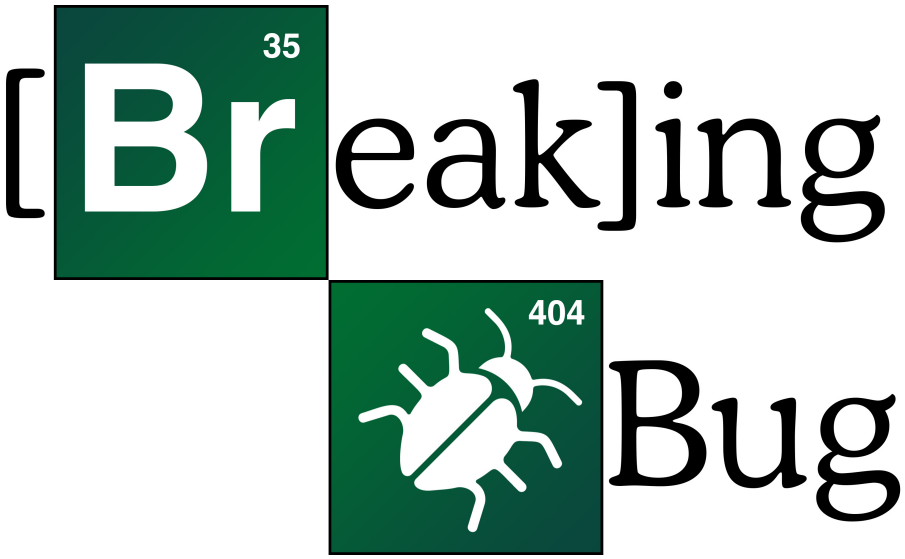

Manuale dello Sviluppatore

Gruppo *Breaking Bug* - Progetto *AJarvis*

breakingbug.swe@gmail.com

Informazioni sul documento

Versione	1.0.0
Redazione	Giacomo Del Moro
Verifica	Tommaso Sotte
	Alessandro Zangari
	Giovanni Righi
Responsabile	Tommaso Loss
Uso	Esterno
Destinatari	*Breaking Bug*
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	zero12 S.r.l.

Registro delle modifiche

Versione	Data	Autore	Ruolo	Descrizione
1.0.0	2018-06-12	Tommaso Loss	Responsabile	Approvazione per il rilascio
0.5.0	2018-06-12	Giacomo Del Moro	Redattore	Ampliata sezione §6.1
0.4.0	2018-06-12	Tommaso Sotte	Verificatore	Nuova verifica del documento
0.3.0	2018-06-11	Alessandro Zangari	Verificatore	Verifica finale del documento
0.2.4	2018-06-11	Giacomo Del Moro	Redattore	Dettagliata §3
0.2.3	2018-06-10	Tommaso Loss	Redattore	Aggiunti dettagli sul text-mining §6.1
0.2.2	2018-06-10	Alessandro Zangari	Redattore	Aggiunti punti di estensione §6.7
0.2.1	2018-05-22	Giacomo Del Moro	Redattore	Correzione grafici sezione §6 e §5
0.2.0	2018-04-29	Giacomo Del Moro	Responsabile	Approvazione per il rilascio
0.1.0	2018-04-27	Paolo Rizzardo	Verificatore	Verifica del documento
0.0.7	2018-04-24	Tommaso Sotte	Redattore	Aggiornamento appendice
0.0.6	2018-04-24	Tommaso Sotte	Redattore	Stesura sezione §6
0.0.5	2018-04-07	Matteo Stabilini	Redattore	Stesura sezione §5
0.0.4	2018-04-02	Matteo Stabilini	Redattore	Aggiornamento appendice

0.0.3	2018-03-30	Tommaso Loss	Redattore	Stesura sezione §4
0.0.2	2018-03-28	Tommaso Loss	Redattore	Stesura sezioni §2 e §3
0.0.1	2018-03-27	Paolo Rizzardo	Redattore	Creazione documento, stesura sezione §1

Indice

1 Introduzione

1.1 Obiettivo del documento

Il documento si propone come guida introduttiva del software *AJarvis*, indirizzata agli sviluppatori che vorranno contribuire ad adattarlo e migliorarlo. Vengono spiegate le tecnologie interessate, l'architettura in dettaglio e come agire su di essa per estenderne le funzionalità.

1.2 Scopo del prodotto

Lo scopo del prodotto è creare un'applicazione di supporto alla visualizzazione dello stato di avanzamento di un progetto, aggiornato ad ogni stand-up giornaliero.
L'applicazione deve essere in grado di registrare gli *stand-up giornalieri*$_{\text{G}}$ di vari progetti. Le registrazioni verranno poi elaborate in *Cloud*$_{\text{G}}$ per comprenderne i dialoghi, e successivamente verrà analizzato il contenuto da un algoritmo di *text-mining*$_{\text{G}}$. Tutti i dati ottenuti serviranno per fornire un rapporto sullo stato del progetto.
L'applicazione sarà composta principalmente dall'interazione tra le due seguenti componenti:

- Interfaccia web per la registrazione dell'audio e la reportistica delle analisi realizzate;
- Servizi cloud, basati su tecnologia Google Cloud Platform, per l'analisi dei dati.

L'assistente dovrà essere sviluppato in lingua italiana o in alternativa in lingua inglese.

1.3 Note esplicative

Allo scopo di evitare ambiguità a lettori esterni si aggiunge in appendice un glossario dei termini utilizzati nel presente documento che verranno segnalati con una G a pedice.

2 Requisiti di sistema

AJarvis è una applicazione web compatibile con qualsiasi dispositivo, per sua natura però necessita di browser e sistemi operativi appropriati.

2.1 Browser

I seguenti browser sono sicuramente supportati da *AJarvis*:

- Google Chrome 64
- Apple Safari 11
- Mozilla Firefox 60

2.2 Sistemi operativi

AJarvis è compatibile con i seguenti sistemi operativi:

- Ubuntu 18.04 LTS
- Microsoft Windows 10, April 2018 Update (1803)
- Apple macOS 10.13 High Sierra

3 Procedura di installazione

Per poter lavorare su *AJarvis* è necessario seguire i seguenti passi. In questo modo verrà predisposto l'ambiente di sviluppo.

1. **Node.js**: Installazione di node.js, versione LTS 8.10.0 o superiore, dal link: https://nodejs.org/it/download/
2. **Npm**: Verificare la disponibilità del comando `npm` da terminale, tramite il comando `npm -v`. In caso negativo, scaricare *npm* da questo link: https://www.npmjs.com/get-npm
3. **Repository**: Clonare la seguente repository di Github: https://github.com/BreakingBugSwe/ajarvis
4. **Installazione delle dipendenze**: Da linea di comando, spostarsi nella cartella principale ed eseguire `npm install` per installare ed aggiornare le dipendenze.
5. **Creare il progetto su Google Cloud Platform**:
   (a) Accedere a Google Cloud Platform e visualizzare il pannello dei progetti dalla barra di navigazione principale;
   (b) Cliccare *Seleziona un progetto* e crearne uno nuovo.
6. **Creare gli indici per Google Cloud Datastore**: Datastore utilizza degli indici per velocizzare le query. *AJarvis* è provvisto di un file $YAML_G$ contenente le proprietà da indicizzare. Una volta creato il progetto sul Datastore effettuare i seguenti passi:
   (a) Da linea di comando spostarsi all'interno della cartella `webserver/lib/gcp`;
   (b) Eseguire `gcloud config set project nome_progetto_target`;
   (c) Eseguire `gcloud datastore create-indexes index.yaml`;

   Per ulteriori informazioni fare riferimento alla documentazione ufficiale: https://cloud.google.com/datastore/docs/tools/indexconfig#Datastore_Updating_indexes
7. **Avvio del server**: Da linea di comando, sempre dalla cartella `webserver/`, eseguire `npm start`. A questo punto *AJarvis* sarà disponibile all'indirizzo:

   http://localhost:3000

   Per avviare il server su una porta specifica, specificare la variabile d'ambiente `PORT` prima di avviare il server (in Windows con il comando: `env:PORT=<port> npm start`, su Unix: `PORT=<port> npm start`). La porta di default è `3000`.

4 Tecnologie interessate

4.1 Google Cloud Platform

Google Cloud Platform è un insieme di servizi di computazione che operano sulla stessa infrastruttura usata da Google per i prodotti dedicati agli utenti. Insieme a un gruppo di strumenti di gestione, la piattaforma provvede una serie di servizi cloud modulari che vengono utilizzate da *AJarvis* per la gestione del database di progetti e registrazione, per la conversione da audio a testo e infine per un'analisi preliminare del testo stesso. E' stato scelto su indicazione dell'azienda proponente zero12 S.r.l.

4.1.1 Google Cloud Storage

Permette l'archiviazione di oggetti unificata per sviluppatori e aziende, con supporto alla distribuzione di dati in tempo reale, all'analisi e all'archiviazione definitiva dei dati.
Viene utilizzato per archiviare i file delle registrazioni audio, troppo pesanti per Google Cloud Datastore.

4.1.2 Google Cloud Datastore

Google Cloud Datastore è un servizio di database non relazionali altamente scalabile offerti da Google sulla sua piattaforma cloud.
Viene utilizzato per memorizzare tutti i dati relativi a progetti, stand-up e all'elaborazione di queste ultimi.

4.1.3 Google Cloud Speech API

L'API Google Cloud Speech consente agli sviluppatori la conversione di audio in testo applicando modelli di rete neurale.
Viene utilizzata per convertire l'audio degli stand-up in testo.

4.1.4 Google Cloud Natural Language API

L'API Google Cloud Natural Language permette di rivelare la struttura e il significato del testo analizzato.
Viene utilizzata per ottenere la lemmatizzazione e l'analisi grammaticale del testo trascritto delle stand-up, che serviranno come input al processo di *text-mining*$_{\text{G}}$.

4.2 Librerie esterne

Listiamo e descriviamo qui le librerie più importanti di cui facciamo uso nel prodotto. Sono tutte librerie open source, disponibili su GitHub e installabili tramite il gestore di pacchetti *npm*.

4.2.1 Express.js

È un framework open source per applicazioni web Node.js, molto utilizzato e ben documentato. E' utilizza per i seguenti motivi:

- Offre un sistema di *routing*$_G$ semplice, che consente di utilizzare *template engine*$_G$ per generare le pagine HTML;
- Permette di accedere in maniera agevole al corpo delle richieste e manipolare quindi i dati;
- Tramite i *middleware*$_G$ consente una efficace ed elegante gestione degli errori;
- Essendo ampiamente utilizzato, è disponibile una grande comunità di supporto online.

Link: http://expressjs.com/it/

4.2.2 express-validator

Questa libreria fornisce un framework per la validazione dei form inviati come request ad un middleware di express.
Link: https://express-validator.github.io/docs/

4.2.3 lda

Questa libreria fornisce una implementazione di un algoritmo di *LDA*$_G$ (Latent Dirichlet Allocation). Questo algoritmo è utilizzato per estrarre argomenti, detti topic, e le keyword correlate da una collezione di documenti testuali.
L'abbiamo utilizzato per estrapolare le parole di maggiore rilevanza all'interno delle frasi nel testo trascritto di una registrazione.
Link: https://www.npmjs.com/package/lda

4.2.4 gstore-node

È una libreria per la modellazione di *entità*$_G$ di Google Cloud Datastore per Node.js, che racchiude molte utilità per interfacciarsi con il servizio. Tra le funzionalità chiave permette di definire *modelli*$_G$ di entità, ciascuno con un proprio *schema*.
Ogni *schema di validazione*$_G$ contiene le proprietà valide per ogni entità di quel modello e i vincoli sui tipi e valori accettabili, in modo che le entità possano essere agevolmente validate quando necessario.
Link: https://www.npmjs.com/package/gstore-node

4.2.5 js-yaml

È un'implementazione di un *parser*$_G$ per il formato YAML. L'abbiamo utilizzata per operazioni di lettura e scrittura del file contenente le impostazioni generali del prodotto.
Link: https://www.npmjs.com/package/js-yaml

4.2.6 hbs

È un semplice *wrapper*$_G$ per il template engine *Handlebars.js*$_G$, che lo integra in Express.js. Handlebars.js permette di creare facilmente template semantici di pagine HTML, che vengono poi 'compilate' con i dati forniti dal backend.
Link: https://www.npmjs.com/package/hbs
Link: https://handlebarsjs.com/

4.2.7 multer

È un middleware per Express.js che gestisce i dati caricati da form di pagine web, primariamente utilizzato per il caricamento di file.
Nel nostro caso ne facciamo uso per gestire il caricamento sul server dell'audio della registrazione.
Link: https://www.npmjs.com/package/multer

4.2.8 winston

Questa libreria fornisce un *logger*$_G$ per Node.js. Lo utilizziamo per tenere traccia su file oppure su console di eventuali errori e per le attività di debugging.
Link: https://www.npmjs.com/package/winston

5 Architettura

In questa sezione descriviamo brevemente l'architettura dell'intero sistema.

Il sistema adotta l'utilizzo di Express.js e dei servizi Google Cloud Platform per realizzare un'architettura *stateless*$_{\text{G}}$ *client-server*$_{\text{G}}$, fornita pertanto di un front-end e di un back-end. Essa realizza due funzionalità in particolare:

- Si interfaccia con il database Google Cloud Datastore e visualizza i dati su una dashboard che fornisce tutti gli strumenti per gestire i progetti con i propri stand-up;
- Gestisce l'elaborazione delle registrazioni attraverso un'apposita pipeline che prevede l'utilizzo di un algoritmo di text mining, fornendo dei comandi per personalizzare il processo, come ad esempio l'aggiunta di parole chiave da parte dell'utente.

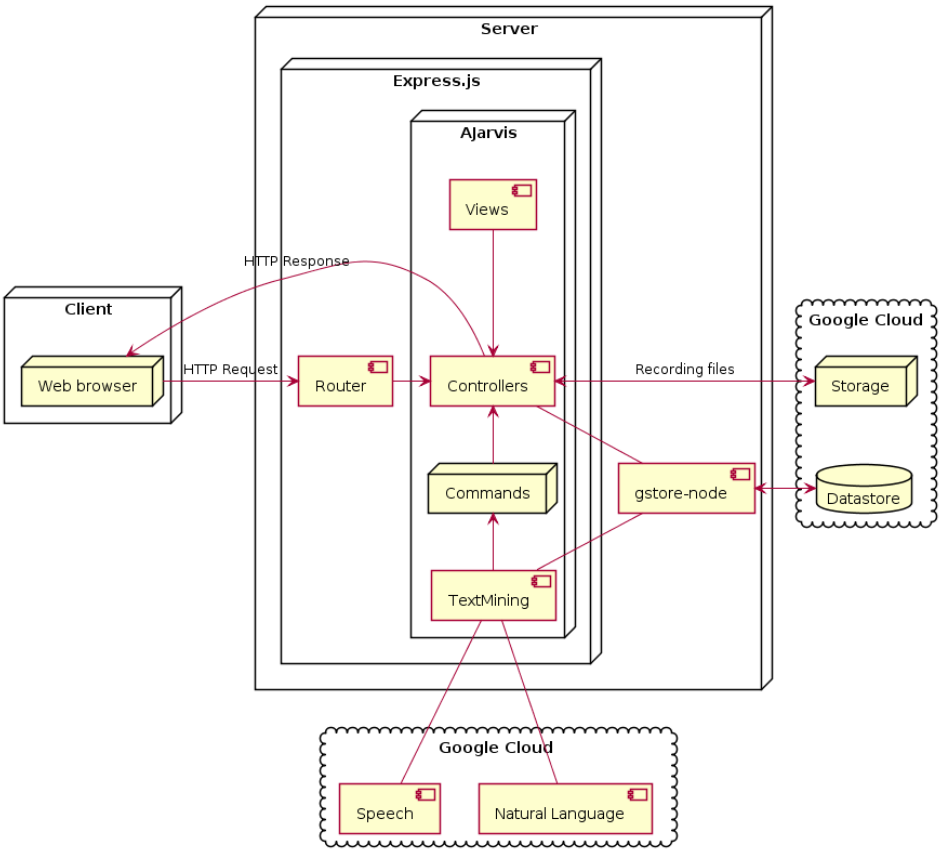

Figura 1: Schema generale dell'architettura di *AJarvis*

5.1 Front-end

Il *front-end*, garantisce all'utente la possibilità di accedere alle funzionalità del sistema quali la registrazione e la consultazione dei risultati delle elaborazioni delle registrazioni. Come tale, risiede interamente a lato *client*.

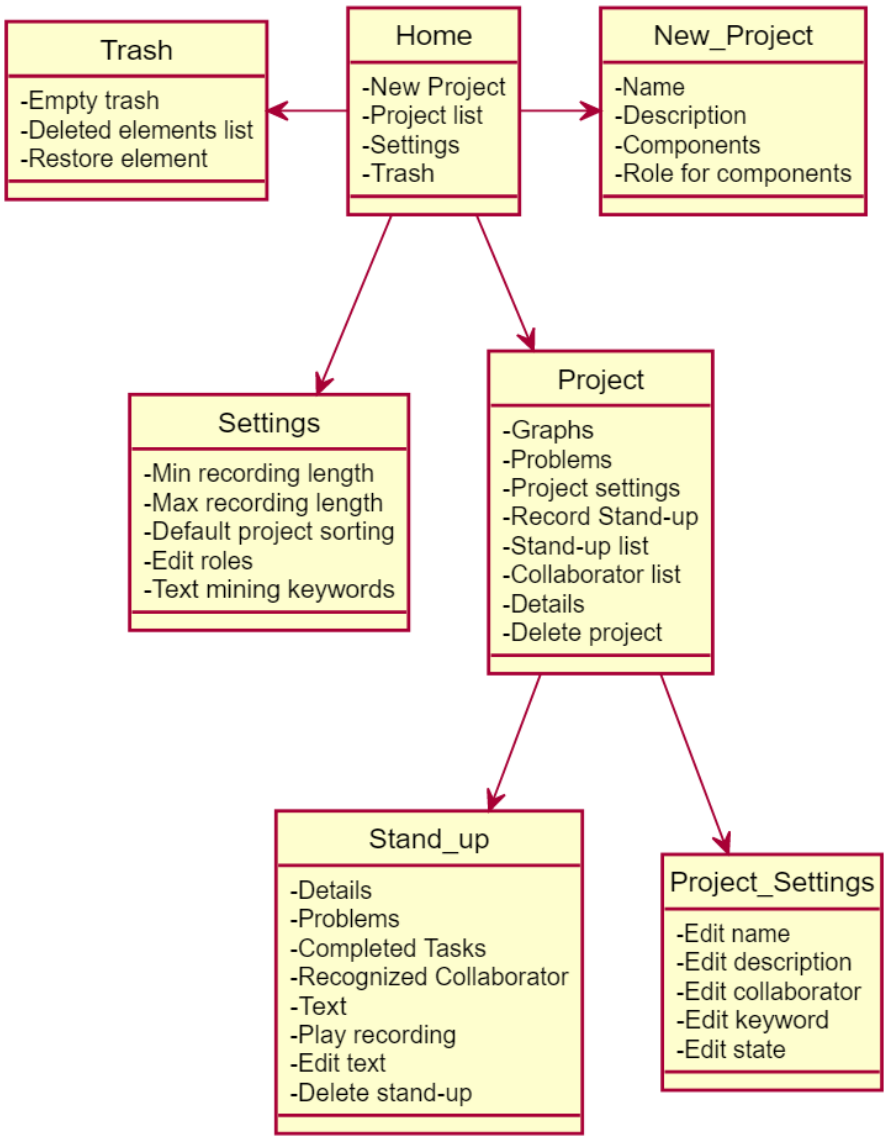

Figura 2: Schema delle pagine web previste su *AJarvis*

I "campi dati" delle pagine sono una rappresentazione concettuale di ciò che è possibile visualizzare e/o eseguire all'interno della specifica vista.
I collegamenti tra una vista e l'altra indicano i link disponibili all'interno delle pagine per navigare da una all'altra. Si è semplificata la rappresentazione rendendo accessibili

`Trash` e `New_Project` esclusivamente dalla `Home`, mentre tali link sono disponibili in tutte le pagine, così come un collegamento alla `Home` stessa.

5.2 Back-end

Il back-end risiede lato server, racchiude il modello dei dati, la logica di business e la logica applicativa propria del sistema e si estende fino ad includere le funzionalità di Google Cloud Platform accedute dall'applicazione che rappresentano l'archiviazione dei dati. Infine il back-end include le funzionalità proprie di text mining che verranno impiegate dall'apposito algoritmo.

5.2.1 Routing e controller

Il controller dell'applicazione è realizzato dal framework Express.js[1] che gestisce le richieste inviate dal browser.

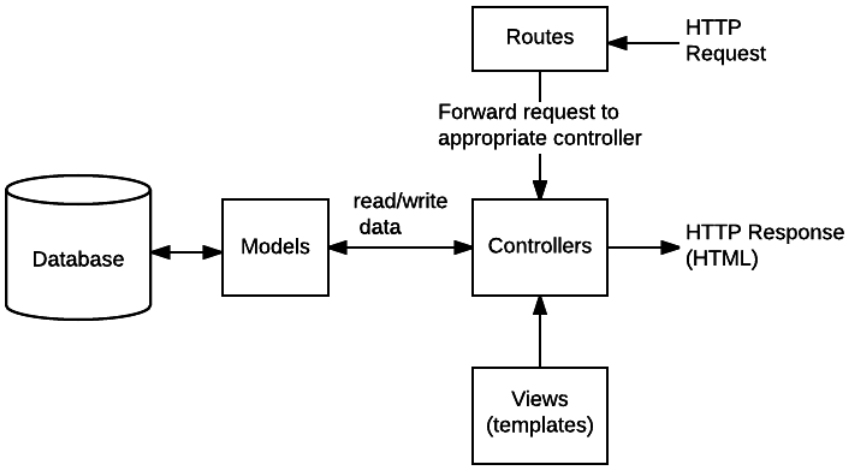

Figura 3: Schema di interazione fra View, Controller e Route

Come indicato dalla figura 3, route, view, controller interagiscono nel seguente modo:

- Le routes indirizzano le richieste dell'utente chiamando opportune funzioni middleware, ciascuna delle quali rappresenta un controller;
- I controller, eseguono operazioni sul modello, ottengono i dati che serve visualizzare e costruiscono le pagine HTML, (che rappresentano le view) da visualizzare nel browser;

[1] http://expressjs.com/it/

- Le view sono passive e sono i template HTML realizzati con il framework Handlebars.js, che vengono popolati dai controller e restituiti al client.

Descriviamo di seguito i percorsi di route utilizzati, con una breve descrizione. Queste route sono dichiarate nel file `routes/index.js`.

- `/`: il percorso di root redirige in `/projects`;
- `/projects`: visualizza la lista con tutti i progetti;
  - `/projects/new`: permette di creare un nuovo progetto, inserendo nome, descrizione, lista di collaboratori con rispettivi ruoli;
  - `/projects/trash`: visualizza la lista dei progetti eliminati, mostrando la data di eliminazione;
    - `/projects/trash/delete`: richiesta per cancellare definitivamente i progetti selezionati;
    - `/projects/trash/untrash`: richiesta per ripristinare i progetti selezionati, rimuovendoli dal cestino.
- `/settings`: visualizza la pagina di impostazioni di *AJarvis* (come form) e permette di modificarle, e salvarle con un apposito pulsante;
- `/standups`: visualizza la lista di tutti gli stand-up di tutti i progetti e permette di visualizzare lo stato di elaborazione, il progetto a cui appartengono ed alcuni dei risultati dell'elaborazione. Include inoltre la possibilità di filtrare gli stand-up per data;
- `/project/:projectId`: mostra la pagina del progetto che ha l'id uguale a *projectId*;
  - `/project/:projectId/recordings`: mostra l'elenco di tutte le registrazioni del progetto che ha l'id uguale a *projectId*;
  - `/project/:projectId/delete`: sposta nel cestino il progetto corrente;
  - `/project/:projectId/settings`: visualizza il form con le impostazioni del progetto specifico e permette di modificarne i campi: nome, collaboratori, keyword significative, stato e descrizione;
  - `/project/:projectId/trash`: visualizza l'elenco di tutti gli stand-up eliminati di un certo progetto;
    - `/projects/:projectId/trash/delete`: cancella definitivamente gli stand-up selezionati;
    - `/projects/:projectId/trash/untrash`: ripristina gli stand-up selezionati;
  - `/project/:projectId/recording/new`: mostra la pagina per la registrazione di un nuovo stand-up, gestendone anche il salvataggio;

- `/project/:projectId/recording/:recordingId`: recupera i dettagli di un recording;
  - `/project/:projectId/recording/:recordingId/redoUpload`: accoda il comando per rieseguire il caricamento del file audio della registrazione su Google Cloud Storage;
  - `/project/:projectId/recording/:recordingId/redoTranscription`: accoda il comando per rieseguire la traduzione dell'audio in testo con Google Cloud Speech API;
  - `/project/:projectId/recording/:recordingId/redoTextMining`: avvia il comando per rieseguire l'algoritmo di text-mining;
  - `/project/:projectId/recording/:recordingId/editTranscription`: permette di modificare il testo trascritto di uno stand-up;
  - `/project/:projectId/recording/:recordingId/delete`: sposta lo stand-up corrente nel cestino del progetto;

5.2.2 Modello dei dati

I dati dei progetti e delle registrazioni sono rappresentati da entità salvate su un database remoto, Google Cloud Datastore affiancato dalla libreria *gstore-node*.

Le entità individuate sono quindi rappresentate da istanze di `Model` del gstore-node definendo uno *schema*. Uno schema struttura le varie proprietà dell'entità, il tipo e una procedura per validarle.

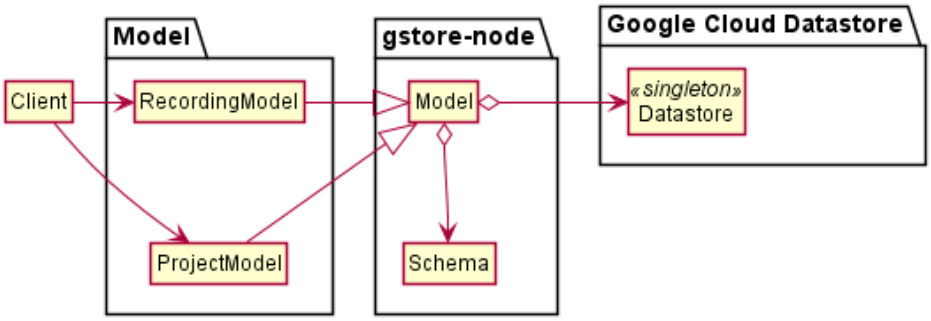

Figura 4: Diagramma generale delle classi e componenti

5.2.2.1 Modelli utilizzati

Entrambi i modelli di seguito descritti sono definiti nella cartella `lib/model/`.

5.2.2.1.1 ProjectModel

`ProjectModel` descrive un progetto di cui verranno registrati i *daily stand-up*$_G$. Esso sarà caratterizzato dal nome, una breve descrizione, lo stato (aperto o chiuso) del progetto.

Conterrà inoltre una lista dei nomi completi dei collaboratori ed una lista di parole chiave, specifiche del progetto, che verranno utilizzate per migliorare il riconoscimento delle parole nelle registrazioni degli stand-up. Infine è presente la data della creazione del progetto.

5.2.2.1.2 RecordingModel

`RecordingModel` rappresenta una registrazione di un daily stand-up di un progetto, per questo motivo conterrà l'id di riferimento dal progetto.

Fra i campi dati avremo il nome del file della registrazione, la data di creazione e lo stato di avanzamento del processo di text mining ed infine l'output, sempre del text mining. Le due entità sono rappresentate in figura 5.

Verrà conservata inoltre la trascrizione dal file audio a testo per verificare la corretta comprensione dello stand-up e l'eventuale modifica manuale del testo.

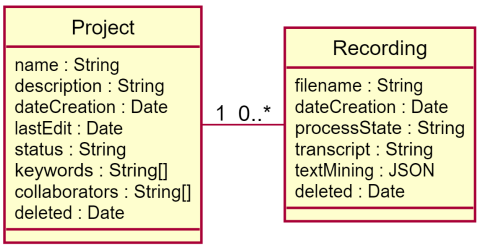

Figura 5: Diagramma delle entità e le loro relazioni

5.2.2.2 Impostazioni dell'applicazione

L'applicazione necessita di un file di configurazione che contenga le impostazioni generali del prodotto. Queste sono:

- Durata minima e massima di una registrazione: nel caso una registrazione abbia durata non conforme, essa sarà scartata automaticamente;
- Lista di ruoli di progetto: l'elenco dei ruoli che è possibile assegnare ad ogni progetto (ad esempio Amministratore, Programmatore, ecc.);
- Ordine di visualizzazione dei progetti nella pagina principale.

Queste impostazioni sono mantenute in un file YAML locale, letto e scritto attraverso la libreria *js-yaml*[2].

[2] https://www.npmjs.com/package/js-yaml

5.2.3 Logica del processo a pipeline

Il processo di elaborazione dell'audio di una registrazione proveniente dal front-end è implementato come una pipeline. Ogni passo della $pipeline_G$ espone una interfaccia e si relaziona con il modello dei dati e con servizi di Google Cloud Platform tramite specifiche interfacce. La figura 6 mostra nel dettaglio le classi che collaborano in questa sequenza di passi.

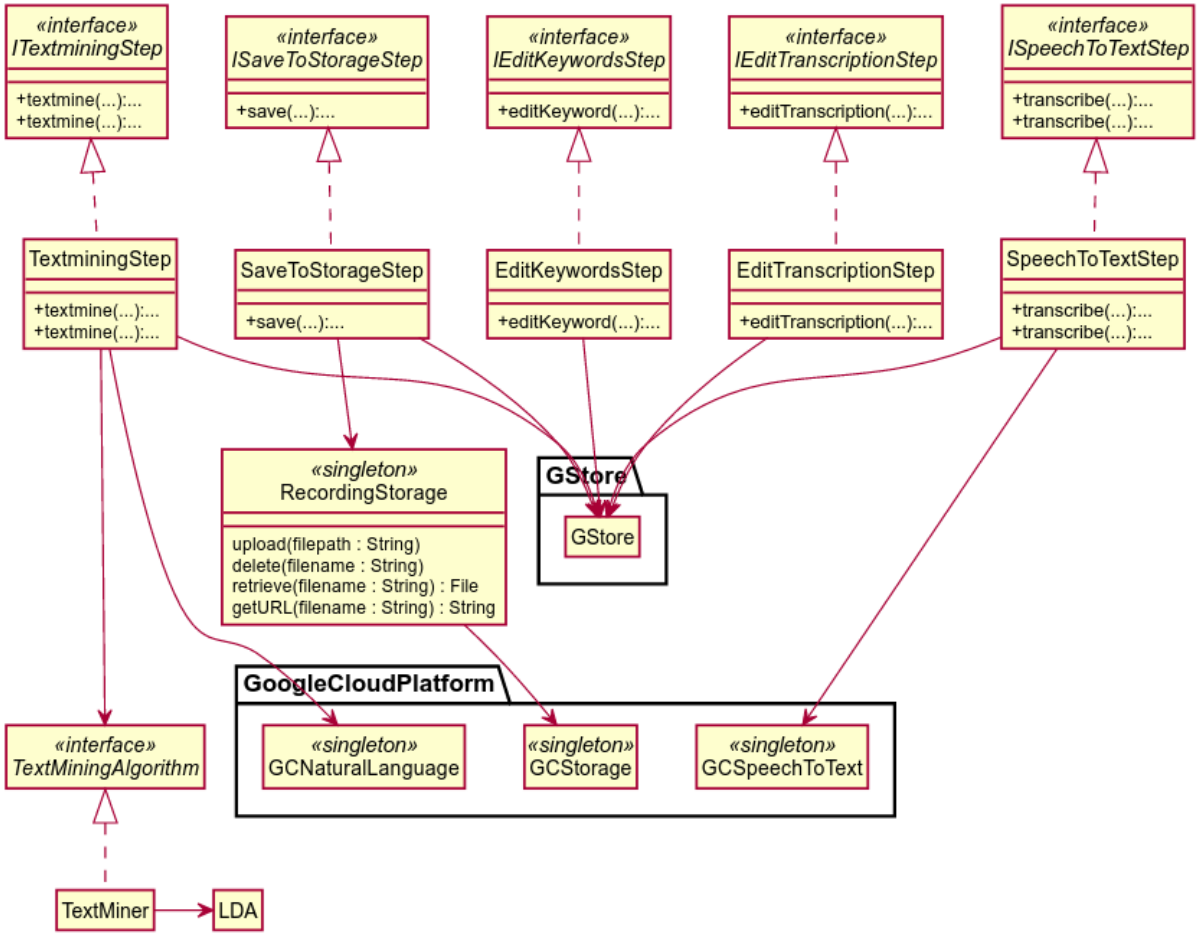

Figura 6: Dettaglio delle funzionalità della pipeline

I passi sono poi richiamati nel loro corretto ordine mediante diverse implementazioni di comandi (in figura 7), che vengono invocati dalle relative route su richieste provenienti dal client. A titolo esemplificativo, riportiamo in figura 8 il funzionamento della pipeline a partire dallo stato `SaveToStorageStep`.

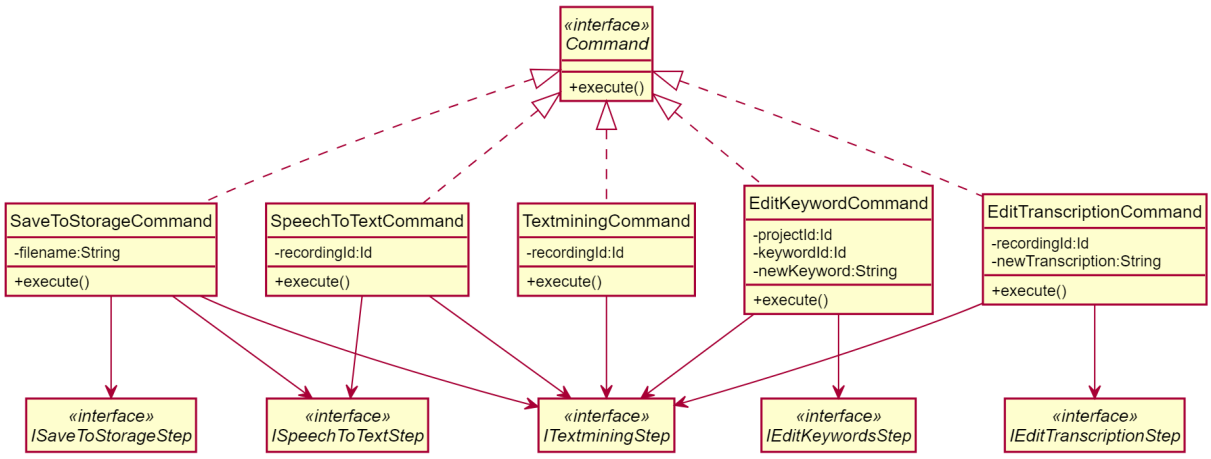

Figura 7: Gerarchia dei comandi che eseguono operazioni sugli *step*

Al termine di ogni step viene aggiornato e memorizzato su Google Cloud Datastore il modello della registrazione in via di elaborazione (`RecordingModel`) contenente i risultati parziali dell'ultimo step eseguito. In questo modo risulta possibile ripristinare l'esecuzione, semplicemente fornendo l'id della Registrazione al comando che si occupa di avviare la pipeline, il quale provvederà a caricare dal Datastore i dati sui quali effettuare l'elaborazione.

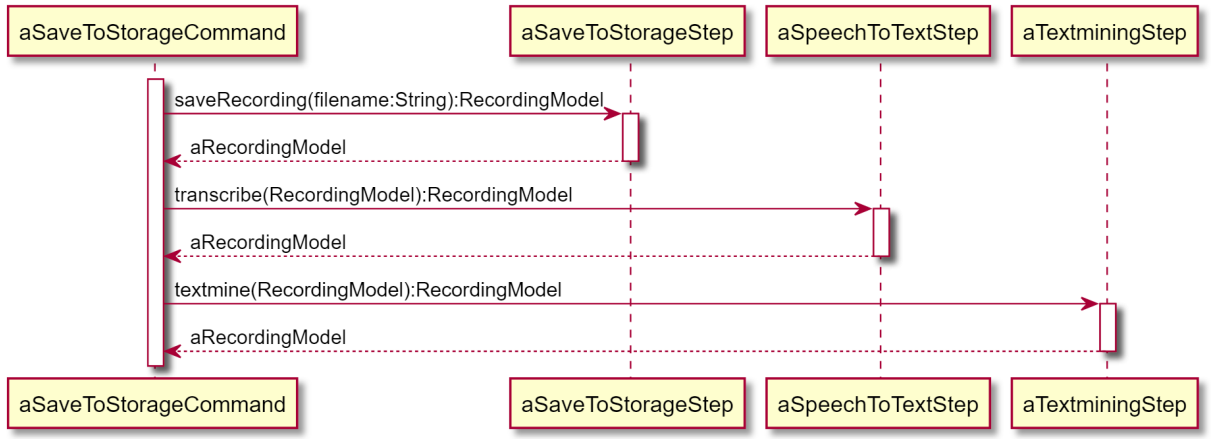

Figura 8: Sequenza del comando `SaveToStorageCommand`

Le funzionalità proprie dei passi della pipeline di processazione delle registrazione sono rappresentate, in ordine di esecuzione, dalle classi:

1. `SaveToStorageStep`: che si occupa di salvare il file audio della registrazione proveniente dal client, sul servizio Google Cloud Storage;

2. `SpeechToTextStep`: che si occupa invece di eseguire la trascrizione dell'audio caricato dal passo precedente, in testo;

3. `TextminingStep`: che si occupa infine di applicare al testo estrapolato al passo precedente, l'analisi effettuata tramite Google Natural Language e successivamente l'algoritmo di text mining vero e proprio.

5.2.4 Interazione tra i componenti

`StandupRoute` si occupa di invocare, attraverso `CommandQueue` e su richiesta del client, i comandi relativi all'avvio della pipeline e relativi agli accessi ed alle modifiche da effettuare agli elementi del modello. `CommandQueue` rappresenta una coda di comandi di tipo `Command`, con politica di ordinamento $FIFO_G$. L'obiettivo della classe è quello di accodare i comandi invocati in maniera asincrona da una `Route` e di garantire il corretto ordinamento della loro esecuzione. Per fare ciò la classe gestisce un flusso di esecuzione separato che si occupa di osservare la coda ed eseguire i comandi qualora non fosse vuota.

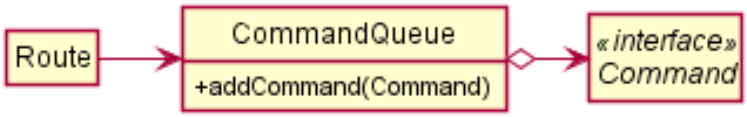

Figura 9: `CommandQueue`

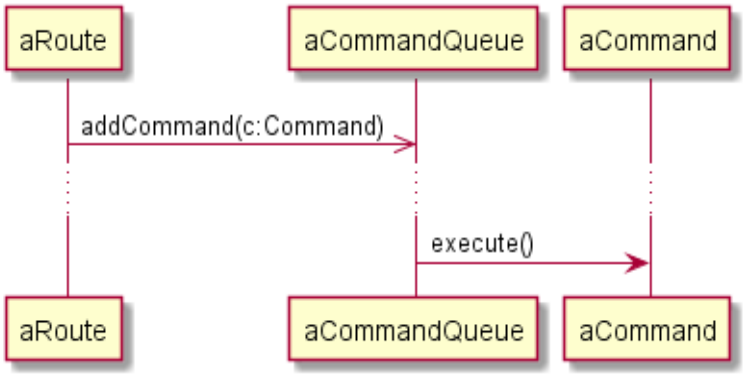

Figura 10: Sequenza di `CommandQueue`

5.2.5 Gestione degli errori

Abbiamo scelto di mantenere la logica di gestione delle eccezioni ed errori il più semplice possibile, anche considerando che molti errori potrebbero dipendere da Google Cloud Platform. Qualsiasi errore non gestito conduce ad un pagina di errore che riporta il messaggio d'errore e permette di tornare alla *Home* e se possibile alla pagina precedente. Il template della pagina di errore si trova in `webserver/views/error.hbs`.

Inoltre eventuali errori durante l'esecuzione di *AJarvis* possono essere visualizzati all'interno dei log nella cartella `webserver/logs`. Ogni log, prodotto con l'ausilio del logger *Winston*, è identificato univocamente dalla data a cui si riferisce.

5.2.6 Algoritmo Text-mining

Figura 11: Diagramma di attività ad alto livello per l'algoritmo di text mining

5.2.6.1 Idea generale

L'algoritmo di text mining ha l'obiettivo di raccogliere tutte le informazioni importanti rilevate durante la cerimonia agile, sia positive che negative. La classe che lo modella riceve in input il testo dello stand-up, ricondotto a lemmi e ripulito dalle parole prive di significato, i tag grammaticali associati a ciascuna parola e i nomi dei partecipanti al progetto. L'output del processo è la categorizzazione delle frasi, tramite la determinazione

della tipologia, l'assegnazione di un punteggio rappresentativo della rilevanza nel discorso e l'identificazione del membro del gruppo che l'ha pronunciata.

6 Estensione delle funzionalità

6.1 Modifica algoritmo text mining

6.1.1 Aggiunta Keywords

Per il corretto funzionamento l'algoritmo, contenuto nella classe `TextMiner.js` si basa su 5 collezioni di parole:

- `problemidentifiers`: Comprende le parole indicanti una frase che spiega un problema rilevato;
- `problemicompletetiondentifiers`: Comprende le parole indicanti una frase che conferma la risoluzione di un problema;
- `taskcompletetionidentifiers`: Comprende le parole indicenti una frase che esprime il completamento di un task;
- `powerwords`: Comprende le parole accrescono il significato della parola successiva;
- `stopwords`: Comprende le parole da rimuovere dal discorso perché ininfluenti(es. articoli).

Per ampliare il vocabolario dell'algoritmo è sufficiente aggiungere i termini ritenuti utili all'array corrispondente, scrivendoli direttamente nel file.

6.1.2 Modifica parametri di rilevazione

Sono definite alcune costanti arbitrarie per l'algoritmo:

- `MAXSCORE`: Punteggio massimo da assegnare a ciascuna frase;
- `MINIMUM_SIMILARITY`: Indice di similarità minimo per dichiarare la somiglianza di due frasi e considerare una la continuazione dell'altra;
- `MINIMUM_EQUALITY`: Indice minimo di somiglianza per dichiarare due frasi come uguali;

Se si volesse operare per aumentare o diminuire la precisione dell'algoritmo basta modificare queste costanti.

6.2 Sostituzione algoritmo text mining

Per aggiungere un diverso algoritmo di text-mining è sufficiente creare una nuova classe concreta che implementi `TextMiningAlgorithm`, e la cui funzione `textMining` che ritorni un elemento di tipo $JSON_{\mathrm{G}}$ contenete i risultati dell'elaborazione.

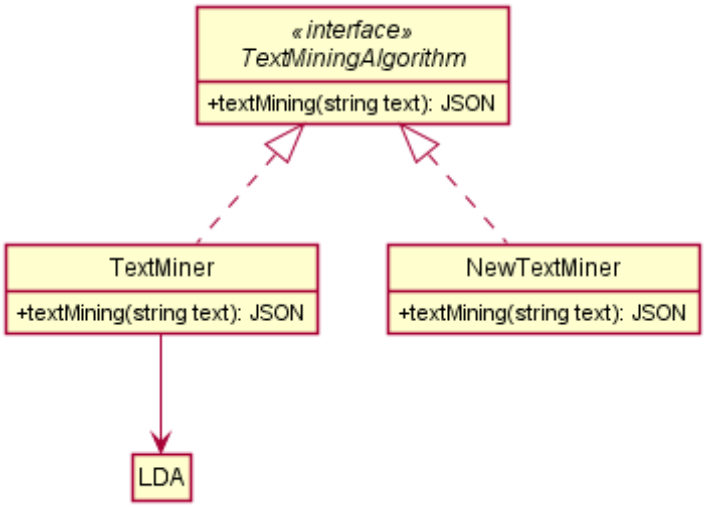

Figura 12: Aggiunta nuovo algoritmo text mining

6.3 Modifica di uno Step esistente

Per modificare un singolo `Step` della pipeline bisogna aggiungere una classe concreta che implementi l'interfaccia corrispondente allo `Step` in questione (es. per aggiungere una nuova versione di `SaveToStorageStep` è necessario creare una classe che implementi `ISaveToStorageStep`). Deve essere rispettato il vincolo che l'output dello `Step` aggiunto sia fornito e sia interpretabile dallo lo Step successivo in ordine logico.

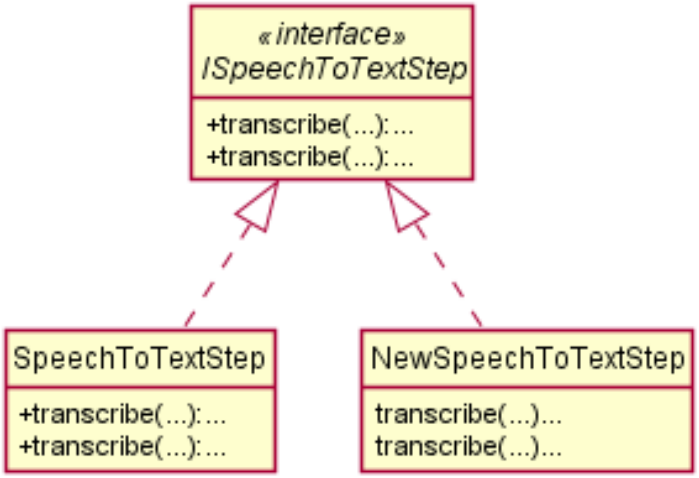

Figura 13: Aggiunta nuova versione di uno `Step` esistente

6.4 Aggiunta di un nuovo Step

Per aggiungere uno `Step` bisogna creare l'interfaccia corrispondente, in modo da mantenere l'estendibilità dello `Step`. Se lo `Step` si integra con almeno un altro `Step` è necessario porre attenzione all'output dello `Step` aggiunto, che deve essere compatibile con ciò che lo `Step` successivo richiede.

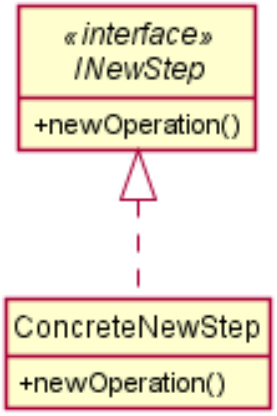

Figura 14: Aggiunta di un nuovo `Step`

6.5 Aggiunta di un Comando

Per aggiungere un nuovo comando bisogna creare una classe concreta che implementi l'interfaccia `Command`. La funzione `execute()` dovrà richiamare in ordine gli `Step` necessari alla risoluzione del comando.

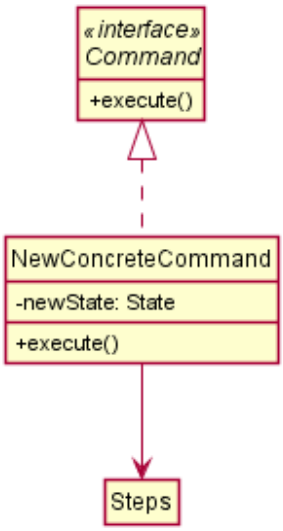

Figura 15: Aggiunta di un nuovo comando

6.6 Estensione dei modelli

Nel caso si desideri aggiungere nuove proprietà ai modelli si può procedere nel seguente modo:

1. Aggiungere le proprietà desiderate allo *schema* del modello da estendere, specificandone il tipo;
2. Opzionalmente si possono inserire delle funzioni di validazione seguendo quanto indicato nella documentazione di *gstore-node*.

Aggiunte di proprietà alle entità non inficieranno sulle entità già presenti nel datastore e verranno aggiunte automaticamente al salvataggio ove mancassero.

6.7 Modifica dell'interfaccia web

Per modificare l'interfaccia in seguito all'aggiunta di nuove funzionalità è necessario seguire i seguenti due passi:

6.7.1 Modifica della view

- **Modifica di una pagina**: modifica del template corrispondente alla pagina presente nella cartella `views/` all'interno di `webserver/`;
- **Aggiunta di una pagina**: aggiunta del template corrispondente alla pagina nella cartella `views/` in formato `.hbs`.

6.7.2 Aggiunta di una route

Definire le routes corrispondenti al template aggiunto o modificato per poter creare la pagina ed inviarla correttamente al client su richiesto. Le routes vanno inserite in un file `.js` nella cartella `routes/controllers`.

6.7.3 Creazione di un *helper*

Nell'utilizzare Handlebars.js torna utile definire delle funzioni chiamate $helpers_{\mathrm{G}}$ che servono ad effettuare operazioni sui dati passati al template engine. Esistono degli helper già presenti di default, i quali permettono di iterare su collezioni di dati, oppure agire in modo condizionale, ecc. Nella cartella `webserver/views/helpers` sono stati definiti helper personalizzati al fine di formattare correttamente i dati da inserire nella pagina HTML generata.
Ogni helper deve essere registrato per poter essere usato nel template hbs. Per aggiungere degli helper procedere nel seguente modo:

1. Aggiungere una nuova funzione ai file esistenti oppure in un nuovo file nella cartella `webserver/views/helpers`;

2. Se si è aggiunto un nuovo file esso deve essere registrato dal file `webserver/views/helpers/index.js`; è sufficiente importarlo con `require` e inserire il nome dell'oggetto importato nell'array `all`.

6.7.4 Creazione di un *partial*

Con hbs è possibile racchiudere blocchi di template in file chiamati *partial*, per facilitarne il riuso. È utile definire dei *partial* quando ci sono elementi che si ripetono in più parti dell'applicazione, in modo da non ripetere interamente il codice nei file sorgente.
È inoltre possibile definire variabili all'interno del file *partial*, in modo da renderlo dinamico. Al momento dell'istanziazione del template, le variabili saranno sostituite dai parametri passati con la chiamata.
Per definire ed utilizzare un nuovo *partial* è sufficiente:

1. Aggiungere un file `.hbs` nella cartella `webserver/views/partials`. Il contenuto del file sarà un normale template hbs, comprese le eventuali variabili desiderate;

2. Effettuare la chiamata all'interno del template hbs desiderato, eventualmente definendo le variabili necessarie per il suo corretto funzionamento. Tutte le variabili non definite saranno passate come nulle.

Una chiamata ad un *partial* è fatta tramite:

```
{{> myPartial myVariable='variableValue'}}
```

dove `myPartial` è il nome del file `.hbs` salvato sulla cartella precedentemente specificata.

Per la documentazione completa e precisa, si rimanda al sito ufficiale:

http://handlebarsjs.com/partials.html

A Glossario

B

Bucket
I bucket sono i contenitori utilizzati in Google Cloud Storage. Tutto quello che è salvato su Storage deve essere contenuto in un bucket, che può essere visto come una cartella, con la differenza che non è possibile annidare bucket.

C

Callback Una funzione, o un "blocco di codice" che viene passata come parametro ad un'altra funzione. In particolare, quando ci si riferisce alla callback richiamata da una funzione, la callback viene passata come argomento ad un parametro della funzione chiamante. In questo modo la chiamante può realizzare un compito specifico (quello svolto dalla callback) che non è, molto spesso, noto al momento della scrittura del codice.

Cloud Si intende la piattaforma di servizi web che viene utilizzata per l'elaborazione ed il salvataggio dei dati. Nel nostro caso si tratta di Google Cloud Platform.

Client-server Un'architettura *client-server* indica la presenza di un server che gestisce le richieste inviate da un client. Si contrappone all'architettura *serverless*, nella quale il server è sostituito da un network distribuito.

F

FIFO Sigla per First-In-First-Out. Indica una politica di gestione di un processo che dà priorità all'elaborazione degli elementi in base all'ordine di inserimento.

H

Handlebars.js Template engine funzionale e semplice da utilizzare per creare pagine web lato server.

Helper Un helper è una funzione che permette di manipolare dati e collezioni che può essere invocata direttamente dal template Handlebars.

J

JSON Acronimo di JavaScript Object Notation, è un formato adatto all'interscambio di dati fra applicazioni client/server.
L

LDA Modello statistico che permette l'estrapolazione di argomenti da un testo complesso.

Logger Strumento che permette la scrittura su file di messaggi (ad es. di errore).

M

Middleware Insieme di software che fungono da intermediari fra strutture e programmi informatici, permettendo loro di comunicare a dispetto della diversità dei protocolli o dei sistemi operativi.

Modello (di entità) Rappresenta l'insieme delle caratteristiche possedute da un entità in un database che la definiscono.

P

Parser Programma che analizza un file, verificandone la correttezza sintattica rispetto a una data grammatica; è utilizzato dai compilatori di un linguaggio di programmazione e, in linguistica, per analizzare testi.

Partial Si tratta di particolari template scritti con *Handlebars.js* che possono essere chiamati da altri template. Sono particolarmente utili per promuovere il riuso del codice in Handlebars.js.

Pipeline Sequenza di comandi da eseguire uno di seguito all'altro ordinatamente.

R

Routing (Express) Per Routing si intende determinare come un'applicazione risponde a una richiesta client a un un URI (o percorso) e un metodo di richiesta HTTP specifico (GET, POST ad esempio). Ciascuna route può disporre di una o più funzioni, le quali vengono eseguite quando si trova una corrispondenza per la route.

S

Schema di validazione Uno schema rappresenta la forma che ogni entità di un certo modello dovrebbe avere. Contiene le proprietà valide per ogni entità di quel modello e i vincoli sui tipi e valori accettabili

Stateless Per architettura stateless si intende un network in cui il server non tiene traccia dello stato di computazione dei dati. Questa informazione viene salvata su un database per garantire la massima consistenza anche in caso di crash o arresto prima del completamento della sequenza di operazioni.

***Stand-up* giornaliera** Incontri giornalieri tra membri dello stesso progetto, di breve durata, in cui si discutono avanzamenti e problemi riscontrati durante il giorno precedente e piani per il lavoro nel giorno stesso.

T

Template Engine Un sistema di template è utilizzato nella programmazione web per permettere agli sviluppatori la creazione automatica di pagine personalizzate con contenuti, ad esempio in seguito ad una ricerca. Questo permette il riuso degli elementi statici delle pagine web, garantendo la creazione di elementi dinamici in base ai parametri della richiesta.

Text mining È una forma particolare di data mining nella quale i dati consistono in testi in lingua naturale, in altre parole, documenti "destrutturati". Il text mining unisce la tecnologia della lingua con gli algoritmi del data mining. L'obiettivo è l'estrazione di informazione implicita contenuta in un insieme di documenti.

W

Wrapper Modulo software che ne "riveste" un altro, ovvero che funziona da tramite fra i propri clienti (che usano l'interfaccia del wrapper) e il modulo rivestito (che svolge effettivamente i servizi richiesti, su delega dell'oggetto wrapper).

Y

YAML Formato per la serializzazione di dati creato per essere facilmente leggibile da esseri umani.