rilevanti all'utente (sempre citando Mark Zuckerberg nel suo articolo relativo al business model di Facebook [7]: *"Le persone ci dicono continuamente che se devono visualizzare pubblicità, vogliono che esse siano rilevanti."*) e, per permettere questo, è necessario avere più informazioni sui suoi gusti. Ciò è realizzabile in gran parte analizzando come l'utente interagisce con i contenuti proposti dalla piattaforma, e perciò le aziende realizzatrici di social media hanno e stanno studiando nuove modalità con cui gli utenti possano esprimere un parere su un determinato contenuto.

Tra le interazioni più comuni c'è sicuramente il famosissimo pulsante Like [8] reso popolare da Facebook dal 2009 in avanti, ma anche tante altre incluse le più recenti reazioni introdotte da alcuni social media come Facebook [9] e LinkedIn [10]. L'interazione più importante però è la possibilità di commentare un determinato contenuto, inserendo del testo (e recentemente anche immagini e video). Sebbbene infatti le interazioni come Like e reazioni esprimano un giudizio approssimativo sul contenuto (*"Mi piace"*, *"Non mi piace"*, ecc.), un commento è in grado di contenere molte più informazioni.

È intuitivo concludere che i commenti pubblicati sui social media contengono tantissime informazioni che possono essere estremamente importanti per un brand o un personaggio pubblico. Il problema fondamentale è che post ad alta visibilità ricevono decine di migliaia di commenti, rendendo praticamente impossibile analizzare ognuno di essi singolarmente. La soluzione più efficace è quella di usufruire di qualche servizio in grado di estrarre conoscenza da essi in maniera automatica, e lo strumento principale che la maggior parte di questi servizi utilizzano è la Sentiment Analysis.

Ma cosa mette in relazione tutto questo e l'obiettivo del progetto DARE?
DARE (*Digital Environment for collaborative Alliances to Regenerate urban Ecosystems in middle-sized cities*) [11] è un progetto nato nella realtà di Ravenna che ha come scopo la rigenerazione urbana della città. È finanziato da UIA (*Urban Innovative Actions*) [12], iniziativa Europea che mira ad individuare e mettere in pratica nuove idee innovative nell'ambito dello sviluppo urbano sostenibile. Esso ha finanziato ben 86 progetti in 20 nazioni Europee.
Nello specifico, il progetto DARE mira a riqualifare la *Darsena* [13], area cittadina del comune di Ravenna di particolare interesse. Essa in passato comprendeva il porto e perciò era diventata la sede del polo industriale più importante della città. Da quando quest'ultimo è stato trasferito lontano dal centro cittadino, la Darsena si è trasformata in un'area abbandonata posizionata accanto ad un'area residenziale densamente popolata (20284 persone). Quello che DARE ha in mente è trasformare la zona seguendo un più ampio processo di transizione digitale. Allo stesso tempo, il progetto punta anche a coinvolgere le prospettive dei singoli cittadini in modo che essi si sentano parte del processo innovativo per proporre nuove idee e progetti a scopo di migliorare la propria qualità di vita.
Questo secondo importante obiettivo evidenzia la necessità di un servizio in grado di raccogliere e analizzare proposte e opinioni dei cittadini Ravennati e, dato che al giorno d'oggi la maggior parte delle interazioni si svolge sui social media, lo strumento ideale è una piattaforma apposita di social media sensing.
Da qui nasce DARE-SAM (in forma estesa *DARE Sentiment Analysis and Monitoring*), piattaforma web pensata per aiutare il team di DARE a raggiungere i propri obiettivi attraverso l'analisi delle interazioni ricevute dai profili DARE presenti sui social media, in particolare pagina Facebook e profilo Instagram.
Questo elaborato ha come scopo quello di sviluppare una versione iniziale

della piattaforma che soddisfi tutti i requisiti richiesti tenendo in considerazione estensibilità ed efficienza. Si è posta particolare enfasi sulle fase di progettazione dell'architettura con l'obiettivo di sviluppare una soluzione in grado di superare le sfide proposte dalle modalità di accesso ai dati presenti sui social media. Un'altra parte fondamentale che ha richiesto un ampio lavoro di ricerca è stata l'implementazione della Sentiment Analysis in quanto diverse soluzioni molto diverse tra di loro (divise tra librerie open source e servizi cloud) hanno richiesto un'ampia comparazione secondo parametri come accuratezza, efficienza, semplicità di implementazione e scalabilità. Per quanto riguarda lo sviluppo, sono state scelte tecnologie considerate all'avanguardia come React [14] e Next.js [15] per la piattaforma web e Node.js [16] per la parte legata al backend, con l'obiettivo di rendere il progetto estremamente attuale e facilmente manutenibile.

Il contenuto di questo elaborato è articolato nei seguenti capitoli:

- Capitolo 1: presenta un'accurata analisi sullo stato dell'arte della Sentiment Analysis, partendo dalle tecniche utilizzate e dal funzionamento fino all'introduzione delle piú popolari librerie open source e i piú astratti servizi offerti dai provider cloud.
- Capitolo 2: contiene la completa fase di progettazione della piattaforma; essa presenta inizialmente i requisiti che il software deve soddisfare e successivamente affronta la fase di progettazione vera e propria in cui si analizzano le sfide principali e se ne propongono le soluzioni; infine é ripotata una sezione dedicata alle tecnologie principali utilizzate per l'implementazione di DARE-SAM divise per ambito.
- Capitolo 3: descrive nel dettaglio l'architettura che é stata scelta per realizzare DARE-SAM e introduce alcune modalitá per realizzare l'hosting del sistema; successivamente elenca alcuni dettagli implementativi che hanno richiesto soluzioni impegnative e interessanti.

- Capitolo 4: presenta una guida completa all'utilizzo del software da parte di utenti finali, incluse illustrazioni relative ai profili social di DARE; infatti, anche se il software é stato pensato avendo in mente la semplicitá di utilizzo, aluni passaggi legati alla gestione dei profili social dell'utente e alla navigazione della dashboard principale possono risultare complessi ad utenti non esperti.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Sentiment Analysis ed estrazione di conoscenza da social media

Nel primo capitolo dell'elaborato è presente un'ampia fase di ricerca sullo stato dell'arte della Sentiment Analysis. Essa inizia dall'analisi di ciò che è presente in letteratura e successivamente introduce diverse soluzioni tra le più popolari ed utilizzate, divise tra librerie open source e servizi offerti da provider cloud. È inoltre presente un breve confronto tra le due tipologie di soluzioni che risulterà cruciale nelle fasi successive del progetto. Infine, come sezione finale di questo capitolo, si riportano le difficoltà principali che il linguaggio utilizzato nei social media presenta per meglio capire dove potrebbero nascondersi problemi di accuratezza nella soluzione finale.

1.1 Stato dell'arte della Sentiment Analysis: librerie e servizi

1.1.1 Introduzione e funzionamento

La sentiment analysis è una specifica tecnica che utilizza algoritmi di Natural Language Processing (abbr. *NLP*) e Machine Learning (abbr. *ML*) con l' obiettivo di determinare il "sentimento" all'interno di un testo.
Generalmente il sentimento può essere classificato come *positivo*, *negativo* o *neutrale* (nessun sentimento rilevato).

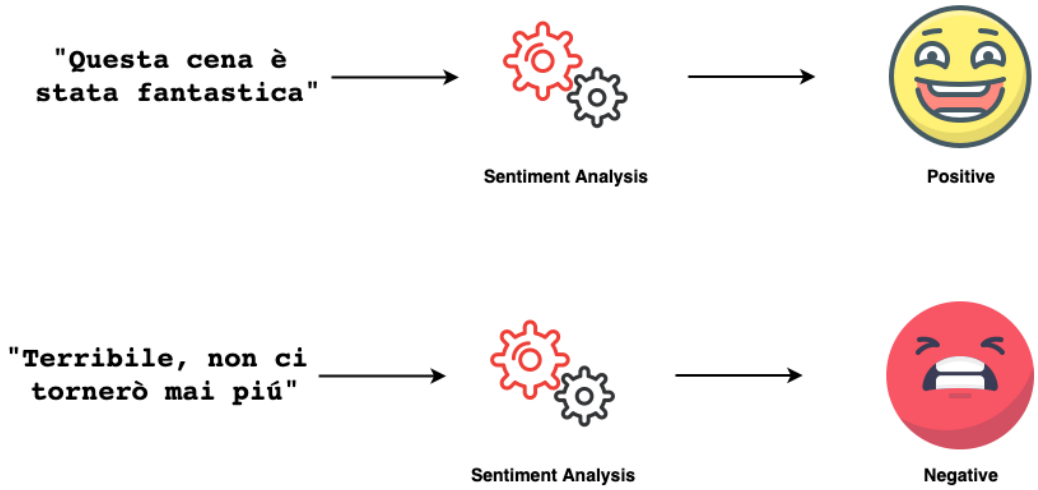

Figura 1.1: Schema illustrativo

Esistono tre tipologie fondamentali di algoritmi per la computazione del sentimento:

- rule-based (basato su tecniche di NLP);
- apprendimento automatico (basato su algoritmi di ML);
- soluzioni ibride.

Rule-based Sentiment Analysis

Per capire i principi del funzionamento della rule-based sentiment analysis è utile fare un paragone con il modo che tutti noi esseri umani utilizziamo per classificare i sentimenti. Data una frase infatti il nostro cervello esegue tre azioni fondamentali per determinarne il sentimento, più o meno simultaneamente.
La prima azione è ricercare i termini della frase all'interno del nostro dizionario mentale: per esempio, data la frase *"Ho passato una giornata fantastica"*, siamo in grado di estrarre il sentimento positivo grazie al termine *"fantastica"* perché nella nostra esperienza abbiamo associato a questo vocabolo una connotazione positiva.
La seconda azione è saper riconoscere i diversi gradi di sentimento: noi umani sappiamo che l'aggettivo *fantastico* ha una positività più marcata rispetto a *bello*, quindi attribuiamo una positività maggiore alla frase *"Ho passato una giornata fantastica"* rispetto a *"Ho passato una bella giornata"*.
Infine, eseguiamo un'associazione del sentimento all'entità della frase, andando alla ricerca del pronome a cui l'aggettivo si riferisce. Quindi, nell'esempio precedente, riusciamo a collegare *fantastica* a *giornata*, pronome della frase.

Nella rule-based Sentiment Analysis si eseguono praticamente le stesse azioni. Viene utilizzato un dizionario che al suo interno contiene migliaia di termini con un *"peso sentimentale"* associato. Più il peso è maggiore e più il termine ha una connotazione positiva e viceversa.
Come esempio si analizza *SentiWordNet* [17], un dizionario open source specifico per la Sentiment Analysis. Estraendo alcuni dati dalla tabella *"SentiWordNet 3.0.0.txt"*:

ID	PosScore	NegScore	SynsetTerms
00001740	0.125	0	able#1
00002098	0	0.75	unable#1
00037188	0.125	0.625	unused#3 idle#3

Tabella 1.1: Esempio di dizionario estratto da SentiWordNet

è possibile notare come ad ogni vocabolo (in `SynsetTerms`) sono associati un peso positivo (in `PosScore`) e uno negativo (`NegScore`).
Quello che viene svolto da un algoritmo rule-based è ricercare i vocaboli presenti nel testo all'interno del dizionario, estrarre il sentimento per ognuno di essi e calcolare il sentimento totale relativo all'intero testo o ad un'entità specifica. Il modo con cui il sentimento totale viene calcolato è deciso da regole stabilite da umani (data scientists, sviluppatori, ecc.), da ciò il nome rule-based. Tra le varie regole ci possono essere passaggi di data preprocessing, traduzione e tante altre tecniche specifiche.
Per esempio, data una qualsiasi frase, alcune regole per calcolare il sentimento potrebbero essere:

1. identificare le entità e gli aggettivi presenti (ad esempio tramite *part-of-speech tagging* [18]);

2. calcolare il sentimento per ogni singolo aggettivo (ad esempio utilizzando un dizionario come *SentiWordNet*);

3. calcolare la somma pesata dei sentimenti rilevati;

4. se la somma è positiva, ritornare sentimento positivo, altrimenti negativo.

La semplicità di questa tipologia di sentiment analysis la rende adatta all'analisi di testi standard in cui la struttura del testo non ha troppe variazioni, come ad esempio risposte a sondaggi.

Appena la complessità del testo cresce, tuttavia, questa tecnica diventa presto non utilizzabile, dato che è necessario prevedere delle regole per ogni tipo di testo in input (ad esempio abbreviazioni, emojis e qualsiasi tipo di variazione alla lingua naturale). Inoltre, risulta particolarmente difficile raggiungere un'elevata accurata nella *Aspect-based Sentiment Analysis* (introdotta successivamente nel paragrafo 1.2.1)

Apprendimento automatico

Al contrario dei sistemi rule-based, la sentiment analysis ad apprendimento automatico si basa su tecniche di Machine Learning. In questo caso, il problema di calcolare il sentimento è riconducibile ad un classico problema di classificazione [19]: le frasi con un sentimento positivo devono ritornare valore 1 (positivo) mentre le frasi a sentimento negativo devono ritornare valore -1 (negativo).
Il procedimento per sviluppare un modello in grado di fare ciò si basa quindi su un'iniziale fase di *allenamento*, in cui vengono fornite in input al modello frasi (opportunamente trasformate in vettori e matrici utilizzando tecniche come *bag-of-words* [20]) con un sentimento noto, con l'obiettivo di permettere al modello di imparare ad associare frasi al loro corrispondente risultato (sentimento). La seconda fase del procedimento è predire il risultato di una frase utilizzando il modello allenato.

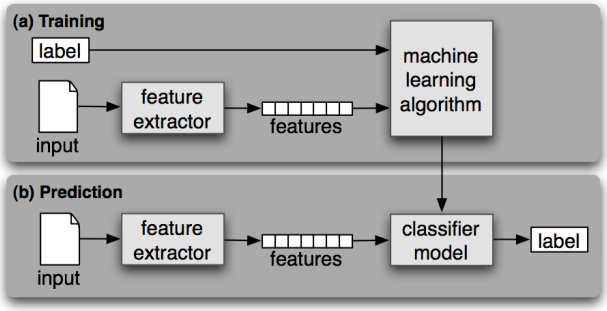

Figura 1.2: Classificazione supervisionata [21]

Il vantaggio fondamentale di applicare il Machine Learning per computare il sentimento è che un modello costantemente ri-allenato si adatta facilmente a variazioni del linguaggio naturale e quindi riesce generalmente a raggiungere un'accuratezza maggiore rispetto ai modelli rule-based. Il principale difetto consiste nel preparare i dati per la fase di training, poiché una persona fisica deve provvedere a classificare il sentimento per ogni input.

Soluzioni ibride

Le soluzioni ibride combinano Machine Learning e Natural Language Processing per ottenere i vantaggi di entrambi.
Ad esempio, utilizzare un dizionario per effettuare *part-of-speech tagging* risulta molto efficiente rispetto ad un modello di Machine Learning. Dove però un sistema rule-based richiede una continua introduzione di nuove regole e un aggiornamento del dizionario per gestire tutte le variazioni del linguaggio, un modello di Machine Learning può essere facilmente riallenato per dare significato a nuovi termini e modi di dire.

1.1.2 Librerie open source

Per sviluppare strumenti capaci di effettuare Sentiment Analysis sono presenti tantissime librerie open source accessibili a qualsiasi sviluppatore in una grande varietà di linguaggi (Python, R, Java, Javascript, ecc.). Esse offrono una grande flessibilità oltre alla possibilità di essere ampiamente personalizzate.
Di seguito vengono elencate e descritte alcune librerie tra le più comuni all'interno dell'ecosistema Python.

NLTK [22]

NLTK è la libreria di Natural Language Processing più popolare per Python, utilizzata sia in ambiti aziendali che di ricerca. È stata originariamente creata nel 2001 in un corso di computazione linguistica nel dipartimento di

Computer and Information Science presso la *University of Pennsylvania*, e da allora è stata costantemente migliorata attraverso tantissime contribuzioni.
Essa include l'accesso a più di 50 corpora [23] e dizionari [24], oltre a tantissimi altri metodi (*API*) per effettuare text processing. Con questa libreria è possibile implementare tutti e tre i tipi di sentiment analysis, dato che presenta API per effettuare rule-based Sentiment Analysis (PoS tagging, tokenization, dizionari, ecc.) ma anche classificatori utili ad allenare modelli di Machine Learning. È anche presente uno strumento rule-based già implementato chiamato VADER (descritto successivamente nella sezione 1.1.2).
Questa elevata disponibilità di informazioni e funzionalità la rendono complessa da utilizzare per chi si introduce per la prima volta al mondo del NLP. Per questo è interessante citare anche *TextBlob* [25], una libreria basata su NLTK e Pattern [26] che rende molto più intuitivo l'accesso alle varie API. Per quanto riguarda la Sentiment Analysis, essa presenta due tipi di "analizzatori" [27]: uno chiamato `PatternAnalyzer`, ereditato dalla libreria Pattern e di tipo rule-based, e uno chiamato `NaiveBayesAnalyzer` basato invece su un classificatore di tipo Naive Bayes [28] e quindi di tipo automatico.

spaCy [29]

spaCy è considerata come una libreria pensata principalmente per costruire applicazioni business (dal sito ufficiale: *"Industrial-Strength Natural Language Processing"*). Essa non presenta alcun modello pre-allenato per computare Sentiment Analysis, in quanto è pensata esclusivamente per NLP, ma offre diversi vantaggi rispetto a NLTK per realizzare un modello di Sentiment Analysis personalizzati a scopo industriale, tra cui:

- elevate performance (spaCy è costruita sul compilatore Cython [30]);
- possibilità di integrazione con framework per realizzare reti neurali (come TensorFlow o scikit-learn).

Secondo R. Tompson, CEO di MonkeyLearn [31] (azienda focalizzata ad offrire servizi SaaS di Text Analysis) [32], le modalità con cui si potrebbe procedere per la realizzazione di un modello di Sentiment Analysis sono:

1. preparazione del testo utilizzando le varie tecniche di NLP rese disponibili da spaCy tra cui Named Entity Recognition, PoS taggint, eccetera;

2. allenamento di un classificatore per classificare il sentimento.

VADER [33]

È stato scelto di includere VADER tra le librerie open source citate in quanto essa si è rivelata il perfetto esempio di un modello rule-based per la Sentiment Analysis. Inoltre, secondo il suo sviluppatore essa è specificatamente pensata per il linguaggio utilizzato nei social media, quindi presenta l'occasione di introdurre alcune problematiche e caratteristiche specifiche di questa particolare tipologia di testo (che verranno discusse ampiamente nella sezione 1.2.1).
È presente un dizionario che, oltre alla terminologia presente in un qualsiasi dizionario come *SentiWordNet*, contiene numerosi termini e modi di dire utilizzati prevalentemente nei social media:

```
SPECIAL_CASES = {"the bomb": 3, "bad ass": 1.5,
"badass": 1.5, "bus stop": 0.0, "yeah right": -2,
"kiss of death": -1.5, "to die for": 3, "beating heart": 3.1,
"broken heart": -2.9 }
```

Oltre a ciò, è incluso un intero dizionario per trasformare le emoji in testo [34] e numerosi termini speciali chiamati *Boosters* [35] che vengono comunemente usati per enfatizzare il sentimento.
Le regole che vengono eseguite per analizzare il sentimento della frase (tramite la funzione `SentimentIntensityAnalyzer.polarity_scores`) sono:

1. conversione delle emojis alla loro descrizione testuale;
2. tokenization e rimozione punteggiatura lasciando emoticons e contrazioni;
3. calcolo sentimento includente `Boosters`, modi di dire e negazioni (tramite la funzione `sentiment_valence`);
4. controllo di modifiche del sentimento dovute alla congiunzione *'but'*;
5. ritorno del sentimento rilevato nella forma

   `{"positive": , "negative": , "neutral": ,"compound": }.`

`"compound"` è la metrica più utile per avere una singola misura del sentimento della frase, e va da -1 (estremamente negativo) a +1 (estremamente positivo).

Le limitazioni di questa libreria sorgono dal fatto che essa è specificatamente pensata per la lingua inglese. È presente la possibilità di tradurre da altre lingue ma al giorno d'oggi la traduzione di detti e modi di dire da un linguaggio ad un altro non è possibile, quindi la precisione dello strumento diminuisce notevolmente. Inoltre, come già introdotto in precedenza, i sistemi rule-based non si adattano a cambiamenti del linguaggio naturale, e quindi l'introduzione di nuovi termini, detti e modi di dire non verrebbe considerata nel calcolo del sentimento senza aggiornare manualmente i dizionari.
Detto questo, lo strumento risulta un ottimo caso di studio per capire il funzionamento dei sistemi rule-based, e testandolo brevemente con frasi in inglese risulta anche decisamente accurato.

1.1.3 Sentiment Analysis in Cloud

Negli ultimi anni, oltre alle più convenzionali librerie di NLP e ML, si sono affermati nuovi servizi per accedere alle potenzialità dell'intelligenza artificiale senza doversi preoccupare in prima persona dello sviluppo di essi.

Questi servizi sono sviluppati direttamente dai provider che si preoccupano inoltre di allenarli correttamente e provisionare capacità per i modelli.
Questa evoluzione è stata resa possibile dal Cloud Computing, un trend molto in voga negli ultimi anni. Attraverso la rete, i provider cloud permettono di provisionare risorse hardware e servizi software on-demand. Questo ha innumerevoli vantaggi per aziende di qualsiasi dimensione, tra cui i più importanti:

- *possibilità di concentrare maggiori risorse direttamente sul prodotto*: l'azienda non ha bisogno di gestire risorse hardware in prima persona, quindi può concentrare un maggior numero di risorse sullo sviluppo del prodotto;

- *costi ridotti*: il cloud ha un modello di business on-demand, ovvero *"pay only for what you use"* (paga solamente quello che utilizzi). Questo, oltre al fatto che non bisogna fisicamente costruire aree in cui disporre le risorse hardware, permette di ridurre notevolmente i costi in anticipo;

- *affidabilità*: i provider Cloud vantano un'estrema affidabilità, dato che le risorse sono specialmente controllate. Detto questo, è sempre consigliato aggiungere ridondanza in modo da difendersi da eventuali periodi di disservizio.

Per quanto riguarda la Sentiment Analysis e più in generale il Natural Language Processing, tutti i maggiori provider cloud offrono servizi proprietari che permettono, tramite la chiamata ad un indirizzo specifico, l'analisi di testo alla ricerca di varie informazioni tra cui entità, temi presenti, lingua utilizzata e sentimento.
Per esempio, per utilizzare il servizio *Cloud Natural Language* offerto da Google, è sufficiente eseguire il seguente codice (in Javascript [36]):

```
1  const language = require("@google-cloud/language");
2  const client = new language.LanguageServiceClient();
```

```
3   const sentimentInput = 'Ho passato una gioranta fantastica'
        ;
4   const document = {
5     content: sentimentInput[i],
6     type: "PLAIN_TEXT",
7   };
8   const [result] = await client.analyzeSentiment({
9     document: document,
10  });
```

Il risultato contiene:

```
1 {
2   "documentSentiment": {
3     "score": 0.8999999761581421
4   },
5   "language": "it"
6 }
```

Nel caso di Google, il risultato contiene il sentimento nella forma di un numero nell'intervallo [-1 (negativo), 1 (positivo)]. Inoltre un'altra funzionalità molto importante che diversi provider offrono è la possibilità di allenare un modello personalizzato per adattarlo direttamente ad un dominio specifico. Ciò è realizzabile senza avere conoscenza avanzata relative a NLP o ML.
Per maggiori dettagli sulle funzionalità offerte da ciascun provider, si invita a consultare la sezione 2.2.2.

1.1.4 Confronto tra i due approcci

I due approcci, seppure in grado di risolvere lo stesso problema, presentano numerose differenze concettuali che rendono ognuno adatto a specifici casi d'uso.
L'utilizzo di una libreria open source è pensato per l'utilizzo da parte di un esperto con una conoscenza avanzata di NLP, allenamento di classificatori e nozioni di reti neurali. Per questo tipo di utenti, queste librerie offrono tan-

tissime possibilità di personalizzazione e modifica di parametri ed algoritmi. Questo non è valido per i software in cloud citati. Essi non richiedono nessuna conoscenza pregressa relativa a ML o NLP e permettono a utenti e sviluppatori di creare prodotti molto più velocemente in quanto è sufficiente attivare il servizio ed effettuare una chiamata di rete all'indirizzo specificato. Oltre a questa differenza sostanziale, un altro punto che rende i due approcci molto lontani fra loro è l'architettura richiesta per renderli operativi.
Un modello basato su una delle librerie open source (se basato su apprendimento automatico o ibrido) richiede una fase di allenamento in cui viene addestrato a riconoscere il sentimento all'interno di frasi in input. Questo richiede notevoli risorse hardware (in particolare risorse grafiche) e, in base alla quantità di dati di allenamento, può durare diverse ore. Nel caso non si avesse a disposizione un hardware proprietario, "affittare" risorse in Cloud (per esempio attraverso Amazon Web Services o Google Cloud) è l'unica opzione, e ciò è molto costoso. In aggiunta, il modello richiede comunque delle risorse fisiche in cui eseguire che devono essere in grado di soddisfare il numero di frasi da analizzare. Tutto ciò si traduce in costi iniziali piuttosto elevati. Il secondo approccio invece ha un modello di costo completamente differente: esso permette agli utilizzatori di non doversi preoccupare di provisionare risorse in quanto ciò è automaticamente gestito dal provider, riducendo la quantità di lavoro iniziale ed eventuale necessità di aumentare capacità. Ma la differenza sostanziale è che il modello in cloud viene continuamente ri-allenato e migliorato dal provider, rendendolo capace di riconoscere ogni variazione del linguaggio inclusa nei dati di allenamento (gestito direttamente dal provider). Ciò non è valido nel caso in cui si decidesse di allenare manualmente un modello su frasi personalizzate inerenti ad uno specifico dominio (come descritto nel paragrafo precedente).
Per un'analisi più accurata delle decisioni che sono state prese in relazione al progetto DARE-SAM, si invita a consultare la sezione di progettazione 2.2.2.

1.2 Sentiment Analysis applicata ai social media

Come già introdotto in precedenza, applicare la Sentiment Analysis ai social media sta riscontrando molteplici utilizzi che permettono di avere nuovi punti di vista ed informazioni a supporto delle decisioni. Il range di applicazioni spazia da business e marketing a politica e sanità. Tra alcuni dei casi d'uso più interessanti che si sono potuti analizzare [37] è possibile citare:

- framework per analizzare proprietà spazio-temporali allo scopo di identificare posizione di possibili focolai [38];
- analisi di contenuti pubblicati da persone per evidenziare il loro livello di depressione [39];
- predizione di elezioni politiche [40];
- monitoring di un prodotto, servizio o brand;
- valutazione di decisioni intraprese da aziende private ed enti pubblici.

1.2.1 Problemi e soluzioni del linguaggio utilizzato nei social media

Tutte le applicazioni della sentiment analysis si basano sull'avere a disposizione uno strumento abbastanza acccurato in grado di fornire predizioni corrette il maggior numero di volte possibile. Questo aspetto non è scontato per testi provenienti da social media in quanto essi presentano diversi problemi e difficoltà relative al linguaggio utilizzato. Di seguito sono presentati alcuni dei problemi principali che un modello deve saper risolvere per poter produrre risultati sufficienti analizzando testi raccolti dalle maggiori piattaforme social.

Abbreviazioni e modi di dire

Nel linguaggio comunemente utilizzato sui social media sono presenti numerosi modi di dire ed abbreviazioni che non sono comunemente utilizzati nel linguaggio classico. Questo deve essere correttamente preso in considerazione dal modello in quanto questi termini possono molto spesso cambiare significato o sentimento della frase. Per dimostrare ciò è riportato un esempio di modo di dire della lingua inglese: *"the bomb"*. Questo modo di dire indica che l'entità a cui è riferito è fantastica [41] e quindi ha una connotazione positiva. Per questo il risultato dell'analisi della frase *"This song is the bomb"* dovrebbe essere estremamente positivo. Analizzando il risultato dell'analisi prodotta da VADER (introdotto in 1.1.2):

```
This song is the bomb.-----------------------------------
{'neg': 0.0, 'neu': 0.5, 'pos': 0.5, 'compound': 0.6124}
```

esso è in linea con quello atteso (il risultato `'compound'` è positivo). Si può notare però nel file sorgente di VADER (`vaderSentiment.py`) alla linea 78:

```
SPECIAL_CASES = {"the bomb": 3, ... "kiss of death": -1.5,
"to die for": 3, "beating heart": 3.1, "broken heart": -2.9 }
```

che questo caso è stato introdotto manualmente dallo sviluppatore. Provando a rimuoverlo, il risultato cambia:

```
This song is the bomb.-----------------------------------
{'neg': 0.444, 'neu': 0.556, 'pos': 0.0, 'compound': -0.4939}
```

Esso è ora diventato negativo (`compound score` negativo), probabilmente perché il termine *"bomb"* ha di base una connotazione negativa.
È possibile notare quanto sia importante realizzare un modello capace di comprendere modi di dire ed abbreviazioni, in quanto essi sono estremamente utilizzati nei social media.
Con un sistema rule-based, ciò richiede molto lavoro in quanto bisogna aggiungere manualmente ogni caso, mentre un sistema ad apprendimento automatico è generalmente in grado di imparare autonomamente nuovi modi di dire se essi sono presenti e correttamente gestiti nei dati di allenamento.

Errori di spelling

Spesso il testo proveniente da social media presenta errori di spelling dovuti alla fretta con cui generalmente si procede a scrivere contenuti. Questo è un problema per un modello in quanto esso potrebbe non comprendere alcuni termini con errori di spelling e quindi non utilizzarli all'interno della computazione del sentimento.
Una soluzione possibile è quella di effettuare uno step di preprocessing in cui si utilizza uno *spell checker*, ovvero un particolare algoritmo in grado di risolvere errori di spelling. L'algoritmo più popolare è quello descritto da P. Norvig in *"How to Write a Spelling Corrector"* [42] che, come cita l'autore, è in grado di raggiungere un'accuratezza del 80-90%. Questo algoritmo è implementato in alcune librerie di Natural Language Processing, tra cui le già citate TextBlob e spaCy.

Emoji

L'uso delle Emoji è rapidamente aumentato negli ultimi anni grazie alla crescente popolarità dei social network. Esse permettono agli utenti di esprimere le proprie emozioni più facilmente e per questo sono diventate ormai una componente fondamentale di qualsiasi testo presente sui social media. Per esempio, già nel 2015 la piattaforma Instagram dichiarava che il 50% dei testi pubblicati conteneva emoji [43]. È facile capire quindi che esse sono parte fondamentale del sentimento espresso all'interno di un testo e perciò devono essere correttamente tenute in considerazione. Prendendo in esempio VADER, è possibile notare che la libreria trasforma le emoji in testo utilizzando un dizionario specifico (`emoji_utf8_lexicon.txt`) in cui ad ogni emoji è associata una descrizione testuale. Un'altra modalità con cui si può procedere è allenare un modello di Machine Learning a riconoscere il sentimento espresso dalle emoji, come proposto nello studio svolto da Novak, Smailovic, Sluban e Mozetic [44]. Questo procedimento è però molto oneroso in quanto richiede un grande numero di dati di allenamento per poter essere accurato.

Utilizzo di più lingue

Non è raro riscontrare contenuti in più di una lingua sui social media. Se infatti si utilizza come esempio un personaggio di spicco come un politico o un cantante, è immediato notare come all'interno dei commenti ricevuti sotto i post ci sono certamente persone provenienti da tutto il mondo.
Questo causa un problema nella creazione di un modello affidabile in quanto diverse lingue devono essere gestite diversamente. Nel caso dei modelli rule-based, infatti, sorge la necessità di avere un dizionario per ogni lingua supportata. Oltre a ciò, è necessario avere uno step di pre-processing per rilevare la lingua del testo. Per quest'ultimo step, ci sono diverse librerie in grado di rilevare la lingua con un'accuratezza estremamente alta (più di 99%). Esse si basano sul riconoscere specifici pattern e regole di spelling specifiche ad ogni linguaggio. Un esempio è *language-detection*, sviluppata da Google [45]. Una volta che la lingua del testo è stata rilevata, si può procedere all'analisi del sentimento utilizzando lo specifico dizionario. Se si usano modelli di Machine Learning, invece, essi devono essere specificatamente allenati in modo da analizzare più lingue.

Sarcasmo

Il sarcasmo consiste nell'esprimere sentimenti negativi utilizzando termini positivi. Questo costituisce una grande difficoltà per i modelli di Sentiment Analysis (ma anche per gli esseri umani) in quanto è difficile da rilevare senza avere il giusto contesto. Esistono vari modi per rilevare il sarcasmo all'interno dei testi, e come per la maggior parte dei problemi si dividono in rule-based e modelli ad apprendimento automatico. In generale non è presente molto in letteratura su questo caso specifico, ma uno studio effettuato da A.G. Prasad [46] indica come sia possibile raggiungere un'accuratezza dell'80% utilizzando vari classificatori come Gradient Booster [47] e Random Forest [48].

Aspect-based Sentiment Analysis

L' Aspect-based sentiment analysis (abbr. *ABSA*) mira ad identificare il sentimento riferito a una specifica entità (aspect) in una frase. Per esempio, nella frase *"Mi è piaciuto veramente tanto questo film, peccato che l'attore principale non sia all'altezza"* un modello in grado di effettuare ABSA è capace di riconoscere il sentimento positivo associato al film e quello negativo riferito all'attore principale.
All'interno di uno studio condotto da M. Hoang [49] è descritto un metodo per computare ABSA in grado di superare i risultati ottenuti precedentemente e considerati stato dell'arte. Inoltre diversi provider Cloud, in particolare Microsoft con Text Analytics [50] e Google con Cloud Natural Language [51], stanno iniziando ad offrire questa possibilità. Purtroppo nessuno dei due attualmente supporta la lingua italiana (al 15 Febbraio 2021) e quindi non potranno essere utilizzati direttamente all'interno di questo elaborato.

Capitolo 2

Il progetto DARE-SAM: requisiti e progettazione

In questo secondo capitolo si affronta la fase più delicata del progetto: la progettazione.
Inizialmente vengono elencati e descritti i requisiti di sistema (funzionali e non funzionali) e successivamente si introducono diverse sfide che richiedono una complessa fase di ricerca. Infine si passa ad una sezione dedicata alla descrizione delle tecnologie principali utilizzate per la realizzazione di DARE-SAM, elencando per ognuna i vantaggi che offre.

2.1 Requisiti di sistema

Il progetto DARE-SAM (in forma estesa *DARE Sentiment Analysis and Monitoring*) si pone come obiettivo quello di realizzare una piattaforma software che permetta a gestori di pagine social (Facebook e Instagram) di visualizzare dashboard e grafici con dati aggregati sulle opinioni e sentimenti rilevati nelle interazioni con i contenuti da essi pubblicati sulle suddette piattaforme. È importante da subito sottolineare come le possibilità di elaborazione di questa particolare tipologia di dati e la conseguente visualizzazione di informazioni utili all'utente finale sono estremamente numerose e non è

possibile considerare ognuna di esse all'interno di questo elaborato.
Di seguito sono presenti i requisiti funzionali che il software dovrà soddisfare. Essi sono divisi in tre principali categorie:

- requisiti relativi alla gestione dell'utente (iscrizione, collegamento di profili social);
- requisiti relativi all'analisi ed estrazione di dati;
- requisiti relativi alla visualizzazione dei dati.

Partendo dalla prima categoria, i principali requisiti relativi alla gestione dell'utente sono:

- *iscrizione di un utente al sistema*: un utente deve potersi registrare e creare un account all'interno del sistema; esso permette di accedere alle funzionalità della piattaforma da più dispositivi e di garantire la sicurezza e l'isolamento dei dati presenti;
- *collegamento del profilo Facebook per la registrazione delle pagine social*: un utente può collegare il proprio profilo Facebook alla piattaforma in modo da poter analizzare i suoi profili, ovvero pagine Facebook e profili Instagram in cui egli ha un ruolo attivo (Amministratore, Editore, Moderatore) [52].

Procedendo alla seconda categoria, i requisiti fondamentali per quanto riguarda l'analisi ed estrazione di dati sono:

- *analisi di una pagina Facebook associata al profilo Facebook di un utente*: la piattaforma deve poter analizzare i contenuti presenti su una pagina Facebook a cui è stato dato accesso da un utente per estrarre informazioni strategiche relative a varie metriche di interesse;
- *analisi di un profilo Instagram associato al profilo Facebook di un utente*: la piattaforma deve poter analizzare i contenuti presenti su un profilo Instagram a cui è stato dato accesso da un utente per estrarre informazioni strategiche relative a varie metriche di interesse;

- *estrazione di dati relativi a ogni singolo post pubblicato da profili social*: la piattaforma deve essere in grado di estrarre e periodicamente aggiornare i dati relativi ai profili collegati; in dettaglio, per ogni post pubblicato da una pagina Facebook collegata ad un utente essa deve raccogliere le seguenti metriche: data di pubblicazione, contenuto del post (testo, immagini, ecc.), numero di like, numero di condivisioni, numero di commenti;

- *estrazione di dati relativi ai commenti ricevuti da contenuti pubblicati da profili social*: la piattaforma deve estrarre e periodicamente aggiornare i commenti ricevuti per ogni post pubblicato da profili social; per ogni commento sono di interesse: data di pubblicazione, testo del commento, numero di like e numero di risposte al commento;

- *analisi di ogni post e relativi commenti alla ricerca del sentimento presente nel testo*: la piattaforma deve essere in grado di analizzare il testo estratto dai vari profili (contenuto in post e commenti) alla ricerca del sentimento che esso esprime.

Infine, il requisito relativo alla visualizzazione dei dati consiste nella *creazione di una dashboard principale con informazioni utili all'utente*. Questa dashboard dovrà includere numerosi dati per ogni post pubblicato da una pagina social, tra cui:

- data di pubblicazione del post;
- messaggio del post;
- sentimento rilevato (se presente);
- numero di like ricevuti;
- numero di condivisioni del post (non applicabile ad Instagram);
- numero di commenti ricevuti;
- sentimento aggregato dei commenti ricevuti.

Inoltre, deve essere possibile espandere un singolo contenuto per analizzare quelli che sono i commenti ricevuti e per ognuno di essi visualizzare:

- data di pubblicazione del commento;
- messaggio del commento;
- numero di like ricevuti;
- sentimento rilevato (se presente).

Sarebbe infine molto utile avere la possibilità di filtrare i contenuti in modo che l'utente possa raggiungere con più rapidità le informazioni che sta cercando.
Passando ora brevemente ai requisiti non funzionali, sono di particolare interesse:

- *sicurezza*: dato che l'account di un utente contiene informazioni sensibili, la sicurezza del sistema è un punto fondamentale che deve essere preso in considerazione in fase di progettazione;
- *semplicità di utilizzo*: sebbene il software sia pensato per utenti con una buona conoscenza tecnologica nell'ambito dei social media, esso deve comunque risultare semplice da utilizzare e offrire le informazioni che gli utenti necessitano nel modo più efficace possibile;
- *efficienza*: il sistema deve essere in grado di soddisfare i requisiti con il minor costo possibile in termini di risorse economiche.

2.2 Ricerca e progettazione

Nella fase di ricerca e progettazione del prodotto finale, già da subito l'idea di una piattaforma web si è rivelata l'opzione migliore in quanto l'elevata quantità di informazioni da presentare all'utente non sono facilmente visualizzabili da uno schermo a ridotte dimensioni come quello di uno smartphone.

Inoltre, a causa delle sfide proposte dall'estrazione dei dati dalle piattaforme di social media (descritte esaustivamente nella prossima sezione), è necessario realizzare un'architettura particolare pensata appositamente per questo caso d'uso.

2.2.1 Come estrarre i dati dalle piattaforme social

Una importante fase della progettazione del software consiste nel progettare le modalità con cui verranno estratti i dati dai social network. Questa operazione è fondamentale e richiede un'accurata analisi.
Dai requisiti si evince che, in una prima versione del progetto, i due social media da supportare sono Facebook e Instagram. Un'ipotesi iniziale è quella del Web Scraping [53], ovvero estrarre i dati dalla pagina web del social network in maniera automatica leggendo il codice sorgente HTML tramite un bot o un web crawler [54]. Ciò è stato subito scartato in quanto comporta una violazione dei termini di servizio di Facebook [55] (Section 3.2.3: *"You may not access or collect data from our Products using automated means (without our prior permission) ..."*), ma anche a causa del fatto che, essendo il codice HTML di un sito come Facebook costantemente aggiornato e modificato, realizzare un crawler robusto richiederebbe un costante lavoro di aggiornamento in modo da adattarlo alle modifiche.
La soluzione più efficace risulta quindi utilizzare l'API pubblica di Facebook chiamata *Facebook Graph API* [56]. Essa è il modo primario con cui è possibile leggere e pubblicare contenuti in maniera automatica all'interno di Facebook. È un'API basata su HTTP utilizzabile per svariati compiti tra cui ciò che è richiesto da questo progetto, ovvero leggere dati e informazioni pubblicate sui social media. Inoltre, essendo Instagram proprietà di Facebook, è anch'esso accedibile tramite Graph API (*Instagram Graph API* [57]) e ciò semplifica significativamente l'estrazione dei dati dalle due piattaforme. L'API presenta però alcune sfide e requisiti da soddisfare per poter essere utilizzata in maniera efficace. Prima di tutto, bisogna registrare un'applicazione tramite Facebook for Developers [58] per poter autorizzare utenti. Ciò

permette ad utenti Facebook di autorizzare quest'ultima a fornire informazioni personali all'applicazione. Successivamente, per avere accesso a pagine Facebook o profili Instagram bisogna ottenere un token di accesso Facebook (*Facebook Access Token*). Questo token permette di conoscere l'identità dell'utente che effettua la richiesta e va aggiunto all'URL ad ogni richesta HTTP a Graph API .
Per poter accedere ai dati relativi ad una pagina Facebook gestita dall'utente sono necessari permessi particolari. In dettaglio [59]:

- `pages_show_list`: permette l'accesso alla lista delle pagine Facebook che l'utente gestisce;
- `pages_read_engagement`: permette di leggere contenuti (post, immagini, video ed eventi) pubblicati da una certa pagina Facebook;
- `pages_read_user_content`: permette di leggere contenuti generati dall'utente sulla pagina Facebook (posts, commenti e recensioni).

Per i profili Instagram, invece, è necessario il permesso `instagram_basic` [60], che permette di leggere informazioni relative al profilo Instagram e media pubblicati (post, commenti, ecc.). Deve quindi essere implementata una soluzione in grado di garantire l'accesso tramite profilo Facebook con i permessi elencati precedentemente.
Una importante limitazione che Facebook impone a qualsiasi suo utente sviluppatore è il limite massimo di chiamate che ciascun utente può effettuare in un certo periodo di tempo. Nel caso di Graph API, la limitazione è di 200 chiamate per ora per utente [61], da intendersi per applicazione. Per chiarire meglio questo aspetto è utile introdurre un esempio: se un'app ha 10 utenti, essa ha diritto a 2.000 chiamate all'ora a Graph API che ipoteticamente potrebbero anche essere effettuate da un solo utente; l'importante è che la somma di chiamate effettuate da ciascun utente non superi il limite totale (in questo caso 2.000).
Questo ha introdotto una sfida aggiuntiva, in quanto richiedere le informazioni che la piattaforma ha bisogno può a volte necessitare svariate chiamate

all'API e potenzialmente superare il limite imposto. Inoltre, risulta necessario trovare una modalità efficiente con cui estrarre periodicamente i dati e aggiornarli in background, in quanto effettuare tutte le operazioni di estrazione dei dati e analisi degli stessi ad ogni richiesta dell'utente non è ideale. La soluzione pensata è quella di un *Background Service* che periodicamente richiede i dati a Graph API e li aggiunge/aggiorna all'interno del database. L'intervallo con cui il Service dovrebbe eseguire è idealmente di un'ora per sfruttare il massimo numero di chiamate offerte da Graph API senza superare il limite. Se il limite di chiamate orarie viene superato, Facebook agisce bloccando per più tempo la possibilità di effettuarne di nuove (citando Facebook: *When the limit has been reached, stop making API calls. Continuing to make calls will continue to increase your call count, which will increase the time before calls will be successful again* [61]).

Breve nota sugli aspetti legali

Facebook Graph API (e di conseguenza Instagram Graph API) ha un complesso set di termini e regole (*Platform Terms* [62]) che vanno rispettate per poter utilizzare il servizio. Le sezioni più importanti relative al caso d'uso di DARE-SAM sono la numero *3. Data Use* e la numero *6. Data Security*.
Analizzando la prima, non sono presenti limitazioni relative all'utilizzo dei dati a scopo di *"business intelligence"*. L'unica nota importante applicabile al progetto è la *3.v*: esso vieta di esporre *Platform Data* a motori di ricerca e web crawler che potrebbero accidentalmente indicizzarli. Ciò richiederà quindi una particolare attenzione durante lo sviluppo.
Passando ora alla sezione relativa alla sicurezza dei dati, Facebook obbliga gli sviluppatori a rendere sicuri i dati e di assicurarsi che persone non autorizzate non possano accederci. Anche questo aspetto deve essere considerato attentamente in fase di implementazione.
L'utilizzo del servizio da parte di DARE-SAM non presenta alcuna violazione dei termini imposti ed è quindi possibile sfruttare i contenuti che esso offre per soddisfare i requisiti elencati in precedenza (incluso l'utilizzo di Senti-

ment Analysis per l'analisi dei testi).
Oltre a rispettare i termini di servizio, Facebook richiede un processo di verifica dell'applicazione creata [63] per autorizzarne l'utilizzo da parte di utenti non sviluppatori o tester ([64]). Esso richiede all'utente numerose azioni (descritte esaustivamente nelle pagine *"Before you submit"* [65] e *"Submitting For Review"* [66]), alcune relativamente semplici da eseguire (come caricare registrazioni che mostrano le funzionalità del prodotto) mentre altre notevolmente complesse. Tra queste ultime spicca la necessità di ottenere la *Privacy Policy* relativa al prodotto che utilizza l'applicazione, e ciò richiede competenze specifiche in materia legale.
Il processo di verifica dell'applicazione Facebook che verrà utilizzata per DARE-SAM sarà quindi affrontato in uno sviluppo futuro del progetto, in quanto prima di ciò è necessario verificare che le funzionalità relative a Facebook e Instagram siano ultimate (eventuali cambiamenti richiederebbero una verifica ulteriore e un'aggiornamento della *Privacy Policy*).

2.2.2 Servizi Cloud per la Sentiment Analysis

Una delle parti fondamentali del progetto consiste nell'analisi dei dati provenienti dai social media per interpretarli ed estrarre conoscenza. Alcuni dei dati ottenibili, come numero di like e condivisioni, non richiedono elaborazione in quanto sono in forma numerica, mentre altri, come ad esempio il testo dei commenti, richiedono una fase addizionale per poter interpretarne il significato. Dato che l'interesse del progetto è quello di estrarre opinioni dei cittadini Ravennati, risulta essenziale estrarre il sentimento contenuto nei testi e messaggi ricevuti. Per fare ciò esistono due approcci fondamentali ampiamente descritti nel Capitolo 1.
Durante l'analisi, è emerso già da subito come non fosse possibile procedere all'implementazione di una soluzione di Sentiment Analysis *ad-hoc* adatta appositamente al nostro caso d'uso in quanto l'implementazione del software richede già un lavoro sostanzioso. Per questo motivo si è scelto di procedere con l'utilizzo di uno dei provider cloud che offrono soluzioni di Natural Lan-

guage Processing e Machine Learning. Questo permette di avere un prodotto iniziale completamente funzionante e successivamente estenderlo aggiungendo fasi di pre-processing del testo o addestramento di modelli personalizzati. I 4 competitors principali in questo ambito sono:

- Google Cloud Platform, con il servizio *Cloud Natural Language*;
- Microsoft Azure, con il servizio *Azure Text Analytics*;
- IBM Cloud, con il servizio Watson *Natural Language Understanding*;
- Amazon Web Services, con il servizio *Amazon Comprehend*.

Tutti e 4 presentano le seguenti funzionalità che potrebbero rivelarsi utili nell'ambito del progetto DARE-SAM:

- *Sentiment Analysis*: estrazione del sentimento da un testo e categorizzazione in positivo, negativo o neutro;
- *Entity Extration*: estrazione di entità (nomi propri di persona, brand, aziende, organizzazioni, ecc.);
- *Language Detection*: utile per gestire automaticamente contenuti in lingue diverse.

Si procede ora all'analisi delle funzionalità uniche di ciascun provider ponendo particolare attenzione sui costi e sulle eventuali limitazioni per sviluppi futuri. L'analisi dei costi risulta una parte importante in quanto l'efficienza del sistema è un requisito non funzionale del sistema. Per fare un calcolo approssimativo dei costi si è presa in esempio la pagina Facebook del sindaco di Ravenna, Michele de Pascale [67], in quanto essa riceve un numero di commenti significativo che potrebbe essere confrontabile al numero di commenti medio che un utente di DARE-SAM potrebbe ricevere. Secondo i dati raccolti a Dicembre 2020, il numero medio di commenti che la pagina riceve per ogni post pubblicato è 53, con una lunghezza media di caratteri per commento di 150. Ipotizzando un totale di 60 post pubblicati per mese (circa

1 al giorno per piattaforma social supportata), il risultato è di circa 3.300 commenti ricevuti per un totale di circa 500.000 caratteri al mese.

Google Cloud Natural Language [68]

Google Cloud Natural Language è il servizio principale offerto da Google per l'analisi testuale alla ricerca di struttura e significato. Oltre alle funzionalità già introdotte in precedenza e comuni a tutti i provider, Google introduce la cosiddetta *"Entity Sentiment Analysis"* [69], ovvero una combinazione di Sentiment Analysis ed Entity Extraction che mira a determinare il sentimento associato a ciascuna entità rilevata nel testo. Questa funzionalità purtroppo non è ancora disponibile per analisi di testo in lingua Italiana (al 15 Febbraio 2021) [70], altrimenti sarebbe stata estremamente importante per avere una maggiore quantità di informazioni. Oltre a ciò, il servizio offre la possibilità di realizzare un modello custom per riconoscere entità personalizzate e altri tipi di testo, perciò permette di estendere le capacità del servizio nel caso se ne verifichi la necessità. Per quanto riguarda il prezzo, Google Cloud Natural Language propone un sistema di pricing a unità chiamata *"unit"* [71], ovvero una parte di testo con lunghezza massima di 1.000 caratteri. Ciò causa la perdita di numerosi caratteri per ogni richiesta, dato che la lunghezza media del testo da analizzare si aggira attorno ai 150. Il costo finale per analizzare tutti i commenti ricevuti è quindi di circa $ 3.30 per mese, calcolato come 3.300 commenti per $ 1 ogni 1.000 unità.

Microsoft Azure Text Analytics [72]

Funzionalità fondamentale che il servizio offerto da Microsoft offre è quello di *"Opinion mining"* [50], molto simile alla *"Entity Sentiment Analysis"* di Google. Essa infatti permette di associare un sentimento ad ogni entità presente nel testo, ma purtroppo non è ancora presente la lingua Italiana (al 15 Febbraio 2021) [73]. A differenza di Google, Azure non supporta la creazione di un modello custom e questo potrebbe comportare una limitazione nell'estendibilità del progetto. Per il costo del servizio, anche Azure come

Google ha un concetto di unità chiamato *"text record"* [74] di 1.000 caratteri e il costo finale è uguale a quello offerto da Google, ovvero $ 3.30 per 3.300 commenti mensili.

IBM Watson Natural Language Understanding [75]

IBM offre diverse funzionalità molto interessanti, tra cui l'estrazione delle emozioni (gioia, rabbia, tristezza, ecc.) e l'analisi del testo alla ricerca del sentimento specifico a delle determinate entità (denominate *target*) [76]. La prima putroppo non supporta l'Italiano (al 15 Febbraio 2021) [77], mentre la seconda è supportata e potrebbe essere interessante nel nostro caso per determinare il sentimento specifico di una particolare entità, come appunto la Darsena. Oltre a ciò, IBM offre la possibilità di creare modelli custom e ha un Free Tier pricing molto competitivo che potrebbe soddisfare il nostro caso d'uso iniziale. Infatti, l'unità di misura utilizzata è la *"Data Item"* [78] che corrisponde a 10.000 caratteri da moltiplicare per il numero di funzionalità richieste (1 nel caso di sola Sentiment Analysis). IBM offre 30.000 *"Item"* al mese nel Free Tier plan, più che sufficienti per coprire l'utilizzo medio di un singolo utente. Se questo non dovesse bastare, però, IBM ha un prezzo molto elevato dovuto al grande numero di caratteri inclusi in un *Data Item*. Infatti, ad ogni commento di circa 150 caratteri, si sprecano di fatto sprecando 9850 caratteri che vengono comunque inclusi nel prezzo, quindi il costo mensile previsto per IBM escludendo il già citato Free Tier è di circa $ 100, ottenuto moltiplicando 3.300 commenti per $ 0.03/*Data Item*.

Amazon Comprehend [79]

Amazon non offre funzionalità aggiuntive rispetto a quelle comuni a tutti i provider, se non la possibilità di creare un modello addestrato a riconoscere entità personalizzate. Esso vanta però un modello di pricing molto aggressivo che lo posiziona come competitor più economico sul caso d'uso di DARE-SAM. Esso ha come unità di pricing una *"unit"* di 100 caratteri [80], e per questo motivo permette di sprecare meno caratteri possibili rispetto

alla lunghezza media. Ciò si traduce in un costo medio stimato di $ 0.66 per mese, calcolato come 3.300 commenti per 2 (dato che 150 caratteri vengono considerati come 2 units) per $ 0.0001/unit.

Per riassumere e semplificare la lettura delle informazioni è presente una tabella riassuntiva (2.1):

Provider	**Funzionalità extra**	**Realizzare modelli custom**	**Prezzo ***
Google Cloud	Entity sentiment	Si	$ 3.30
Azure	Opinion mining	No	$ 3.30
IBM Watson	Analisi di emozioni, sentimento verso entità target	Si	$ 100
Amazon		Si	$ 0.66

Tabella 2.1: Tabella riassuntiva delle funzionalità legate alla Sentiment Analysis offerte dai principali Provider Cloud sul mercato

* = riferito all'esempio utilizzato in precedenza.

È evidente come tutte e 4 le proposte siano interessanti per motivi diversi e perciò la scelta finale per decidere quale provider utilizzare verrà discussa ampiamente nel capitolo dedicato all'implementazione (3.2).

2.3 Principali tecnologie utilizzate

In questa sezione verranno elencate quelle che sono le tecnologie principali utilizzate nella realizzazione della piattaforma, divise in 4 macro categorie:

- tecnologie presenti all'interno di ciascun componente dell'infrastruttura, raccolte nella sottosezione *"Introduzione alla Codebase"* 2.3.1;

- tecnologie relative al Client Web 3.1;
- tecnologie relative al Backend Service 3.1.

2.3.1 Introduzione alla codebase

Il codice sorgente di DARE-SAM è completamente open source e visitabile all'indirizzo `https://github.com/Gobbees/dare-sam`.

TypeScript

Il linguaggio di programmazione che è risultato ideale per la realizzazione del software è TypeScript [81], progetto sviluppato da Microsoft che ha come funzionalità principale quella di aggiungere i tipi a JavaScript [36]. È ben noto infatti come uno dei problemi principali di quest'ultimo sia la mancanza di tipi, in quanto ciò causa molti errori non facilmente rilevabili in fase di compilazione ma solo a runtime. Un altro aspetto molto utile che TypeScript introduce è quello di rendere l'ambiente di sviluppo molto più potente grazie ai suggerimenti sui tipi che software come Visual Studio Code [82] e Webstorm [83] sono in grado di offrire.
TypeScript mantiene però tutti i vantaggi di JavaScript, primo fra tutti la versatilità. Infatti, l'ecosistema JavaScript permette di realizzare applicativi web (scopo principale per cui il linguaggio di sviluppo è stato realizzato), mobile (grazie a framework come React Native [84]), web servers (Node.JS [16]), applicazioni di machine learning (con librerie come Tensorflow.JS [85]), realtà virtuale e tanto altro ancora. L'approccio che TypeScript ha adottato è quello di *"Gradual Adoption"*. Esso è in grado di convivere con JavaScript per offrire agli sviluppatori un modo facile per testarlo ed eventualmente convertire interamente la propria codebase. Ciò ha permesso a TypeScript di essere adottato su larga scala da aziende enormi del calibro di Google, Facebook e tante altre.

2.3.2 Client Web

Le tecnologie utilizzate per realizzare il Client Web sono innumerevoli e, per semplicità, verranno elencate e descritte solo le principali.

React [14]

React è una libreria realizzata da Facebook e resa pubblica nel 2014 che ha come obiettivo quello di realizzare interfacce grafiche. Essa introduce diverse novità che si sono rivelate vincenti nel panorama dello sviluppo web, dato che come è possibile notare nella figura 2.1, React è la libreria legata allo sviluppo front-end più scaricata ed utilizzata.

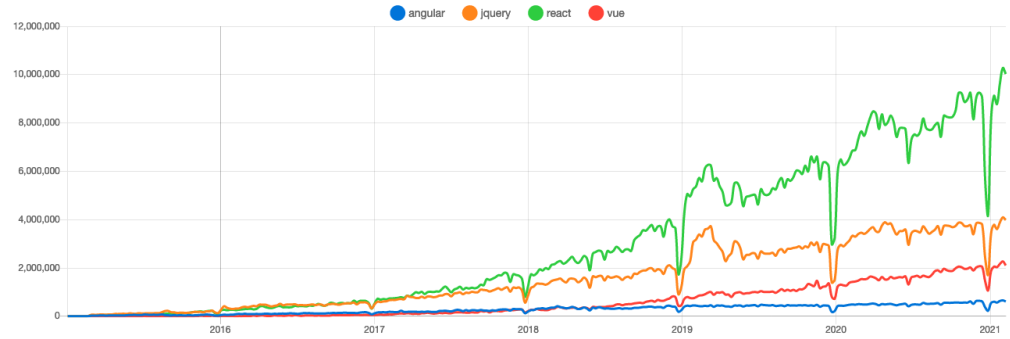

Figura 2.1: Numero di download aggregato per le maggiori librerie per lo sviluppo di applicativi web complessi [86]

Primo concetto fondamentale che React introduce è quello di *Componente*, ovvero parte di interfaccia grafica indipendente e riutilizzabile in diversi punti. Il componente è responsabile di renderizzare l'interfaccia grafica e computare possibili dati. Per modellare dati in React esistono due tecniche fondamentali: *props* (proprietà) e *state* (stato). Le props sono descrivibili come input arbitrario che ogni componente può ricevere quando viene utilizzato, mentre lo state è leggermente più complicato: esso rappresenta appunto

lo stato in cui il componente si trova in un determinato istante. Ogni cambiamento di dati (sia *props* che *state*) genera un cosiddetto *"rerender"*, ovvero un aggiornamento del componente che ha come obiettivo quello di rispecchiare il cambiamento di stato nell'interfaccia grafica.
Ogni applicazione React può essere considerata come una piramide di componenti (*Component Tree*), in cui né componenti figli né componenti padri conoscono lo stato degli altri componenti, ma dove solo i componenti padri possono passare dati (attraverso le props) ai componenti figli. Per questo si dice che React ha un data flow unidirezionale, dall'alto verso il basso (*top-down*).
Un'altra funzionalità estremamente importante che React introduce è JSX [87], una sintassi specificatamente pensata per descrivere l'interfaccia grafica di un componente.

Figura 2.2: Esempio di sintassi JSX introdotta da React [88]

Essa ricorda vagamente un più classico linguaggio di markup come HTML [89], ma include tutta la potenza e i vantaggi di JavaScript. Questo introduce la possibilità di includere sia logica che aspetto di un componente in un unico punto, migliorando notevolmente la leggibilità del codice.
In conclusione, React ha innumerevoli vantaggi (ben più di quelli brevemente descritti) che hanno portato alla decisione di utilizzarlo come elemento portante dello sviluppo del Client Web.

Next.js [15]

React di per sé è una libreria per sviluppare interfacce grafiche, ma per utilizzare tecniche come *client-based routing* (utilizzare JavaScript per cambiare pagina all'interno del sito web), *bundling* [90] e altro è necessario rivolgersi ad altre librerie, rallentando significativamente lo sviluppo. Per risolvere questo problema si è deciso di utilizzarer Next.js, un React framework considerato tra i più maturi e completi. Esso ha tantissime funzioni e configurazioni che velocizzano enormemente la fase di sviluppo iniziale di un progetto React. La prima opzione fondamentale che il framework introduce è la possibilità di servire una pagina web tramite Server Side Rendering (abbr. *SSR*) [91] e Static Site Generation (abbr. *SSG*) [92], due tecniche molto diverse ma che portano innumerevoli vantaggi. SSR consiste brevemente nel computare (tecnicamente *build*) la pagina ad ogni richiesta direttamente sul server e ritornare HTML al client. SSG, invece, è un approccio basato su una singola computazione della pagina (*build*) e successivamente questa viene servita ad ogni richiesta. Entrambi sono superiori al più classico Client Side Rendering (abbr. *CSR*) [93] in quanto garantiscono velocità maggiori grazie al fatto che il client non deve scaricare ed eseguire tutto il codice JavaScript del sito, ma riceve direttamente del codice HTML e una parte minimale di codice JavaScript. Inoltre, questi due approcci hanno prestazioni migliori per quanto riguarda la Search Engine Optimization (SEO).
Altra funzionalità estremamente utile offerta da Next.js è quella di built-in routing basato su File System [94]. In breve, ogni file presente nella directory `pages` corrisponde ad una route, ovvero una pagina web visitabile ad un indirizzo.
Una delle funzionalità più importanti che il framework offre è quella delle cosiddette API Routes [95]. Queste route si trovano nella directory `pages/api` e sono concettualmente diverse dalle route relative alle pagine web introdotte precedentemente. Ogni route è infatti pensata come un API endpoint che riceve una richiesta HTTP e ritorna una risposta. Per esempio, se nella directory `pages/api` è presente un file chiamato `user.js` con il seguente

contenuto:

```
1 export default function handler(req, res) {
2   res.status(200).json({ name: req.query.name || "John Doe"
      }))
3 }
```

sarà possibile inviare una richiesta all'indirizzo `/api/user?name=Nextjs` e ricevere una risposta HTTP con codice di stato 200 contenente un JSON della forma {`"name": "Nextjs"`}.
Questa è una delle funzionalità per cui Next.js è risultato la scelta ideale in quanto, accoppiata alla piattaforma Vercel (descritta maggiormente nella sezione 3.1.2 legata all'architettura), è in grado di velocizzare enormemente lo sviluppo dell'API relativa a DARE-SAM.

NextAuth.js [96]

Dato che la sicurezza è uno dei requisiti fondamentali del progetto, è necessario introdurre una soluzione di autenticazione all'avanguardia che rispetti tutti gli standard odierni. È infatti sconsigliato implementare soluzioni ad-hoc quando si hanno capacità ridotte (come in questo caso) in quanto il rischio di introdurre vulnerabilità è molto alto. Per questo si è deciso di optare per la libreria open source *NextAuth.js*, pensata appositamente per il framework Next.js. Essa si integra perfettamente con le API Routes e permette, con pochissime linee di codice, di aggiungere autenticazione tramite provider (come Google, Facebook, Github, e tanti altri), passwordless (detto anche tramite *"magic link"* [97]) o tramite le più classiche credenziali (username e password). Un altro vantaggio è quello di poter collegare un database in modo che i dati relativi a utenti e sessioni vengano salvati su un supporto deciso dallo sviluppatore. Maggiori informazioni relative all'autenticazione presente in DARE-SAM sono presenti alla sezione 3.3.2.

2.3.3 Backend Service

Le tecnologie utilizzate per il Backend Service sono principalmente legate alla gestione dei dati e aggiornamento del database.
Di seguito sono elencate le principali.

Node.js [16]

Node.js è una delle tecnologie più rivoluzionarie che si sono affermate negli ultimi 10 anni per lo sviluppo di applicativi back-end. Esso ha come obiettivo quello di eseguire codice JavaScript fuori da browser, e questo è un grande vantaggio in quanto milioni di sviluppatori front-end possono apportare modifiche a codice back-end senza aver bisogno di imparare un nuovo linguaggio. Node.js è basato su V8 [98], JavaScript engine open source sviluppato da Google e utilizzato in Chrome, che lo rende molto performante. Tra le peculiarità spicca il fatto che un'applicazione Node.js è composta da un singolo thread, al contario di molti altri runtime dove ogni richiesta genera un nuovo thread. Questo ha dei vantaggi consistenti in quanto Node.js può gestire migliaia di richieste concorrenti con un singolo server senza doversi preoccupare di gestire la concorrenza dei thread e con ciò risparmiando numerosi bug.

PostgreSQL [99]

PostgreSQL, o più seplicemente *Postgres*, è un database relazionale open source che è da molti anni uno strumento estremamente popolare nella community di sviluppatori back-end. Esso, essendo un RDBMS, garantisce le proprietà ACID (*Atomicity, Consistency, Isolation, Durability*) e ha supporto completo per funzionalità come *join*, *foreign keys* e tante altre. Offre inoltre supporto per una grandissima quantità di tipi, tra cui array, oggetti in linguaggio JSON, tipi geometrici e tipi compositi. Un'altra caratteristica per cui PostgreSQL è in cima alle scelte per quanto riguarda i database è il fatto che un grandissimo numero di provider cloud lo supportano e perciò

non si è legati ad uno in particolare. Molte soluzioni di basi di dati che sono nate ultimamente, infatti, come Amazon DynamoDB [100] o Firebase Firestore [101], offrono numerose funzionalità interessanti ma non permettono il trasferimento su altri provider cloud o su soluzioni self-hosted.

TypeOrm [102]

Dopo aver citato PostgreSQL, è doveroso citare le modalità con cui si accede ai dati contenuti in esso. Un ORM (*"Object Relational Mapper"*) [103] è indubbiamente una parte fondamentale di ogni applicativo back-end per garantire un accesso ai dati semplificato e una maggiore sicurezza. In breve, l'obiettivo di un ORM è quello di mappare entità presenti nel database ad oggetti in un qualsiasi linguaggio di programmazione object-oriented. Nel panorama JavaScript uno degli ORM più utilizzati e apprezzati dalla community è TypeOrm. Questa libreria è estremamente versatile, dato che può essere eseguita da qualsiasi browser, Node.js, React Native e praticamente ogni piattaforma che supporti il runtime JavaScript. Inoltre, è pensato appositamente per TypeScript (è realizzato in TypeScript) e quindi garantisce un'estrema accuratezza nei tipi delle sue API.

Capitolo 3

Architettura e Implementazione

Nel terzo capitolo di questo elaborato si procede ad analizzare l'architettura del sistema e come essa è stata pensata per soddisfare gli obiettivi già introdotti nel capitolo precedente. Dopodiché è presente una fase dedicata all'implementazione della Sentiment Analysis che include le decisioni che hanno portato alla scelta di un particolare provider cloud; ed infine nell'ultima parte vengono discussi alcuni aspetti implementativi di particolare interesse, divisi per componenti principali.

3.1 Architettura

L'architettura del progetto DARE-SAM si compone di diversi componenti ben definiti e separati tra di loro. Questo permette, come già visto in precedenza nello scorso capitolo, di superare le varie sfide proposte da *Facebook Graph API* e allo stesso tempo risulta una soluzione più efficiente. Inoltre, nel caso si introduca la necessità di aumentare capacità in futuro, la seguente architettura permette di farlo solamente per i componenti che lo necessitano.

3.1.1 Componenti

Di seguito viene riportato uno schema esplicativo dell'architettura di DARE-SAM (3.1):

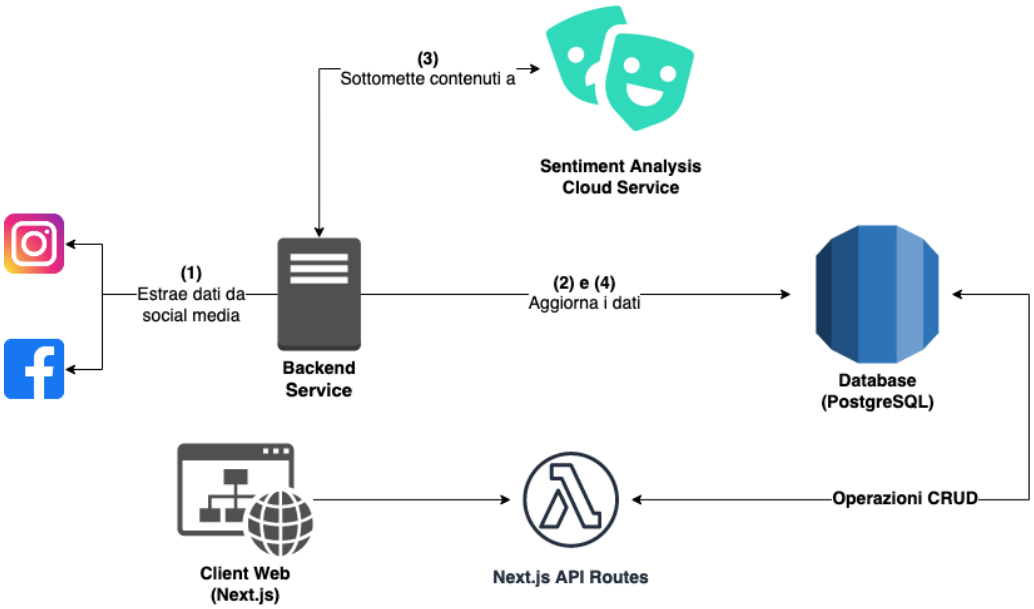

Figura 3.1: Schema concettuale dell'architettura di DARE-SAM

I componenti in dettaglio sono i seguenti:

- *Backend Service*: è il servizio responsabile di estrarre nuovi contenuti dai social network, sottomettere i testi al servizio di Sentiment Analysis per l'elaborazione e aggiornare i dati all'interno del database. I requisiti principali che esso deve soddisfare sono:
  - eseguire ad ogni intervallo di tempo precedentemente stabilito (idealmente ogni ora);
  - per ogni utente, estrarre i contenuti pubblicati sui suoi profili social registrati ed aggiornarli all'interno del database;

  - computare il sentimento dei nuovi testi non ancora analizzati (post, commenti) tramite l'utilizzo del *Sentiment Analysis Cloud Service*.

- *Next.js API Routes*: considerato come l'API che mette in comunicazione Client Web e database. I compiti principali sono:
  - gestire l'aggiornamento dei profili social da parte degli utenti;
  - gestire l'autenticazione;
  - rendere i dati contenuti nel database correttamente accessibili dalla piattaforma web.

- *database*: PostgreSQL, accedibile sia dal Backend Service che dalle Next.js API Routes;

- *Sentiment Analysis Cloud Service*: servizio responsabile di effettuare Sentiment Analysis sui testi raccolti dalle piattaforme social;

- *Client Web*: piattaforma web accedibile da qualsiasi browser, permette di visualizzare informazioni e gestire i profili social collegati all'account dell'utente.

3.1.2 Hosting

L'hosting dell'infrastruttura deve essere correttamente progettato per mantenere i vantaggi che l'architettura scelta permette di raggiungere. È stato scelto di includere due approcci differenti, uno con lo scopo di una maggior semplicità nell'aumentare capacità mentre l'altro è pensato per una maggior efficienza (costi ridotti). Prima di elencare le differenze tra essi è però necessario introdurne i punti in comune:

- *Sentiment Analysis Cloud Service*: l'accesso al servizio di Sentiment Analysis rimane costante per i diversi approcci. Esso avverrà infatti tramite provider cloud, quindi non è necessario gestirne la capacità e allo stesso modo non è possibile avera variazioni di prezzo;

- *Client Web + Next.js API Routes*: come già introdotto nel capitolo precedente, una delle funzionalità più interessanti di Next.js è la presenza delle API Routes, *endpoints* raggiungibili ad un particolare indirizzo che ricevono una richiesta HTTP e ritornano una risposta. Esse possono essere utilizzate tramite un più classico server (con il comando `'next start'` [104]), ma il modo suggerito dai creatori del framework è quello di utilizzare funzioni serverless [105] per ciascuna richiesta, in quanto ciò semplifica notevolmente l'architettura. Questa è una delle funzionalità più interessanti offerte dalla piattaforma *Vercel* [106], provider con cui si è deciso di effettuare hosting di Client Web e Next.js API Routes. Oltre alla già citata possibilità di lanciare API Routes in modalità serverless, Vercel offre una CDN (*Content Delivery Network* [107]) globale con cui è possibile servire le parti statiche del Client Web da una location il più vicina possibile all'utente finale, con l'obiettivo di ridurre il tempo necessario a caricare pagine web. Tutto ciò è ottenibile già dal piano Hobby [108] a $ 0 al mese, il che rende *Vercel* un'ottima scelta per il caso d'uso di DARE-SAM.

Ora si analizzaranno invece le differenze tra i due approcci introdotti in precedenza.

Primo approccio: una singola macchina

In questa architettura, Backend Service e database vengono posizionati su una stessa macchina per avere un costo più basso. Una piattaforma che permette di semplificare numerose azioni relative alla gestione di questa architettura è Dokku [109]. Essa è una PaaS (*"Platform as a Service"*, [110]) che, sfruttando Docker [111], permette di realizzare architetture componibili con più servizi su una singola macchina. Dokku raggruppa tutte le risorse necessarie al funzionamento di un servizio (database, message buses, ecc.) e rende più semplice il monitoraggio e il deployment di nuovi cambiamenti.
Di seguito è presente uno schema esplicativo di questo approccio (3.2):

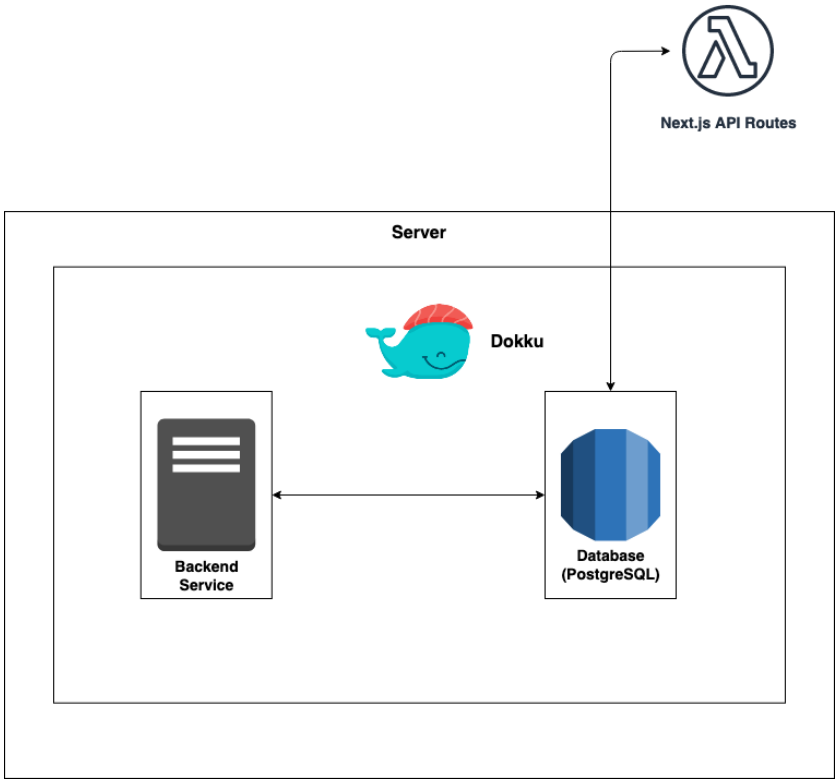

Figura 3.2: Schema concettuale dell'approccio architetturale ad una singola macchina

Il server potrebbe essere una qualsiasi macchina con Docker e Dokku installati, e alcune soluzioni cloud online sono decisamente economiche. Un esempio di provider cloud che soddisfa pienamente i requisiti di questa architettura è DigitalOcean [112], che offre VPS (*"Virtual Private Server"*) chiamate *"Droplets"* [113] a partire da $ 5 al mese [114]. Inoltre è possibile creare una *Droplet* con già Dokku e Docker installati, e ciò ha permesso di risparmiare tempo per inizializzare il server. Con questo approccio si ottengono i seguenti vantaggi:

- *centralizzazione*: essendo Backend Service e database sulla stessa macchina, gestire e deployare modifiche ad entrambi risulta semplificato; ciò permette inoltre di avere latenza nulla tra i due componenti;

- *costo*: l'unico costo da sostenere è quello dell'istanza che, come descritto in precedenza, può essere particolarmente ridotto;

- *modulabilità*: Dokku presenta numerosi plugin e, nel caso si verifichi la necessità di aggiungerne di altri (ulteriori database, Message buses, ecc.) all'occorrenza, ciò sarebbe estremamente semplice e non comporterebbe aumenti di costi.

Gli svantaggi principali invece sono:

- *singolo punto di fallimento*: un possibile problema all'istanza risulterebbe in un malfunzionamento di entrambi Backend Service e database;

- *difficile aumentare capacità selettivamente*: anche se, come abbiamo visto, Dokku permette di installare plugin per aggiungere risorse all'architettura, tutte le risorse sarebbero istanziate su una singola macchina; ciò renderebbe complesso aumentare la capacità di un singolo componente (database o Backend Service) lasciando gli altri invariati.

Secondo approccio: due macchine separate con Database as a Service

In questa seconda proposta, il Backend Service risiede su una macchina mentre per il database vengono utilizzati servizi all'avanguardia che permettono di avere numerosi vantaggi per quanto riguarda hosting di database: *Database as a Service* (abbr. *DBaaS*). Come prima cosa, essi non richiedono allo sviluppatore installazione e configurazione manuale di macchine in quanto ciò viene automaticamente eseguito dal provider, risparmiando tempo. Oltre a ciò, un *DBaaS* permette di aumentare capacità in maniera estremamente semplice, aumentando le risorse hardware dell'istanza (numero di CPU e memoria) oppure aggiungendo *"repliche di lettura"*, ovvero macchine addizionali il cui compito è quello di rispondere alle richieste al posto del database primario. Infine, numerosi provider cloud permettono di avere un backup automatico del database ad ogni momento in modo da avere la

certezza di non perdere dati neanche nel caso di malfunzionamenti all'istanza. Un esempio di *DBaaS* è Amazon RDS [115], servizio che include tutte le funzionalità già citate e molte altre, offrendo allo sviluppatore la massima configurabilità.

Nella seguente figura (3.3) è riportato un diagramma di questo secondo approccio:

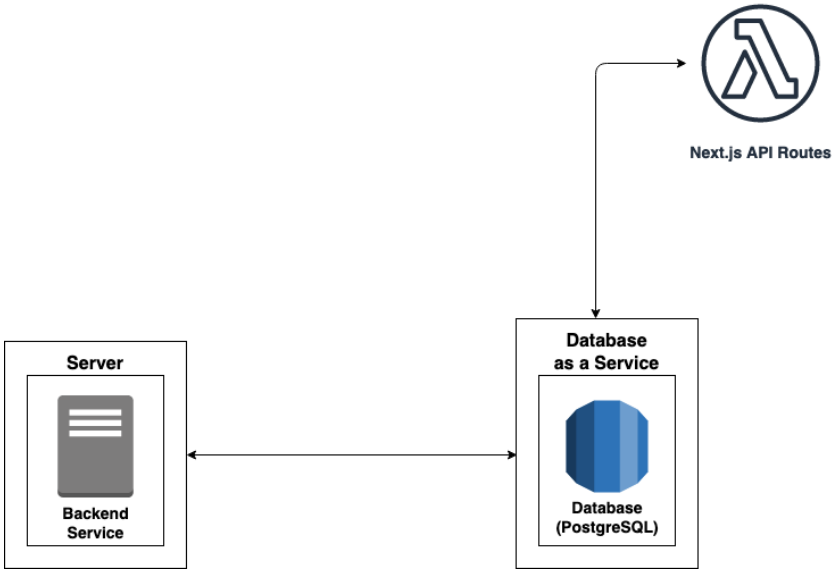

Figura 3.3: Schema concettuale dell'approccio architetturale con DaaS

I vantaggi principali che questo approccio introduce sono:

- *semplicità nell'aumentare capacità*: è veramente semplice scalare solamente uno dei due componenti (Backend Service e database) in quanto essi risiedono su due macchine separate; inoltre, come abbiamo visto, un *DBaaS* permette di avere numerose funzionalità aggiuntive estremamente utili per lavorare con i database;

- *affidabilità*: alcuni *DBaaS* prevedono funzionalità aggiuntive per replicare i dati in modo da aumentare l'affidabilità; oltre a ciò, in questo

caso un eventuale malfunzionamento ad uno dei due componenti non causerebbe downtime di tutto il servizio.

I punti negativi invece si riassumono in:

- *costo maggiore*: quest'architettura ha un costo potenziale più alto in quanto sono necessarie più risorse hardware.

Per lo sviluppo e la validazione della fase iniziale del progetto DARE-SAM si è optato per il primo approccio con una singola macchina in quanto i costi ridotti hanno una rilevanza maggiore rispetto all'affidabilità e alla maggiore scalabilità. Sono stati incluse entrambe le architetture in quanto eventuali sviluppi futuri del progetto potrebbero avere diversi requisiti per cui il secondo approccio risulterebbe più adeguato.

3.2 Sentiment Analysis

La Sentiment Analysis, come già descritto in precedenza, è un aspetto fondamentale del sistema che deve essere accuratamente architettato per garantire un'elevata accuratezza ed efficienza. Nel secondo capitolo (2.2.2) si sono discusse le motivazioni che hanno portato alla scelta di utilizzare un servizio offerto da un provider cloud, mentre ora verrà affrontata una comparazione dei vari servizi già introdotti per scegliere quello migliore per DARE-SAM.

SentiRace

Per questo scopo è stato realizzato un software specifico chiamato *SentiRace* (open source, visitabile all'indirizzo `https://github.com/Gobbees/SentiRace`). Esso è in grado di ricevere una lista di frasi in input (da elencare nel file `input.json`) assieme alla lingua (in alcuni dei servizi è obbligatorio specificare la lingua), computare il sentimento per ognuna delle frasi tramite tutti e quattro i servizi visti in precedenza (*Google Cloud Natural Language*, *Azure Text Analytics*, *IBM Watson Natural Language Understanding* e

Amazon Comprehend) e creare un report consultabile dall'utente.
Per scegliere il più accurato relativo al caso d'uso di DARE-SAM sono stati accuratamente scelti testi specifici relativi a vari argomenti di interesse estratti dalla pagina Facebook del sindaco di Ravenna Michele de Pascale [67]. In essi sono presenti vari temi caldi nell'ultimo periodo per la realtà Ravennate, tra cui Covid 19, politica e Darsena. Il risultato realizzato da *SentiRace* è quindi il seguente:

Sentence	GCP Sentiment (-1 negative, 1 positive)	Azure Sentiment	AWS Sentiment	IBM Sentiment
Oltre purtroppo ai decessi , sarebbe interessante sapere come procedono le vacinazioni anche per i disabili c'è poca comunicazione grazie	-0.699999988079071	negative	NEGATIVE	neutral
È stato scelto dal Presidente Mattarella, poteva scegliere tra tanti, forse ha fatto bene?? Almeno diamogli fiducia dato che Conte e company non erano in grado di gestire i soldi che arriveranno. Poi ci ricordiamo cosa ha fatto Arcuri, il loro scelto?	-0.30000001192092896	mixed	NEUTRAL	negative
Conte coraggioso fino all'ultimo, con Draghi sarà baracca	0.10000000149011612	neutral	NEUTRAL	neutral
Troppo giovani per morire dl covid, condoglianze a tutti i parenti dei defunti.🌹. Buona notte Sindaco e staff.	0	mixed	POSITIVE	positive
Signor sindaco mi dia una risposta se è possibile perché i circoli devono stare chiusi siamo anche noi partite IVA paghiamo tasse utenze e altro .perché siamo così penalizzati abbiamo diritto di lavorare. Grazie	0	mixed	NEUTRAL	negative
Mi chiedo se il coprifuoco diventerà parte integrante del nuovo stile di vita, all'insegna della povertà e della perdita dei diritti.	-0.10000000149011612	negative	NEUTRAL	negative
Di questo passo non si potrà neanche andare al mare, le persone sono esauste non se ne può più di questo tira molla senza avere informazioni concrete e chiare	-0.6000000238418579	negative	NEGATIVE	negative
Complimenti Sindaco, per tutta la Sua attività! Orgogliosa di essere residente a Ravenna	0.5	positive	POSITIVE	positive
piace a molti e piacerà sempre di più. è una bella zona per stare insieme, per chi ha piacere di starci.	0.8999999761581421	positive	POSITIVE	positive
La passerella è bellissima. Però bisognerebbe inviare più spesso gli addetti ai rifiuti. Inoltre i cestini rifiuti sono troppo piccoli e troppo pochi	-0.10000000149011612	mixed	MIXED	negative
La Darsena a Ravenna ha un grande potenziale e il comune fa bene a fare investimenti perché in futuro ci sono grandi opportunità e progetti.	0.800000011920929	positive	POSITIVE	positive
Ottimo ! La valorizzazione della Darsena è il vero potenziale futuro per Ravenna!	0.8999999761581421	positive	POSITIVE	positive
Ma su quali basi viene stanziato il denaro? Senza togliere nulla alla passerella, però le priorità di Ravenna sarebbero altre....	-0.10000000149011612	neutral	NEUTRAL	neutral

Figura 3.4: Risultato della computazione del sentimento effettuata da SentiRace

Dopo un'accurata analisi dei risultati e comparazione tra sentimento computato dai servizi e sentimento riscontrato da esseri umani, si è deciso che *Azure Text Analytics* è il più accurato, seguito da *Google Cloud Natural Language* (esso ha un formato del risultato diverso dagli altri in quanto il sentimento è classificato su una scala da -1 a 1).
Un secondo risultato interessante offerto da *SentiRace* è visibile nel file `result.json` generato a seguito dell'esecuzione, che contiene, per ogni ser-

vizio, la risposta completa ritornato dalla richiesta. Se si analizza il risultato di una frase computato da *Azure Text Analytics*:


```
{
  "id": "0",
  "warnings": [],
  "sentiment": "mixed",
  "confidenceScores": {
    "positive": 0.62,
    "neutral": 0.05,
    "negative": 0.33
  },
  "sentences": [
    {
      "text": "Troppo giovani per morire dl covid,
         condoglianze a tutti i parenti dei defunti.",
      "sentiment": "negative",
      "confidenceScores": {
        "positive": 0.27,
        "neutral": 0.08,
        "negative": 0.65
      },
      "offset": 0,
      "length": 81
    },
    {
      "text": "Buona notte Sindaco e staff.",
      "sentiment": "positive",
      "confidenceScores": {
        "positive": 0.97,
        "neutral": 0.03,
        "negative": 0
      },
      "offset": 82,
      "length": 28
    }
  ]
}
```

si può notare come sia presente una divisione in frasi con computazione del sentimento relativo a ciascuna di esse (nel testo di esempio la prima parte ha sentimento negativo mentre la seconda ha sentimento positivo). Questo consiste in una funzionalità interessantissima che potrebbe essere sfruttata in uno sviluppo futuro del software. Una funzionalità simile è offerta anche dal servizio *Google Cloud Natural Language*.

Grazie alle precedenti considerazioni in materia di accuratezza ed efficienza (affrontata nel secondo capiolo) e all'introduzione di questa interessante funzionalità, si è quindi deciso di proseguire con l'utilizzo del servizio *Azure Text Analytics* per la prima versione di DARE-SAM.

3.3 Implementazione

Questa sezione ha come obiettivo quello di far risaltare alcuni aspetti implementativi principali realizzati durante lo sviluppo del progetto DARE-SAM, divisi per componente architetturale, introducendo inoltre una breve sezione che illustra la struttura del repository (visitabile all'indirizzo `https://github.com/Gobbees/dare-sam`).

3.3.1 Struttura del Repository

Il repository contiene tutti i componenti architetturali di DARE-SAM, ovvero Client Web, Next.js API Routes e Backend Service. Ciò è reso possibile da una tecnologia che è risultata estremamente utile durante l'intero sviluppo del progetto: Yarn [116]. Essa è fondamentalmente un package manager per applicazioni Node.js, simile al ben più noto NPM [117], ma ha una funzione fondamentale che ha semplificato enormemente la struttura del progetto: Yarn Workspaces [118]. Questa funzionalità permette di creare *Monorepository* [119], ovvero repository in cui vivono diversi pacchetti (*packages*) che dipendono tra di loro. Il fatto di poter creare un package A che ha come dipendenza un altro package B semplifica la gestione e

l'aggiornamento di entrambi, dato che, se ciò non fosse possibile, bisognerebbe caricare il package B su un repository raggiungibile da Yarn o NPM (come `https://www.npmjs.com`) e successivamente aggiornare le dipendenze del package A. Oltre a ciò, Yarn Workspaces semplifica l'installazione delle dipendenze dei vari packages in quanto, tramite il comando `yarn install`, esso installa tutte le dipendenze definite nei vari file `package.json` di ciascun package.
Introdotto il concetto di Monorepository, si procede quindi ad elencare e giustificare le dipendenze tra i vari packages presenti all'interno del repository. Nella directory `packages` sono presenti 4 packages: `web`, `service`, `database` e `common`. Il primo contiene il codice sorgente relativo al Client Web e Next.js API Routes. Esso dipende sia da `common` che da `database`, perciò entrambi sono presenti nelle sue dipendenze. Il package `service` contiene il codice sorgente del Backend Service, l'applicativo che si occupa di estrarre e analizzare i contenuti provenienti dai social media. Anche esso dipende da `common` e `database`, come `web`. Passando a `common` e `database`, essi sono packages che contengono componenti utilizzate dagli altri packages già citati e che sono stati estratti per evitare ripetizioni. Il primo contiene funzioni di aiuto relative a *Facebook Graph API* e tipi in comune, mentre il secondo contiene le entità che modellano i dati contenuti all'interno del database (si rimanda alla sezione 3.3.3 per maggiori informazioni relativi all'implementazione).

3.3.2 Client Web e Next.js API Routes

Durante la fase di implementazione del Client Web e delle Next.js API Routes sono state incontrate numerose sfide impegnative che hanno richiesto un'ampia ricerca e un'attenta implementazione per essere superate. Di seguito si elencano alcuni aspetti implementativi di questi due componenti architetturali che risultano particolarmente interessanti nel contesto del progetto DARE-SAM.

Autenticazione e sicurezza

L'autenticazione è un aspetto fondamentale del software in quanto deve essere garantita la riservatezza e la sicurezza degli utenti. Come già introdotto nel secondo capitolo, la soluzione di autenticazione scelta si basa su NextAuth.js [96], libreria open source che si sposa perfettamente con il framework Next.js. NextAuth.js, tra le varie funzionalità offerte, permette di effettuare la registrazione di utenti al sistema e gestirne l'autenticazione tramite sessione. Queste sono le due principali funzionalità che sono state implementate in DARE-SAM.
Il funzionamento della piattaforma per quanto riguarda la registrazione e autenticazione di un utente, è ampiamente descritto alla sezione 4.1. In questa sezione vengono invece descritti e discussi alcuni frammenti di codice di elevata importanza nell'ambito dell'autenticazione.
Per la registrazione alla piattaforma, è presente una schermata apposta all'indirizzo `/login` (illustrata alla figura 4.1) che richiede l'email dell'utente. Essa contiene un form realizzato tramite *Formik* [120], una libreria per semplificarne la realizzazione. Il form di login presenta la seguente funzione che viene chiamata una volta che il bottone *Login with email* viene premuto con quando è presente una email valida nella casella di testo:

```
1 // login.tsx
2   ...
3   onSubmit={(values) =>
4     signin('email', {
5         email: values.email,
6         callbackUrl: `${process.env.NEXT_PUBLIC_BASE_URL}/
              account`,
7     })
8   }
```

La funzione `signin` è importata da NextAuth.js e invia una richiesta di signin all'endpoint `/api/auth/signin`, che viene gestita dalla funzione di default

contenuta nel file `packages/web/src/pages/api/auth/[...nextauth].ts`:

```
1  // [...nextauth].ts
2  export default async (req: NextApiRequest, res:
       NextApiResponse) =>
3    NextAuth(req, res, {
4      providers: [
5        Providers.Email({
6          server: {
7            host: process.env.SMTP_HOST || '',
8            port: Number(process.env.SMTP_PORT),
9            auth: {
10             user: process.env.SMTP_USER || '',
11             pass: process.env.SMTP_PASSWORD || '',
12           },
13         },
14         from: process.env.SMTP_FROM || '',
15       }),
16     ],
17     database: {
18       type: 'postgres',
19       host: process.env.POSTGRES_HOST,
20       port: Number(process.env.POSTGRES_PORT),
21       username: process.env.POSTGRES_USERNAME,
22       password: process.env.POSTGRES_PASSWORD,
23       database: process.env.POSTGRES_DATABASE,
24     },
25   });
```

La funzione `NextAuth` alla terza riga si occupa di gestire tutto il processo di registrazione, verifica dell'email e mantenimento delle sessioni. Essa si collega al database specificato nell'oggetto `"database"` e gestisce l'autenticazione tramite un singolo provider, ovvero `Email`. L'autenticazione è quindi di tipo passwordless (o tramite *"magic link"* [97]) e utilizza il server specificato dalle variabili d'ambiente per inviare la mail contenente il token di accesso. Come server email si è utilizzato SendGrid [121], un servizio che tra le tantissime

funzionalità offre la possibilità di avere un server SMTP per inviare email. Una volta cliccato il bottone *Sign In* contenuto nella mail (illustrato nell'immagine 4.1), viene registrato un nuovo utente con l'email specificata e si può procedere all'utilizzo della piattaforma. Quello che NextAuth.js fa è creare un utente e una sessione, inviando successivamente al client i cookies che permettono di accedere più volte senza aver bisogno di rieffettuare il login. Questa soluzione permette di avre un'ottima esperienza utente in quanto egli non deve memorizzare alcuna password ma può accedere comodamente da ogni device inserendo la propria email e cliccando il bottone nella email ricevuta.

Oltre alla iniziale autenticazione, il sistema deve anche assicurare un corretto isolamento dei dati di ogni utente, in quanto poter accedere a dati di altri utenti sarebbe considerata una grave falla di sicurezza. Per garantire ciò entra in gioco ancora una volta NextAuth.js che, unito alle Next.js API Routes, permette di creare cosiddette *"API Routes sicure"*. Come esempio si analizzi il seguente estratto di codice (opportunamente semplificato per una maggiore chiarezza):

```
1  // comments.ts
2  export default async (req: NextApiRequest, res:
       NextApiResponse) {
3    const session = await getSession({ req });
4    if (!session) {
5      return res.status(401).end();
6    }
7    const postId = req.query.postid as string;
8    const { userId } = await NextSession.findOneOrFail({
9      where: { accessToken: session.accessToken },
10     select: ['userId'],
11   });
12   const { postUserId } = await Post.createQueryBuilder('post'
         )
13     .innerJoin(SocialProfile, 'sp', 'post.parentProfile = sp.
```

```
        id')
14      .innerJoin(User, 'user', 'sp.owner = user.id')
15      .where('post.id = :id', { id: postId })
16      .select('user.id', 'postUserId')
17      .getRawOne();
18    if (postUserId !== userId) {
19      return res.status(403).json({error: 'Missing user access'
            });
20    }
21    ...
22  }
```

La prima linea della funzione (linea 3) si occupa di ottenere la sessione dell'utente che ha effettuato la chiamata utilizzando `getSession`, funzione resa disponibile da NextAuth.js. Successivamente, partendo dal token della sessione, si verifica che l'utente abbia accesso alla risorsa specificata dal parametro `postid` contenuto nell'URL. Se questo non è rispettato, la funzione ritorna una risposta HTTP con codice 403 (*Forbidden*) e stato di errore, altrimenti si procede con l'esecuzione della richiesta.

Ottenere Facebook Tokens

Una sfida interessante è stata quella di realizzare una soluzione che garantisca una semplice modalità di ottenere Facebook Tokens che possano essere utilizzati per accedere ai profili social. Dopo una iniziale analisi, la soluzione migliore era sembrata quella di utilizzare direttamente la libreria *Facebook SDK* per Javascript [122] realizzata da Facebook. Questa non è però particolarmentte ottimizzata per soluzioni che utilizzano React e ciò ha portato a valutare altre opzioni. La più convincente grazie alla sua semplicità di utilizzo ed estensiva documentazione è *Firebase Authentication* [123], soluzione realizzata da Google come parte del più ampio servizio cloud Firebase [124]. Essa permette di eseguire l'accesso con un ampio numero di provider di autenticazione e quindi offre ampie possibilità per estendere il progetto con il supporto ad ulteriori social media come Twitter e Youtube. Il seguente

frammento di codice invia una richiesta a Firebase Authentication richiedendo l'accesso a Facebook con i permessi necessari al caso d'uso di DARE-SAM:

```
1  // facebook-tokens.ts
2  const facebookProvider = new firebase.auth.
       FacebookAuthProvider();
3
4  // pages related permissions
5  facebookProvider.addScope('pages_show_list');
6  facebookProvider.addScope('pages_read_engagement');
7  facebookProvider.addScope('pages_read_user_content');
8  if (instagramAccess) {
9    facebookProvider.addScope('instagram_basic');
10 }
11 const result = await firebase.auth().signInWithPopup(
       facebookProvider);
12 return (result.credential as firebase.auth.OAuthCredential).
       accessToken;
```

Come è possibile notare, è estremamente semplice aggiungere permessi addizionali a differenza di altre soluzioni analizzate. Il risultato restituito all'ultima linea di codice è il token di accesso che, quando inserito nelle richieste a Facebook Graph API, permette di accedere ai profili social dell'utente.

Dashboard

La dashboard principale (visitabile a `/dashboard`) è la schermata principale della piattaforma. Essa contiene tutte le informazioni relative ai profili social collegati dall'utente, navigabili tramite una tabella interattiva. Per maggiori informazioni in merito alle varie informazioni e al funzionamento di essa si invita a consultare la sezione 4.3. In seguito verranno invece elencati alcuni aspetti implementativi di particolare interesse.

Come prima cosa è necessario verificare che l'utente sia correttamente autenticato prima di accedere a qualsiasi informazione. Per tener conto di ciò

è stato implementato un React Custom Hook [125] chiamato `useUser`, ovvero un particolare componente React che ha lo scopo di verificare l'esistenza della sessione NextAuth.js e successivamente richiedere i dati dell'utente alla corretta API Route:

```
1  // UseUser.ts
2  const useUser = (): UseUserResult => {
3    const [session, isSessionLoading] = useSession();
4    const { user, setUser } = React.useContext(UserContext);
5    const [state, setState] = React.useState<UseUserState>({
6      isLoading: isSessionLoading || !user,
7    });
8    const { data, status, error } = useQuery(
9      'user',
10     async () => {
11       const response = await fetch('/api/user');
12       return response.json();
13     },
14     {
15       enabled: !!session && !isSessionLoading,
16     },
17   );
18   React.useEffect(() => {
19     if (data && status !== 'error') {
20       ReactDOM.unstable_batchedUpdates(() => {
21         // to avoid two rerenders
22         setUser(data as User);
23         setState({ errorState: undefined, isLoading: false })
               ;
24       });
25     } else if (!data && status === 'error') {
26       setState({ isLoading: false, errorState: error as Error
             });
27     } else if (!session && !isSessionLoading) {
28       setState({
29         isLoading: false,
30         errorState: new Error('Missing Authentication'),
```

```
31        });
32      }
33    }, [session, isSessionLoading, data, status, error, setUser
        , setState]);
34
35    return {
36      user,
37      session,
38      loading: state.isLoading,
39      setUser,
40      errorState: state.errorState,
41    };
42  };
```

`useSession` alla linea 2 è responsabile di ottenere la sessione. Una volta che la sessione è presente, si procede a richiedere i dati dell'utente attraverso la funzione `useQuery` alla linea 7. Essa è un'API della libreria React Query estremamente utile per gestire chiamate ad API. Si rimanda alla documentazione ufficiale [126] per maggiori informazioni. Successivamente, una volta che la funzione `useQuery` termina, si aggiorna correttamente lo stato dell'applicazione all'interno del costrutto `if-else-if` alla riga 18. Tornando alla pagina di Dashboard, consultabile al file `packages/web/src/pages/dashboard/index.tsx`, possiamo evidenziare l'utilizzo del custom Hook `useUser`:

```
1  // index.tsx
2  const DashboardPage = () => {
3    const router = useRouter();
4    const { user, loading } = useUser();
5
6    React.useEffect(() => {
7      if (!user && !loading) {
8        router.replace('/login');
9      }
10   }, [user, loading, router]);
11   ...
```

Il frammento di codice permette, nel caso l'utente non sia correttamente autenticato di reindirizzarlo alla pagina di login, migliorando notevolmente l'esperienza di utilizzo.

Il prossimo dettaglio implementativo descrive la creazione della tabella. Essa è contenuta nel componente `DashboardTable` (in `packages/web/src/components/dashboard/DashboardTable.tsx`) e al suo interno contiene diversi componenti figli che permettono di gestirne varie funzionalità. Innanzitutto, guardando lo stato del componente:

```
1 // DashboardTable.tsx
2 interface DashboardTableState {
3   fromDate: Date;
4   sinceDate: Date;
5   sourcesStatus: {
6     facebookSelected: boolean;
7     instagramSelected: boolean;
8   };
9 }
```

possiamo notare che esso mantiene i parametri con cui vengono applicati dei filtri alla tabella: `fromDate` e `sinceDate` esprimono la data iniziale e finale dei post da visualizzare, mentre `sourcesStatus` mantiene lo stato del filtro per profilo social, in quanto (come visibile alla sezione 4.3) è possibile selezionare solo alcuni dei profili social collegati all'account in modo da consultare i dati in maniera più efficace. Lo stato è quindi passato ai componenti figli di `DashboardTable`, nello specifico `IntervalSelector` e `SocialSelector`, i due componenti responsabili di gestire e apportare modifiche ai filtri. I post, nel caso siano presenti, sono invece passati a `PostTable`, il componente principale contenente la tabella in figura 4.13. Essa si serve di una libreria per semplificare numerose funzioni relative all'ordinamento, selezione e aggiornamento: React Table [127]. Grazie ad essa numerose delle funzionalità della

pagina sono risultate estremamente semplici da implementare. L'istruzione che gestisce l'inizializzazione della tabella è la seguente:

```
// PostTable.tsx
const {
  getTableProps,
  headerGroups,
  getTableBodyProps,
  rows,
  prepareRow,
  setFilter,
  toggleAllRowsExpanded,
  visibleColumns,
}: any = useTable(
  // any because toggleAllRowsExpanded doesn't exist in the
     basic TableInstance.
  {
    columns,
    data: tableData,
  },
  useFilters,
  useExpanded,
  );
```

La possibilità di applicare un filtro per profilo social è estremamente semplice: basta aggiungere `useFilters` alla funzione `useTable` e associare un filtro alla colonna desiderata. Il filtro in questo caso si occupa di verificare che la `source` di un post sia selezionata:

```
// PostTable.tsx
const sourceFilter = (
  rows: Array<Row<Post>>,
  id: Array<string>,
  filterValue: SourceFilter,
) =>
  rows.filter(
    (row) =>
```

```
9       (row.original.source === Source.Facebook &&
10        filterValue.facebookSelected) ||
11      (row.original.source === Source.Instagram &&
12        filterValue.instagramSelected),
13  );
```

Similmente, la possibilità di espandere un post e renderizzare la tabella dei commenti ricevuti su di esso (graficamente rappresentato tramite la figura 4.16) è realizzabile specificando l'attributo `useExpanded` alla funzione `useTable` e aggiungendo:

```
1  // PostTable.tsx
2  {row.isExpanded && (
3    <Tr>
4      <Td colSpan={visibleColumns.length}>
5        <Flex flexDir="column" align="center">
6          <Text fontWeight="extrabold">Comments</Text>
7          <Flex flexDir="column" align="center">
8            <CommentTable postId={tableData[row.index].id} />
9          </Flex>
10       </Flex>
11     </Td>
12   </Tr>
13 )}
```

alla parte di codice che si occupa di renderizzare ogni riga.

Dopo aver introdotto il componente `CommentTable` nel frammento di codice precedente è doveroso spiegarne la funzione. Esso si occupa di visualizzare una nuova tabella contenente i commenti per un post determinato dalla *props* `postId`. Esso, una volta renderizzato, si occupa di richiedere i commenti alla corrispondente Next.js API Route e visualizzarli. Anche in questo caso si è utilizzata la libreria React Table per implementare l'ordinamento delle righe. Infatti, la funzione `useTable` ha specificato l'attributo `useSortBy` che permette l'ordinamento di righe della tabella utilizzando diversi criteri. Sarà quindi possibile effettuare ordinamento crescente e decrescente per sentimen-

to, data di pubblicazione e altri meglio descritti nella sezione apposita 4.3 di funzionamento.

3.3.3 Backend Service

Come già introdotto nella sezione relativa all'architettura di DARE-SAM (3.1), Backend Service è il componente che si occupa di estrarre i contenuti dai profili social degli utenti, computare i sentimenti attraverso l'utilizzo del *Sentiment Analysis Cloud Service* e aggiornare tutto ciò all'interno del database. La scelta è ricaduta sullo sviluppo di un applicativo Node.js in quanto questa tecnologia introduce numerosi vantaggi di integrazione con gli altri componenti architetturali (Client Web e Next.js API Routes). I vantaggi sono legati principalmente al riutilizzo di parti di codice per l'accesso al database e condivisione di tipi tra i vari componenti. Node.js è inoltre estremamente diffuso e presenta un elevato numero di librerie con cui è possibile semplificare l'implementazione di alcune funzionalità.
Di seguito sono riportati alcuni particolari implementativi del Backend Service che sono risultati interessanti e impegnativi che abbiamo ritenuto interessanti da inserire all'interno dell'elaborato.

Esempio di Social Fetcher

La prima parte legata all'implementazione del Backend Service è senza dubbio legata alla sperimentazione e alla ricerca delle varie informazioni che è possibile estrarre da Facebook Graph API. Questa fase si è svolta in parte attraverso ripetute prove da codice effettuando richieste HTTP e in parte attraverso l'applicativo web *Graph API Explorer* [128] realizzato da Facebook per semplificare l'interazione con la propria API. Successivamente si è iniziata la fase di implementazione del Backend Service in modo che esso possa richiedere le informazioni richieste direttamente a Graph API. Questo particolare componente del Backend Service è chiamato *Social Fetcher*. Ciascuna piattaforma social presenta un *Fetcher* e la seguente parte di codice ne analizza il funzionamento:

```
/**
* Fetches the Facebook Posts for the input page.
*
* **Includes Paging.**
* @param options
* - `pageId`: the page id
* - `token`: the auth token
* - `fromDate`: fetches posts from a certain date
* - `withComments`: should fetch also the posts comments
* - `withCommentsReplies`: should fetch also the posts comments replies
*/
export const fetchFacebookPagePosts = async (options: {
  pageId: string;
  token: string;
  fromDate: Date;
  withComments?: boolean;
  withCommentsReplies?: boolean;
}): Promise<FacebookPost[] | undefined> => {
  let url = `${
    options.pageId
  }/posts?limit=${MAX_LIMIT}&since=${options.fromDate.toISOString()}&fields=id,created_time,message,picture,permalink_url,likes.summary(true),shares,comments`;
  if (options.withComments) {
    url = url.concat(
      `.order(reverse_chronological){id,message,created_time,likes.summary(true)${
        options.withCommentsReplies
          ? ',comments.order(reverse_chronological){id,message,created_time,likes.summary(true)}'
          : ''
      }}`,
    );
  }
  let response = await sendTokenizedRequest(url, true, options.token);
```

```
32    const posts: FacebookPost[] = [];
33    ... // adds from response to posts
34
35    return posts;
36  };
```

Procedendo all'analisi, la funzione è responsabile dell'estrazione dei post partendo da una pagina Facebook di input specificata dal parametro `pageId`. Altri parametri obbligatori per la funzione sono `token`, in quanto esso è richiesto da Graph API per ottenere l'identità dell'utente che sta effettuando la richiesta e `fromDate`, parametro che indica la data minima che i post devono soddisfare per essere ritornati; questo è necessario per evitare di ritornare ogni volta tutti i post pubblicati dalla pagina. La prima parte della funzione si occupa di computare l'URL di Graph API a cui inviare la richiesta. È possibile vedere che l'endpoint inizia con il `pageId` e richiede `posts` con una serie di parametri in richiesta (specificati dal campo `fields`). Successivamente, se il chiamante della funziona ha specificato anche la richiesta di commenti e risposte a commenti, si procede a modificare l'URL per prevedere correttamente questo caso. Una volta inviata la richiesta tramite la funzione `sendTokenizedRequest` (che si occupa di inviare la richiesta a Graph API e gestire alcuni errori in ritorno), si procede con la creazione dell'array di post che verrà ritornato dalla funzione. Una caratteristica importante che non verrà discussa ampiamente ma solo accennata è il supporto alla cosiddetta *paginazione* di Facebook Graph API [129], una tecnica che ha come obiettivo quello di limitare il numero massimo di dati in una singola risposta. Al posto di un'unica risposta contenente tutti i dati, Graph API ritorna una *pagina*, ovvero una parte di risposta contenente parte di dati e l'indirizzo alla prossima pagina. Ciò richiede l'implementazione di un meccanismo in grado di leggere tutte le pagine e sul Backend Service questo è presente per tutte le funzioni che inviano richieste a Graph API, sia per Facebook che per Instagram.

Gestione del database

Il passaggio seguente all'estrazione dei dati da social network è naturalmente quello di implementare una soluzione per salvarli in modo permanente. Qui entra in gioco la creazione del database PostgreSQL, soluzione scelta in quanto si configura ottimamente con tutte le tecnologie di DARE-SAM che richiedono memorizzazione di dati (NextAuth.js, Next.js API Routes e Node.js). Per l'accesso al database da parte del Backend Service si è introdotto *TypeOrm*, un ORM [103] molto utilizzato nell'eccosistema JavaScript/TypeScript. Il seguente è un esempio di entità TypeOrm, che corrisponde ad una tabella all'interno del database:

```
1  // SocialProfile.ts
2  @Index('social_profile_id', ['id'], { unique: true })
3  @Unique(['source', 'owner'])
4  @Entity('social_profiles')
5  export default class SocialProfile extends
       BaseEntityWithMetadata {
6    @PrimaryGeneratedColumn('uuid')
7    id!: string;
8
9    @Column({
10     name: 'source',
11     type: 'enum',
12     enum: Source,
13     nullable: false,
14   })
15   source!: Source;
16
17   @Column({
18     name: 'external_id',
19     nullable: false,
20   })
21   externalId!: string;
22
23   @Column({
```

```
24      name: 'name',
25      nullable: false,
26    })
27    name!: string;
28    ...
29    /**
30     * Profile owner reference
31     */
32    @ManyToOne(() => User, { nullable: false })
33    owner!: User;
34
35    /**
36     * Profile posts
37     */
38    @OneToMany(() => Post, (post) => post.parentProfile)
39    posts!: Post[];
40  }
```

La classe estende `BaseEntityWithMetadata`, una specifica classe realizzata estendendo l'oggetto `BaseEntity` di TypeOrm. Ciò indica all'ORM che questa classe dovrà rappresentare una tabella all'interno del database. Successivamente è presente una lista di campi, ciascuno con una specifica annotazione, il cui compito è quello di assegnare al campo numerose informazioni. Per esempio, `@PrimaryGeneratedColumn('uuid')` indica che il campo `id` è la chiave primaria dell'entità di tipo `uuid` [130], mentre `@Column` indica che il campo associato è una colonna della tabella e può specificare numerosi parametri come nome, tipo, eccetera.

Tutte le entità sono contenute in una directory che viene specificata a TypeOrm durante la creazione della connessione al database. Per il caso d'uso di DARE-SAM, dato che queste entità devono essere condivise con le Next.js API Routes, l'altro componente architetturale che ha accesso al database con lo scopo di leggere i dati e mostrarli all'utente, esse sono state estratte in un package specifico chiamato `database` che è elencato nelle dipendenze di Backend Service (package `service`) e Client Web (package `web`).

Una volta create le entità, il prossimo passo consiste nel leggere e scrivere

dati all'interno del database. Per fare ciò si utilizzano direttamente i metodi forniti da ogni classe che estende `BaseEntity`, ovvero le entità. Ad esempio, per leggere i dati degli utenti presenti all'interno del database associando ad ognuno di essi i propri profili social, basta eseguire:

```
1  // User.ts
2  const users = await User.find();
3  const returnObject = [];
4  for (const user of users) {
5    const facebookPage = user.facebookPageId
6      ? await SocialProfile.findOne(user.facebookPageId)
7      : undefined;
8    const instagramProfile = user.instagramProfileId
9      ? await SocialProfile.findOne(user.instagramProfileId)
10     : undefined;
11   returnObject.push({
12     user,
13     facebookPage,
14     instagramProfile,
15   });
16 }
```

Per quanto riguarda la scrittura, invece, si rimanda alla prossima sezione (3.3.3) dato che parte integrante di essa riguarda proprio l'aggiornamento dei dati.

Aggiornamento dei contenuti e computazione del sentimento

In quest'ultima parte relativa all'implementazione del Backend Service viene discusso brevemente l'intero aggiornamento di un profilo social. Le parti fondamentali sono:

- richiesta dei contenuti ai vari *Social Fetcher*, in base ai profili che un utente ha collegato (l'implementazione di esso è già stata affrontata nella sezione *Esempio di Social Fetcher* 3.3.3);

- aggiornamento dei contenuti all'interno del database;
- sottomissione dei nuovi testi non ancora analizzati al *Sentiment Analysis Cloud Service*;
- aggiornamento dei nuovi sentimenti all'interno del database;
- aggiornamento del sentimento aggregato per ogni post pubblicato.

Le prime due parti sono gestite dal cosiddetto *Social Worker*, un componente specifico per ogni piattaforma social supportata. Dopo aver richiesto i dati al Social Fetcher, esso prosegue a salvarli nel database:

```
1  // FacebookWorker.ts
2  ...
3  // updates posts
4  for (const post of posts) {
5    let postInDB = await Post.findOneBySource(Source.Facebook,
       post.id);
6    let postUpdateProperties: Partial<Post> = {};
7    if (postInDB) {
8      const shouldRecomputeSentiment = postInDB.message !==
         post.message;
9      postUpdateProperties = {
10       ...
11       sentiment: shouldRecomputeSentiment ? undefined :
           postInDB.sentiment,
12     };
13   } else {
14     postUpdateProperties = {
15       externalId: post.id,
16       source: Source.Facebook,
17       publishedDate: post.publishedDate,
18       permalink: post.permalink,
19       message: post.message,
20       picture: post.picture,
21       commentCount: post.commentCount,
```

```
22        shareCount: post.sharesCount,
23        likeCount: post.likeCount,
24        parentProfile: page,
25      };
26    }
27    postInDB = await Post.save({ // this updates data
28      ...postInDB,
29      ...postUpdateProperties,
30    } as Post);
31
32    ... // updates comments
33    ... // updates replies
34  }
```

Il metodo `save` di ogni entità è quindi responsabile di inserire o aggiornare un nuovo oggetto nel database.

Tutti i compiti relativi alla Sentiment Analysis sono invece svolti dalla funzione `updateSentiments` nel file `packages/service/src/workers/sentiment-analysis/Updater.ts`. Essa si occupa di raccogliere i testi non ancora analizzati e sottometterli alla funzione `submitToSentimentAnalysisService` (nel file `package/service/src/workers/sentiment-analysis/SentimentAnalysisSubmitter.ts`). Il vantaggio di avere questa struttura è derivante dal fatto che il servizio di Sentiment Analysis è ben diviso dal resto del codice e quindi un eventuale cambiamento ad esso non richiederebbe alcuna modifica alle restanti parti.

Dopo aver ottenuto i nuovi sentimenti, la funzione si occupa di salvarli (similmente a quanto visto in precedenza) e a ricomputare il sentimento aggregato per i post che hanno ricevuto nuovi commenti:

```
1  // OverallSentimentAnalysis.ts
2  const computeSentiment = (sentiments: Array<Sentiment>):
       number => {
3    let sentimentSum = 0;
4    sentiments.forEach((sentiment) => {
```

```
5       if (sentiment === Sentiment.Positive) {
6         sentimentSum += 1;
7       } else if (sentiment === Sentiment.Negative) {
8         sentimentSum -= 1;
9       }
10    });
11    return sentimentSum / sentiments.length;
12  };
```

La funzione `computeSentiment` effettua semplicemente una media del sentimento contando solamente commenti positivi e negativi (il sentimento *mixed* è particolarmente difficile da interpretare e per la prima versione del software si è deciso di non considerarlo nella computazione del sentimento aggregato).

Capitolo 4

Funzionamento e guida all'utilizzo della piattaforma

In quest'ultima parte di elaborato viene presentata una guida al funzionamento del software finale. Questa parte risulta necessaria in quanto alcuni aspetti dell'utilizzo del software possono risultare complessi ad utenti che non hanno un'ampia conoscenza di tecnologie come Facebook, Instagram, Facebook Login e navigazione all'interno di software protetti da autenticazione. Diverse illustrazioni riportate si riferiscono direttamente all'account che ha collegato pagina Facebook e profilo Instagram di DARE Ravenna.

4.1 Registrazione alla piattaforma

Per la registrazione alla piattaforma, è presente una schermata apposita (4.1) all'indirizzo `/login` che richiede l'email all'utente:

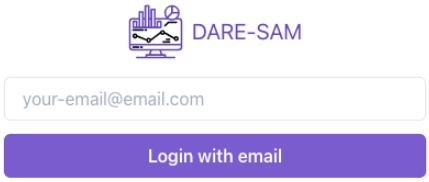

Figura 4.1: Schermata di login di DARE-SAM

Una volta inserito un indirizzo email valido e premuto il bottone *Login with Email*, dopo un iniziale caricamento verrà mostrata la seguente schermata (4.2) che conferma il corretto invio della mail all'indirizzo specificato:

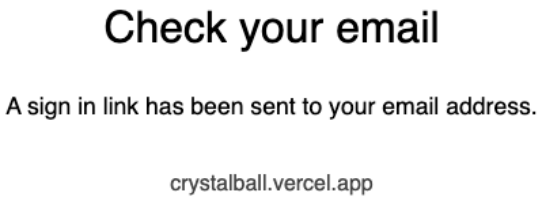

Figura 4.2: Schermata di conferma dell'avvenuto invio della mail all'indirizzo specificato

La mail è lo strumento utilizzato per confermare l'identità dell'utente. Essa conterrà il seguente form (4.1):

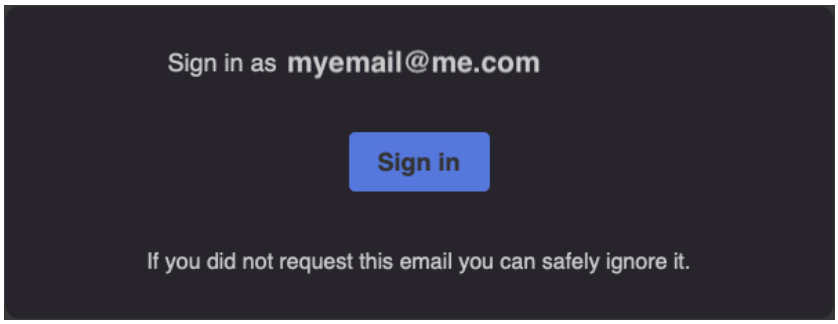

Figura 4.3: Contenuto dell'email inviata da DARE-SAM

in cui l'indirizzo *myemail@me.com* sarà sostituito dall'indirizzo email inserito dall'utente.
Una volta cliccato il bottone *Sign In* all'interno della mail si sarà rimandati alla piattaforma web e, dopo un breve caricamento, si avrà accesso alla pagina di account, spiegata estensivamente nella prossima sezione.

4.2 Gestione dell'account

La gestione dell'account è particolarmente complessa in quanto prevede diversi step successivi che l'utente deve compiere per poter correttamente collegare i propri profili social. Innanzitutto, una volta cliccato il bottone presente nella mail di registrazione (4.1), l'utente si ritroverà sulla pagina di account (visitabile all'indirizzo `/account`). Questa pagina inizialmente ha il seguente contenuto (4.4):

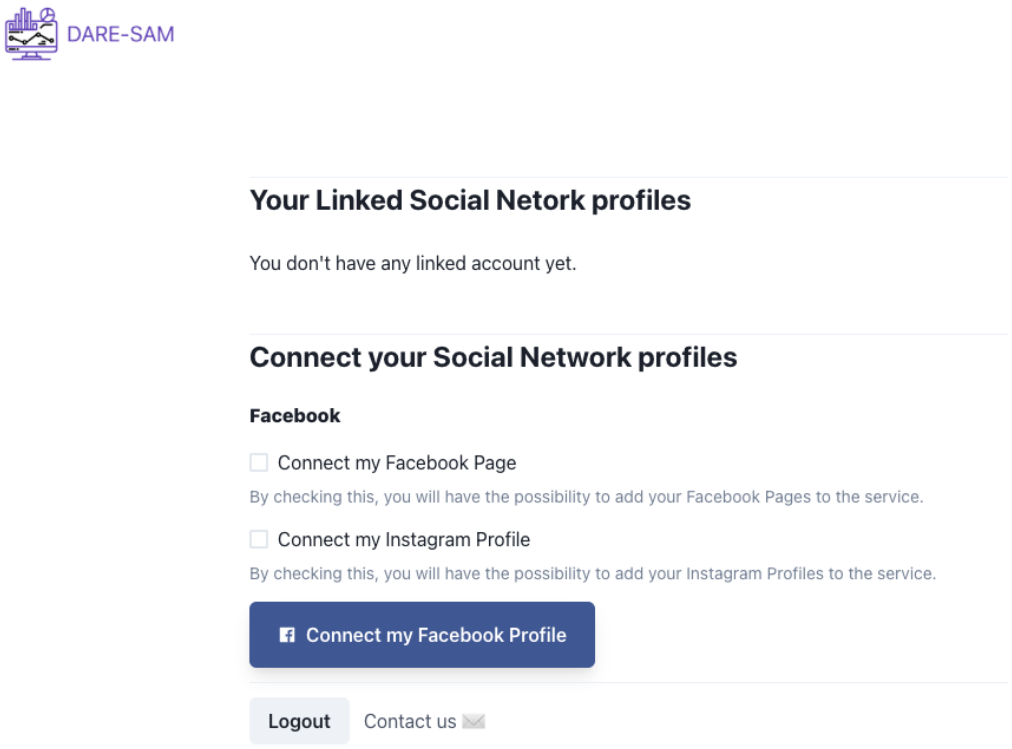

Figura 4.4: Illustrazione della pagina di account iniziale

Come è possibile notare, non sono presenti profili social ed è possibile collegarne nella sezione "Connect your Social Network profiles". Questa sezione presenta due *checkbox*, una per collegare la propria pagina Facebook e una per il proprio profilo Instagram. Una volta selezionate una o entrambe le caselle, sarà possibile cliccare il bottone *Connect my Facebook Profile*, che aprirà un nuovo popup per effettuare l'accesso a Facebook. Una volta cliccato, si verrà reindirizzati alla seguente schermata (4.5):

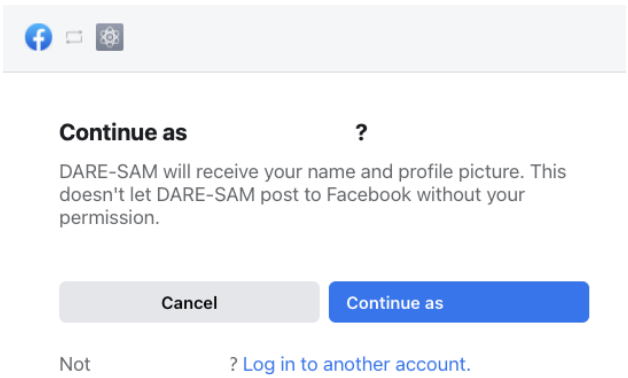

Figura 4.5: Illustrazione del popup di login Facebook

Ciò elenca alcuni permessi che verranno concessi a DARE-SAM, tra cui la possibilità di leggere nome e foto del profilo. Successivamente, una volta cliccato su *Continue as ...* si avranno due ulteriori schermate (4.6 e 4.7):

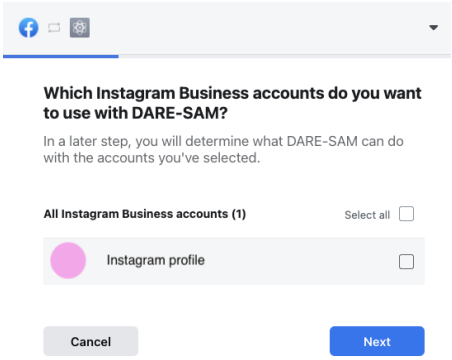

Figura 4.6: Selezione dei profili Instagram Business

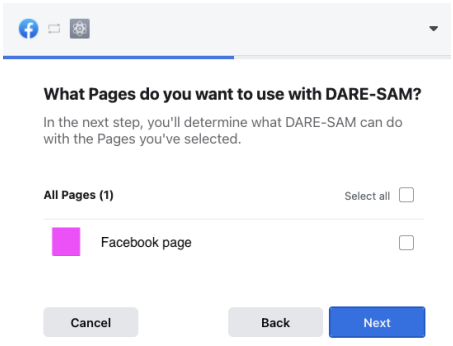

Figura 4.7: Selezione delle pagine Facebook

Qui è possibile selezionare le pagine Facebook e i profili Instagram a cui potenzialmente DARE-SAM può avere accesso. Infine, l'ultima schermata del popup Facebook prima di tornare a DARE-SAM è la seguente (4.8):

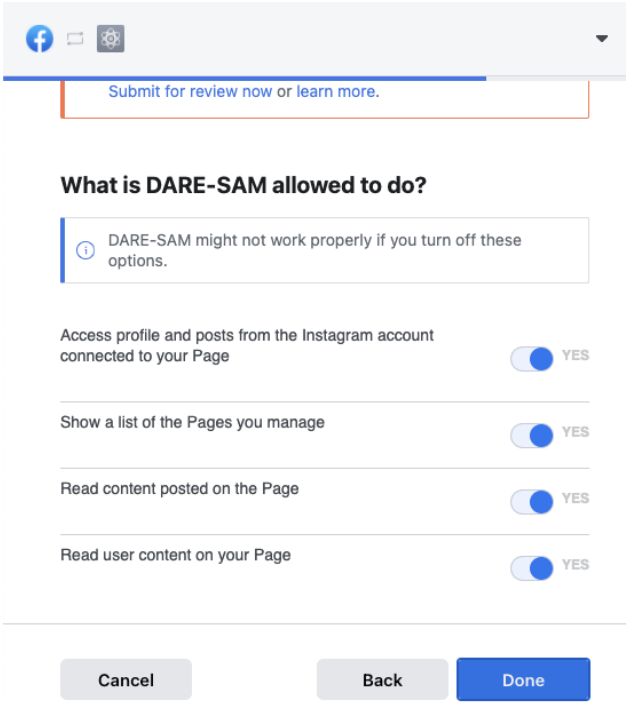

Figura 4.8: Riepilogo dei permessi forniti a DARE-SAM da parte dell'utente

La lista dei seguenti permessi è utile all'utente a capire ciò che DARE-SAM può fare con i dati dei suoi profili social. Come è possibile verificare, tutte le operazioni sono legate alla lettura di post e contenuti: DARE-SAM non può in alcun modo modificare o creare contenuti.
Una volta tornati a DARE-SAM, la pagina di gestione dell'account sarà stata aggiornata e rispecchierà la seguente illustrazione (4.9):

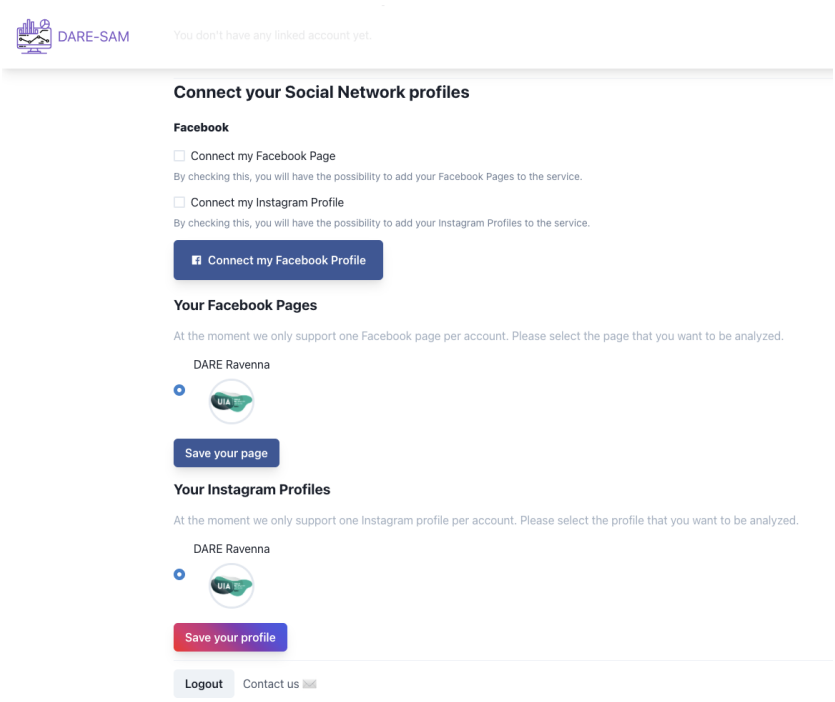

Figura 4.9: Illustrazione della pagina di account una volta collegato il proprio profilo Facebook

Dato che, almeno in una fase iniziale del progetto, non è mai emersa la necessità di collegare più di una singola pagina Facebook o profilo Instagram ad un singolo utente, è presente un limite di una pagina Facebook e un profilo Instagram per account (come illustrato nelle sezioni *Your Facebook Pages* e *Your Instagram Profiles*). Una volta selezionati i profili e salvati si avrà la seguente schermata finale (4.10) che conclude i passaggi richiesti per collegare i profili social a DARE-SAM:

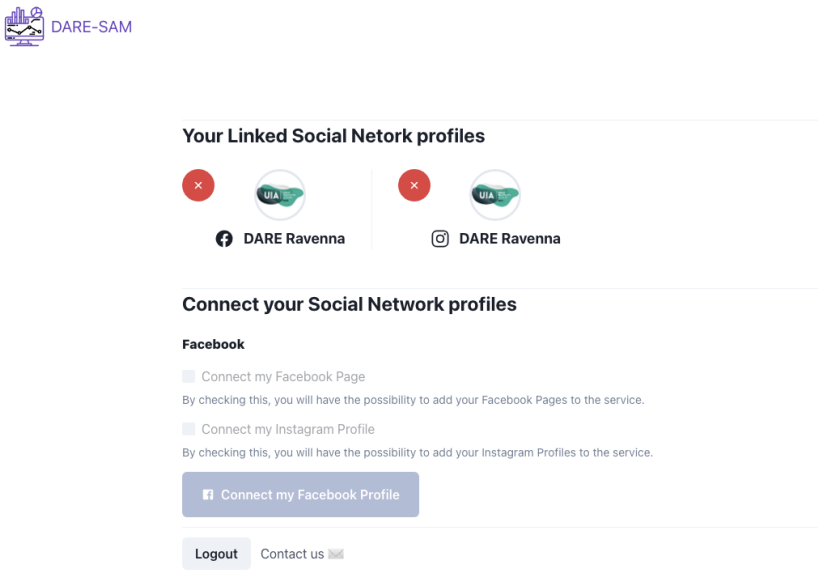

Figura 4.10: Illustrazione della pagina di account una volta collegati i propri profili a DARE-SAM

In questo modo il sistema potrà procedere ad analizzare i profili collegati e aggiornare i dati per presentarli nella pagina di dashboard, discussa ampiamente nella prossima sezione.

4.3 Navigazione della Dashboard

La dashboard è la pagina principale della piattaforma. Essa presenta tutte le informazioni relative ai post pubblicati dai profili social associati all'utente. Inizialmente, se nessun profilo è collegato, non ci sarà nessun post da mostrare e quindi verrà riportato il seguente messaggio (4.11):

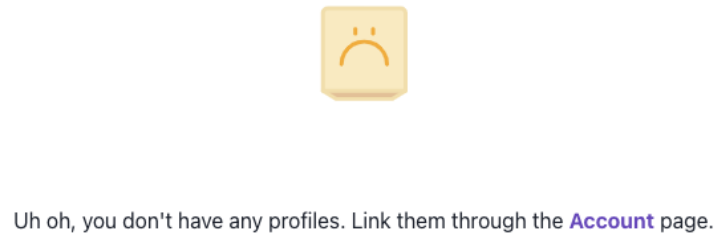

Figura 4.11: Messaggio riportato nella dashboard se non sono presenti profili social collegati

Cliccando sul link viola `Account` si verrà rimandati alla pagina di Account in cui si potrà procedere al collegamento di profili social.
Se invece sono presenti profili social ma non è presente alcun post verrà mostrato il seguente messaggio (4.12):

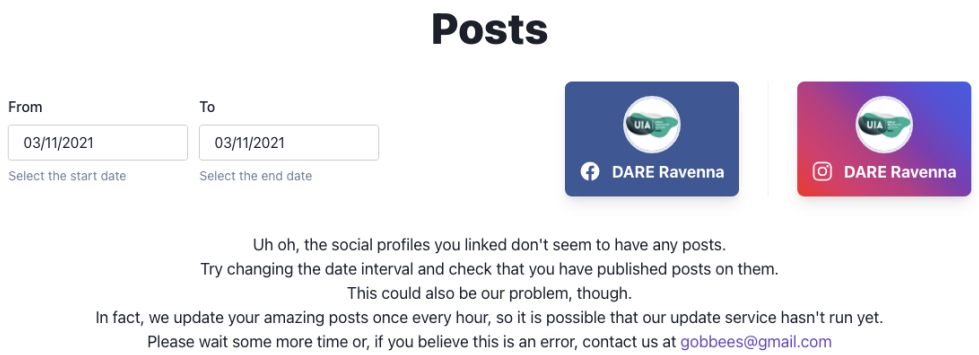

Figura 4.12: Messaggio riportato nella dashboard quando non sono presenti post per i profili collegati

Come riportato, ciò potrebbe essere causato da due problemi:

- non sono presenti post per nessun profilo social collegato per le date scelte nell'intervallo; in questo caso basterà cambiare la data di inizio e fine dell'intervallo;
- il servizio che si occupa di aggiornare i post (Backend Service) non ha ancora eseguito una prima volta dopo aver collegato i profili; ciò è derivante dal fatto che esso esegue una volta all'ora. La soluzione è semplicemente aspettare.

Ora è invece esaminata la dashboard nel caso in cui ci siano post per i profili collegati, ovvero il più interessante. Di seguito è presente l'illustrazone di una parte della tabella (4.13):

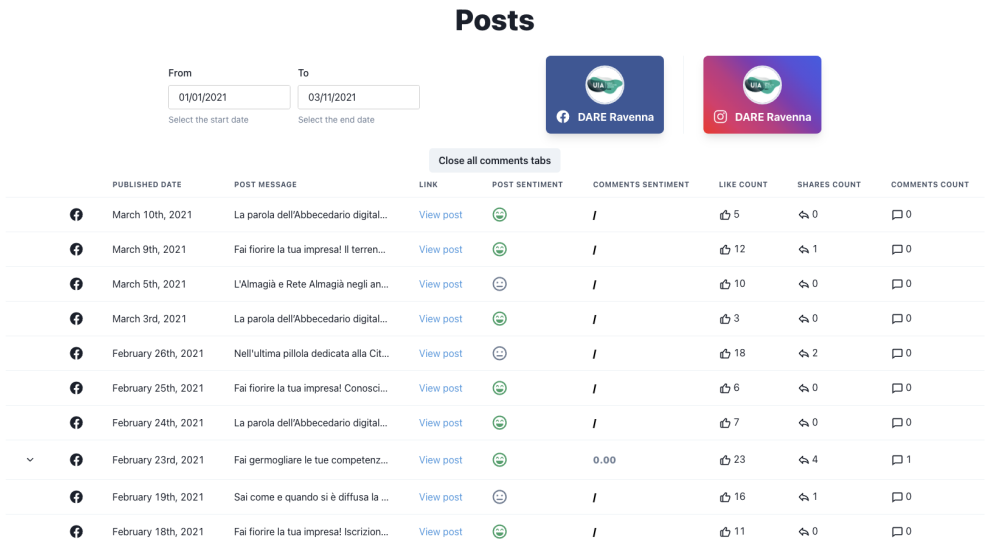

Figura 4.13: Illustrazione della tabella all'interno della dashboard

I campi per ogni post sono:

- data di pubblicazione

- messaggio del post (contenuto testuale)
- link al post
- sentimento rilevato nel messaggio del post, classificato come:
  - emoji sorridente di colore verde: sentimento positivo
  - emoji triste di colore rosso: sentimento negativo
  - emoji dubbiosa di colore giallo: sentimento misto
  - emoji dubbiosa di colore grigio: sentimento neutro
- sentimento aggregato dei commenti ricevuti dal post che può essere:
  - "/" se il post non ha ricevuto nessun commento
  - un numero da -1 a 1 se sono presenti commenti
- numero dei Like ricevuti dal post
- numero delle condivisioni del post (non applicabile per post di Instagram)
- numero di commenti ricevuti dal post.

I post sono ordinati per data di pubblicazione, in modo che quelli più recenti siano in cima alla tabella per una più veloce consultazione. Le funzionalità di filtering presenti per i post sono:

- selezione dei profili singoli (4.14)

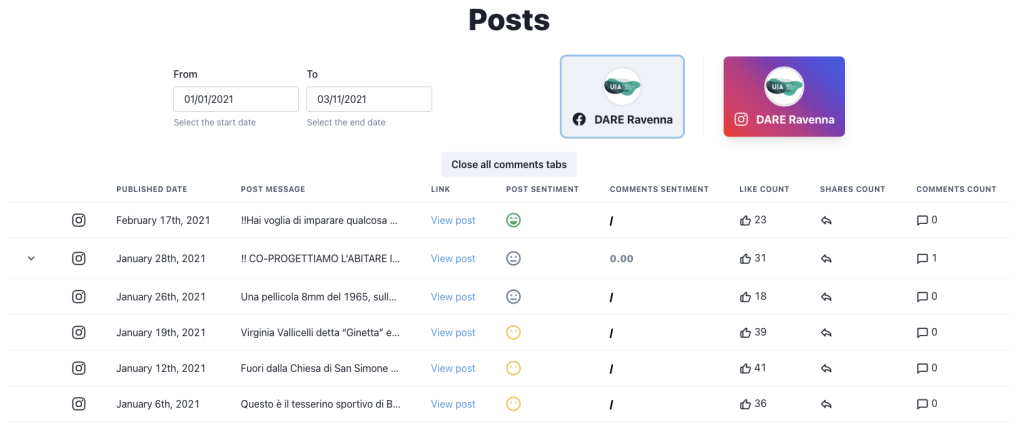

Figura 4.14: Illustrazione della tabella all'interno della dashboard applicando il filtro per profilo

- selezione dell'intervallo di date (4.15):

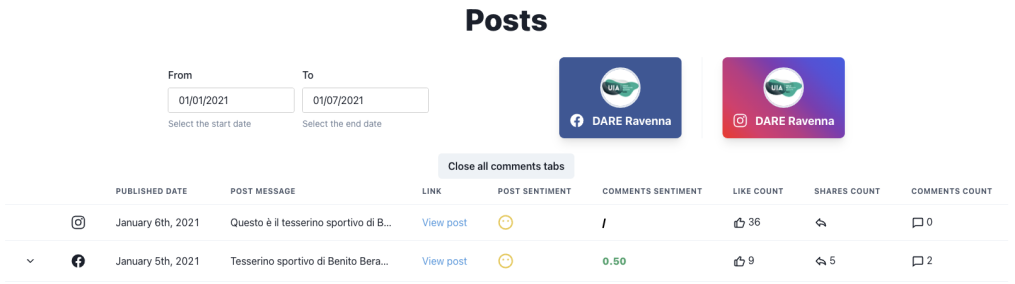

Figura 4.15: Illustrazione della tabella all'interno della dashboard applicando il filtro per intervallo date

Premendo sulla freccia a lato della data per i post che hanno dei commenti è possibile aprire una nuova vista con i commenti di quel post, come visualizzato nell'illustrazione seguente (4.16):

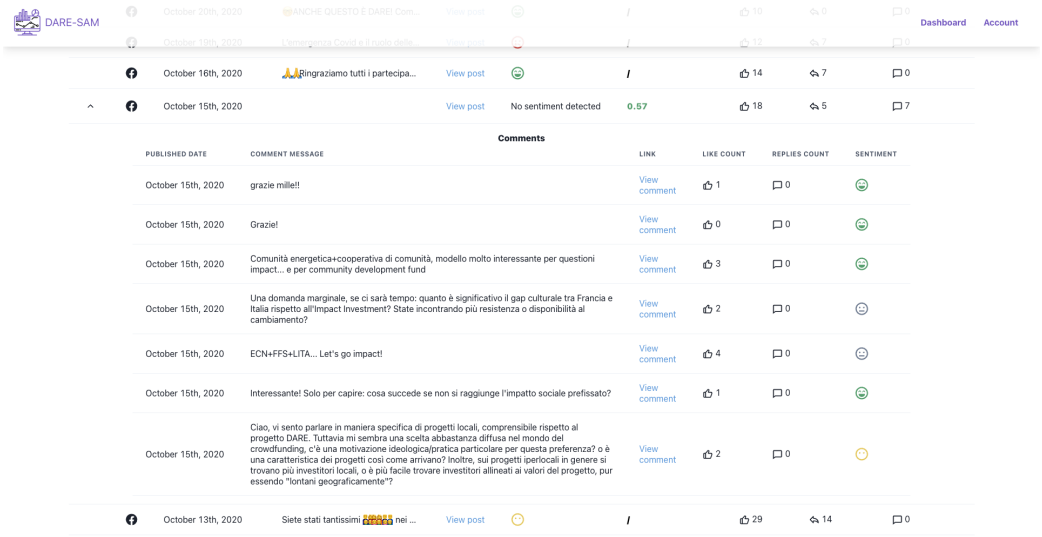

Figura 4.16: Illustrazione della tabella aprendo un post per visualizzarne i commenti

In questa nuova tabella sono presenti le seguenti informazioni per ogni commento:

- data di pubblicazione
- messaggio del commento (contenuto testuale)
- link al commento (non applicabile per commenti di Instagram)
- sentimento rilevato nel messaggio del commento, classificato come:
  - emoji sorridente di colore verde: sentimento positivo
  - emoji triste di colore rosso: sentimento negativo
  - emoji dubbiosa di colore giallo: sentimento misto

  - emoji dubbiosa di colore grigio: sentimento neutro

- numero dei Like ricevuti dal commento
- numero di risposte ricevute dal commento.

È inoltre possibile ordinare questa nuova tabella in base a tutte le colonne in essa presenti semplicemente cliccando sull'intestazione della colonna desiderata. Il primo click effettuerà un ordinamento crescente in base ai valori contenuti in quella colonna, mentre un secondo click effettuerà un ordinamento decrescente. Per semplificare la visualizzazione dei criteri di ordinamento sono presenti frecce ad indicare su quali campi e che tipo di ordinamento è attualmente visualizzato.

Conclusioni e sviluppi futuri

L'obiettivo di questo elaborato era l'implementazione di uno strumento in grado di fornire al team di DARE numerose informazioni in merito ai contenuti da loro pubblicati sui canali social. Ciò ha il vantaggio di permettere una migliore analisi delle opinioni dei cittadini Ravennati in merito a numerose questioni di interesse per il progetto DARE.
Dopo una analisi introduttiva del problema alla ricerca delle modalità con cui ottenere queste informazioni, è subito emersa la necessità di servirsi di tecniche avanzate di Machine Learning e Natural Language Processing, e la nicchia più adatta in una prima versione del software è risultata la Sentiment Analysis. Si è quindi iniziato un ampio lavoro di ricerca e consultazione di numerose fonti per avere una visione ottimale sulle varie soluzioni presenti in letteratura. Da qui è emerso come la presenza di servizi offerti da provider cloud (tra cui quello utilizzato attualmente: *Azure Text Analytics*) sarebbe risultato estremamente vantaggioso in una prima fase del progetto, in quanto avrebbe risparmiato una grande quantità di tempo che verrebbe invece investita nella realizzazione dell'architettura e della piattaforma web.
A seguire, si è svolta una seconda fase di progettazione del sistema partendo dai requisiti funzionali e non. Si sono analizzate le sfide principali e delineate le modalità di accesso ai dati presenti sui social media. Sono quindi venute alla luce le limitazioni che esse comportano e si è iniziato a disegnare possibili soluzioni. Scelta quindi l'architettura finale, ovvero quella che rispecchiava il *trade-off* migliore tra efficienza, prestazioni e scalabilità, si è proceduto alla fase di implementazione.

Nella fase più sostanziosa dell'elaborato si è prestata particolare attenzione alla realizzazione di un software completo, efficiente e facilmente manutenibile. Si sono utilizzate tecnologie all'avanguardia come React e Node.js che hanno introdotto numerosi vantaggi e semplificato notevolmente lo sviluppo. Infine si è svolta una fase di test parziale con il team di DARE, in cui si è testato il corretto funzionamento della piattaforma collegando i profili e verificando la veridicità dei risultati offerti da DARE-SAM.

Sebbene il software risulti funzionante al giorno d'oggi, sono innumerevoli le possibilità di sviluppi ed estensioni future. Questo è dovuto dall'ampiezza del panorama di social media presenti attualmente e alla grandissima mole di informazioni che è possibile raccogliere con essi. Un primo miglioramento potrebbe essere l'introduzione del supporto a Twitter, dato che è presente il profilo ufficiale di DARE Ravenna. Ciò non è purtroppo stato possibile nell'ambito di questo elaborato in quanto avrebbe richiesto una grande quantità di tempo dedicato alla ricerca e all'implementazione della miglior soluzione possibile. Estendendo questo punto, non è difficile da pensare una futura versione della piattaforma che offra il supporto a molti altri social media, tra cui possibilmente YouTube, LinkedIn e tanti altri.
La seconda estensione che potrebbe naturalmente seguire è quella di una maggior elaborazione e analisi dei contenuti già estratti alla ricerca di maggiori informazioni e metriche di interesse per l'utente finale. I metodi per realizzare ciò potrebbero essere vari: partendo da un miglioramento dell'attuale soluzione per Sentiment Analysis per arrivare all'introduzione di altre tecniche di ML e NLP con lo scopo di sfruttare ogni singolo carattere estratto dai social media.
Una maggior quantità di informazioni richiederebbe certamente una maggior quantità di dashboard, grafici e tabelle, e questo è sicuramente uno sviluppo futuro interessante e utile all'utente finale. Sebbene infatti la attuale Dashboard offra già numerose informazioni, siamo consapevoli del fatto che

un'ulteriore fase di *Data Visualization* gioverebbe notevolmente al progetto. Infine, risulta importante creare una completa fase di test automatici [131] del software realizzato. Ciò non è al momento presente e causa maggiore insicurezza nello sviluppo di nuove funzionalità avendo la certezza di non introdurre malfunzionamenti. Partendo dai più semplici *Unit test* [132] per raggiungere successivamente fasi avanzate come *Integration test* [132] e *End-to-end test* [132], essa è certamente una miglioria che il software necessita per garantire elevati standard di qualità ed esperienza utente.

Ringraziamenti

Inizio ringraziando la mia famiglia per avermi spinto alla scelta di questo percorso di studi rivelatosi la scelta ideale per i miei obiettivi di vita; i nuovi amici incontrati in questi anni che mi hanno continuamente motivato a dare il massimo per realizzare progetti ambiziosi ed esperienze che ho avuto la fortuna di poter vivere; i miei più vecchi amici, con cui è sempre bello ritrovarsi dopo una giornata di studio intenso per condividere bei momenti; e infine Sara, che in questo anno difficile segnato dalle incertezze mi ha aiutato tanto a mantenere la voglia di fare e di impegnarmi.

In merito all'elabotato, un sincero ringraziamento va alla Dott.ssa Silvia Mirri e alla Dott.ssa Catia Prandi per la grande disponibilità e la continua guida durante lo sviluppo del progetto e la stesura dell'elaborato che segna la fine di questo bellissimo percorso.

Bibliografia

[1] Our World in data. Internet. https://ourworldindata.org/internet, visitato in Febbraio 2021.

[2] Statista.com. Number of monthly active facebook users worldwide as of 3rd quarter 2020. https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide, visitato in Gennaio 2021.

[3] Google. Youtube for press, . https://blog.youtube/press/, visitato in Febbraio 2021.

[4] Esteban Ortiz-Ospina. The rise of social media. https://ourworldindata.org/rise-of-social-media, visitato in Gennaio 2021.

[5] Suzan Wold. Covid-19 is changing how, why and how much we're using social media. https://www.digitalcommerce360.com/2020/09/16/covid-19-is-changing-how-why-and-how-much-were-using-social-media/, visitato in Marzo 2021.

[6] Irfan Ahmad. How much data is generated every minute? https://www.socialmediatoday.com/news/how-much-data-is-generated-every-minute-infographic-1/525692/#:~:text=%22Over%202.5%20quintillion%20bytes%20of,for%20every%20person%20on%20earth.%22, visitato in Febbraio 2021.

[7] Mark Zuckerberg. Understanding facebook's business model. https://about.fb.com/news/2019/01/understanding-facebooks-business-model/, visitato in Gennaio 2021.

[8] Jason Kincaid. Facebook activates like button; friendfeed tires of sincere flattery. https://techcrunch.com/2009/02/09/facebook-activates-like-button-friendfeed-tires-of-sincere-flattery/, visitato in Marzo 2021.

[9] Facebook. Reactions, . https://en.facebookbrand.com/facebookapp/assets/reactions, visitato in Gennaio 2021.

[10] Linkedin. Use linkedin reactions. https://www.linkedin.com/help/linkedin/answer/101466/use-linkedin-reactions?lang=en, visitato in Gennaio 2021.

[11] Dare project overview. https://uia-initiative.eu/en/uia-cities/ravenna, visitato in Febbraio 2021.

[12] What is urban innovative actions? https://uia-initiative.eu/en/about-us/what-urban-innovative-actions, visitato in Febbraio 2021.

[13] Comune di Ravenna. Darsena di cittá. https://www.turismo.ra.it/nature-and-seaside/darsena-citta-ravenna/, visitato in Febbraio 2021.

[14] Facebook. React official website, . https://reactjs.org, visitato in Febbraio 2021.

[15] Vercel. Next.js official website, . https://nextjs.org, visitato in Febbraio 2021.

[16] OpenJS Foundation. Node.js official website. https://nodejs.org/en/, visitato in Febbraio 2021.

[17] Fabrizio Sebastiani Stefano Baccianella, Andrea Esuli. Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. https://www.researchgate.net/publication/220746537_SentiWordNet_30_An_Enhanced_Lexical_Resource_for_Sentiment_Analysis_and_Opinion_Mining.

[18] Kurtis Pykes. Part of speech tagging for beginners. https://towardsdatascience.com/part-of-speech-tagging-for-beginners-3a0754b2ebba, visitato in Febbraio 2021.

[19] Sidath Asiri. Machine learning classifiers. https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623, visitato in Gennaio 2021.

[20] Jason Brownlee. A gentle introduction to the bag-of-words model. https://machinelearningmastery.com/gentle-introduction-bag-words-model, visitato in Gennaio 2021.

[21] NLTK team. Learning to classify text, . https://www.nltk.org/book/ch06.html, visitato in Gennaio 2021.

[22] NLTK team. Nltk official documentation, . https://www.nltk.org, visitato in Gennaio 2021.

[23] Treccani. Corpora di italiano. https://www.treccani.it/enciclopedia/corpora-di-italiano_%28Enciclopedia-dell%27Italiano%29/, visitato in Marzo 2021.

[24] NLTK team. Nltk corpora, . http://www.nltk.org/nltk_data/, visitato in Gennaio 2021.

[25] TextBlob team. Textblob official documentation, . https://textblob.readthedocs.io/en/dev/, visitato in Gennaio 2021.

[26] Computational Linguistics and University of Antwerp Psycholinguistics Research Center. Pattern. https://github.com/clips/pattern, visitato in Gennaio 2021.

[27] TextBlob. https://github.com/sloria/TextBlob/blob/dev/textblob/en/sentiments.py, visitato in Gennaio 2021.

[28] Rohith Gandhi. Naive bayes classifier. https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c, visitato in Gennaio 2021.

[29] spaCy team. spacy official website. https://spacy.io/, visitato in Gennaio 2021.

[30] Cython team. Cython official website, . https://cython.org/, visitato in Gennaio 2021.

[31] MonkeyLearn. official website. https://monkeylearn.com/, visitato in Gennaio 2021.

[32] Raul Garreta Tompson. https://www.quora.com/How-can-I-use-Spacy-to-perform-sentiment-analysis, visitato in Gennaio 2021.

[33] C.J. Hutto. Vader sentiment, . https://github.com/cjhutto/vaderSentiment, visitato in Gennaio 2021.

[34] C.J. Hutto. Vader emojis lexicon, . https://github.com/cjhutto/vaderSentiment/blob/master/vaderSentiment/emoji_utf8_lexicon.txt, visitato in Gennaio 2021.

[35] C.J. Hutto, . https://github.com/cjhutto/vaderSentiment/blob/master/vaderSentiment/vaderSentiment.py#L46, visitato in Gennaio 2021.

[36] Mozilla, . `https://developer.mozilla.org/en-US/docs/Web/JavaScript`, visitato in Febbraio 2021.

[37] Drus and Khalid. Sentiment analysis in social media and its application: Systematic literature review. *The Fifth Information Systems International Conference*, (2019).

[38] Ali et al. Sentiment analysis as a service: A social media based sentiment analysis framework. *IEEE International Conference on Web Services (ICWS), Honolulu, HI, USA: IEEE*, (2017).

[39] Hassan et al. Sentiment analysis of social networking sites (sns) data using machine learning approach for the measurement of depression. *International Conference on Information and Communication Technology Convergence (ICTC), Jeju, South Korea: IEEE*, (2017).

[40] Brandon Joyce and Deng. Sentiment analysis of tweets for the 2016 us presidential election. *IEEE MIT Undergraduate Research Technology Conference (URTC), Cambridge, MA, USA: IEEE*, (2017).

[41] Cambridge Dictionary. Meaning of 'be the bomb'. `https://dictionary.cambridge.org/dictionary/english/be-the-bomb`, visitato in Gennaio 2021.

[42] Peter Norvig. How to write a spelling corrector. `https://norvig.com/spell-correct.html`, visitato in Gennaio 2021.

[43] Instagram Engineering. Emojineering part 1: Machine learning for emoji trends. `https://bit.ly/3pdyy27`, visitato in Gennaio 2021.

[44] Petra Kralj Novak, Jasmina Smailović, Borut Sluban, and Igor Mozetič. Sentiment of emojis. *PLOS ONE*, 10(12):1–22, 12 2015. doi: 10.1371/journal.pone.0144296. URL `https://doi.org/10.1371/journal.pone.0144296`.

[45] Shuyo Nakatani. Language detection library for java, 2010. URL https://github.com/shuyo/language-detection.

[46] A. G. Prasad, S. Sanjana, S. M. Bhat, and B. S. Harish. Sentiment analysis for sarcasm detection on streaming short text data. In *2017 2nd International Conference on Knowledge Engineering and Applications (ICKEA)*, pages 1–5, 2017. doi: 10.1109/ICKEA.2017.8169892.

[47] Harshdeep Singh. Understanding gradient boosting machines. https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab, visitato in Febbraio 2021.

[48] Tony Yiu. Understanding random forest. https://towardsdatascience.com/understanding-random-forest-58381e0602d2, visitato in Febbraio 2021.

[49] Mickel Hoang, Oskar Alija Bihorac, and Jacobo Rouces. Aspect-based sentiment analysis using BERT. In *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, pages 187–196, Turku, Finland, September–October 2019. Linköping University Electronic Press. URL https://www.aclweb.org/anthology/W19-6120.

[50] Microsoft Azure. https://docs.microsoft.com/en-us/azure/cognitive-services/text-analytics/quickstarts/client-libraries-rest-api?tabs=version-3-1&pivots=programming-language-javascript#opinion-mining, visitato in Gennaio 2021.

[51] Google Cloud Platform. https://cloud.google.com/natural-language/docs/analyzing-entity-sentiment, visitato in Gennaio 2021.

[52] Facebook. What are the different page roles and what can they do?, . https://www.facebook.com/help/289207354498410, visitato in Febbraio 2021.

[53] Imperva. Web scraping. https://www.imperva.com/learn/application-security/web-scraping-attack/, visitato in Marzo 2021.

[54] Cloudflare. What is a web crawler? — how web spiders work. https://www.cloudflare.com/learning/bots/what-is-a-web-crawler/, visitato in Marzo 2021.

[55] Facebook. Terms of service, . https://www.facebook.com/legal/terms, visitato in Febbraio 2021.

[56] Facebook. Graph api official documentation, . https://developers.facebook.com/docs/graph-api, visitato in Febbraio 2021.

[57] Facebook. Instagram graph api official documentation, . https://developers.facebook.com/docs/instagram-api, visitato in Febbraio 2021.

[58] Facebook. App development, . https://developers.facebook.com/docs/development/, visitato in Febbraio 2021.

[59] Facebook. Graph api permissions reference, . https://developers.facebook.com/docs/permissions/reference, visitato in Febbraio 2021.

[60] Facebook. instagram_basic permission reference, . https://developers.facebook.com/docs/permissions/reference/instagram_basic, visitato in Febbraio 2021.

[61] Facebook. Graph api rate limits, . https://developers.facebook.com/docs/graph-api/overview/rate-limiting, visitato in Febbraio 2021.

[62] Facebook. Platform terms, . https://developers.facebook.com/terms, visitato in Marzo 2021.

[63] Facebook. App review, . https://developers.facebook.com/docs/app-review/introduction, visitato in Marzo 2021.

[64] Facebook. App review introduction - app models, . https://developers.facebook.com/docs/app-review/introduction, visitato in Marzo 2021.

[65] Facebook. App review - before you submit, . https://developers.facebook.com/docs/app-review/before-you-submit, visitato in Marzo 2021.

[66] Facebook. App review - submitting for review, . https://developers.facebook.com/docs/app-review/submission-guide, visitato in Marzo 2021.

[67] Michele de Pascale. Official facebook page. https://www.facebook.com/micheledepascalesindaco, visitato a Febbraio 2021.

[68] Google. Cloud natural language website, . https://cloud.google.com/natural-language, visitato in Febbraio 2021.

[69] Google. Cloud natural language entity sentiment analysis documentation, . https://cloud.google.com/natural-language/docs/reference/rest/v1/documents/analyzeEntitySentiment, visitato in Febbraio 2021.

[70] Google. Cloud natural language language support, . https://cloud.google.com/natural-language/docs/languages, visitato in Febbraio 2021.

[71] Google. Cloud natural language pricing, . https://cloud.google.com/natural-language/pricing, visitato in Febbraio 2021.

[72] Microsoft. Azure text analytic website, . https://docs.microsoft.com/en-us/azure/cognitive-services/text-analytics/, visitato in Febbraio 2021.

[73] Microsoft. Azure text analytics language support, . https://docs.microsoft.com/en-us/azure/cognitive-services/text-analytics/language-support?tabs=sentiment-analysis, visitato in Febbraio 2021.

[74] Microsoft. Azure text analytics pricing, . https://azure.microsoft.com/en-us/pricing/details/cognitive-services/text-analytics/, visitato in Febbraio 2021.

[75] IBM. Watson natural language understanding website, . https://www.ibm.com/cloud/watson-natural-language-understanding, visitato in Febbraio 2021.

[76] IBM. Watson natural language understanding sentiment analysis feature, . https://cloud.ibm.com/apidocs/natural-language-understanding?code=node#sentiment, visitato in Febbraio 2021.

[77] IBM. Watson natural language understanding language support, . https://cloud.ibm.com/docs/natural-language-understanding?topic=natural-language-understanding-language-support, visitato in Febbraio 2021.

[78] IBM. Watson natural language understanding pricing, . https://www.ibm.com/cloud/watson-natural-language-understanding/pricing, visitato in Febbraio 2021.

[79] Amazon. Comprehend website, . https://aws.amazon.com/comprehend, visitato in Febbraio 2021.

[80] Amazmon. Comprehend pricing. https://aws.amazon.com/comprehend/pricing/, visitato in Febbraio 2021.

[81] Microsoft. Typescript official website, . https://www.typescriptlang.org/, visitato in Febbraio 2021.

[82] Microsoft. Visual studio code official website, . https://code.visualstudio.com/, visitato in Febbraio 2021.

[83] JetBrains. Webstorm official website. https://www.jetbrains.com/webstorm/, visitato in Febbraio 2021.

[84] Facebook. React native official website, . https://reactnative.dev/, visitato in Febbraio 201.

[85] Google. Tensorflow.js official website, . https://www.tensorflow.org/js, visitato in Febbraio 2021.

[86] NPM Trends. https://www.npmtrends.com/angular-vs-jquery-vs-react-vs-vue, visitato in Febbraio 2021.

[87] Facebook. Introducing jsx, . https://reactjs.org/docs/introducing-jsx.html, visitato in Febbraio 2021.

[88] Daniel Brain. Jsx is a stellar invention, even with react out of the picture. https://bluepnume.medium.com/jsx-is-a-stellar-invention-even-with-react-out-of-the-picture-c597187134b7, visitato in Febbraio 2021.

[89] Mozilla. Html: Hypertext markup language, . https://developer.mozilla.org/en-US/docs/Web/HTML, visitato in Febbraio 2021.

[90] Marius Schulz. Bundling and minification: an introduction. https://mariusschulz.com/blog/bundling-and-minification-an-introduction#bundling-combining-assets-together, visitato in Febbraio 2021.

[91] Bartosz Szczeciński. What's server side rendering and do i need it? https://medium.com/@baphemot/

whats-server-side-rendering-and-do-i-need-it-cb42dc059b38, visitato in Febbraio 2021.

[92] Phil Hawksworth. What is a static site generator? and 3 ways to find the best one. https://www.netlify.com/blog/2020/04/14/what-is-a-static-site-generator-and-3-ways-to-find-the-best-one/, visitato in Febbraio 2021.

[93] Jason Miller & Addy Osmani. Rendering on the web: Client side rendering. https://developers.google.com/web/updates/2019/02/rendering-on-the-web#csr, visitato in Febbraio 2021.

[94] Vercel. Next.js routing introduction, . https://nextjs.org/docs/routing/introduction, visitato in Febbraio 2021.

[95] Vercel. Next.js api routes introduction, . https://nextjs.org/docs/api-routes/introduction, visitato in Febbraio 2021.

[96] Nextauth.js official website. https://next-auth.js.org/, visitato in Febbraio 2021.

[97] Pthukaraka. Why magic link authentication? https://medium.com/authenticate/why-magic-link-authentication-4cc0178d5515, visitato in Febbraio 2021.

[98] V8 official website. https://v8.dev.

[99] Postgresql official website. https://www.postgresql.org/, visitato in Febbraio 2021.

[100] Amazon. Dynamodb official website, . https://aws.amazon.com/dynamodb/, visitato in Febbraio 2021.

[101] Google. Firebase firestore official website, . https://firebase.google.com/products/firestore, visitato in Febbraio 2021.

[102] Typeorm official website. https://typeorm.io/#/, visitato in Febbraio 2021.

[103] Mario Hoyos. What is an orm and why you should use it. https://blog.bitsrc.io/what-is-an-orm-and-why-you-should-use-it-b2b6f75f5e2a, visitato in Marzo 2021.

[104] Vercel. Next.js deployment options: Node.js server, . https://nextjs.org/docs/deployment#nodejs-server, visitato in Febbraio 2021.

[105] Adam Bavosa. What is a serverless function? https://www.pubnub.com/blog/what-is-a-serverless-function/, visitato in Febbraio 2021.

[106] Vercel. Vercel official website, . https://vercel.com/home, visitato in Febbraio 2021.

[107] Akamai. What does cdn stand for? https://www.akamai.com/uk/en/cdn/what-is-a-cdn.jsp, visitato in Febbraio 2021.

[108] Vercel. Vercel platform pricing, . https://vercel.com/pricing, visitato in Febbraio 2021.

[109] Dokku official website. https://dokku.com/, visitato in Febbraio 2021.

[110] Microsoft. What is a paas, . https://azure.microsoft.com/en-us/overview/what-is-paas/, visitato in Febbraio 2021.

[111] Docker. Docker official website. https://docker.com/, visitato in Febbraio 2021.

[112] DigitalOcean. Digitalocean official website, . https://www.digitalocean.com/, visitato in Febbraio 2021.

[113] DigitalOcean. Digitalocean droplet, . https://www.digitalocean.com/products/droplets/, visitato in Febbraio 2021.

[114] DigitalOcean. Basic droplets pricing, . https://www.digitalocean.com/pricing/#basic-droplets, visitato in Febbraio 2021.

[115] Amazon. Amazon relational database service (rds) official website, . https://aws.amazon.com/rds/, visitato in Febbraio 2021.

[116] Yarn official website, . https://yarnpkg.com/, visitato in Febbraio 2021.

[117] Npm official website. https://npmjs.com, visitato in Febbraio 2021.

[118] Yarn workspaces official documentation, . https://yarnpkg.com/features/workspaces, visitato in Febbraio 2021.

[119] Yarn definition of monorepository. https://yarnpkg.com/advanced/lexicon#monorepository, visitato in Febbraio 2021.

[120] Formik official website. https://formik.org/, visitato in Febbraio 2021.

[121] Twilio SendGrid. Sendgrid official website. https://sendgrid.com/, visitato in Febbraio 2021.

[122] Facebook. Facebook login for the web with the javascript sdk, . https://developers.facebook.com/docs/facebook-login/web/, visitato in Dicembre 2020.

[123] Google. Firebase authentication official website, . https://firebase.google.com/docs/auth, visitato in Dicembre 2020.

[124] Google. Firebase official website, . https://firebase.google.com/, visitato in Dicembre 2020.

[125] Facebook. Building your own hooks, . https://reactjs.org/docs/hooks-custom.html, visitato in Gennaio 2021.

[126] Tanner Linsley. React query official website, . https://react-query.tanstack.com/, visitato in Gennaio 2021.

[127] Tanner Linsley. React table official website, . https://react-table.tanstack.com/, visitato in Gennaio 2021.

[128] Facebook. Graph api explorer website, . https://developers.facebook.com/tools/explorer, visitato in Dicembre 2020.

[129] Facebook. Using the graph api - traversing paged results, . https://developers.facebook.com/docs/graph-api/using-graph-api/v2.1#paging, visitato in Dicembre 2020.

[130] PostgreSQL. Uuid type. https://www.postgresql.org/docs/12/datatype-uuid.html, visitato in Febbraio 2021.

[131] Atlassian. Automated software testing. https://www.atlassian.com/continuous-delivery/software-testing/automated-testing, visitato in Febbraio 2021.

[132] Vitali Zaidman. An overview of javascript testing in 2020. https://medium.com/welldone-software/an-overview-of-javascript-testing-7ce7298b9870, visitato in Febbraio 2021.

[133] Lexalytics. Sentiment analysis explained. https://www.lexalytics.com/technology/sentiment-analysis, visitato in Gennaio 2021.

[134] Monkeylearn. Sentiment analysis: A definitive guide, . https://monkeylearn.com/sentiment-analysis, visitato in Gennaio 2021.

[135] Monkeylearn. Top sentiment analysis apis (saas & open source), . https://monkeylearn.com/blog/sentiment-analysis-apis, visitato in Gennaio 2021.