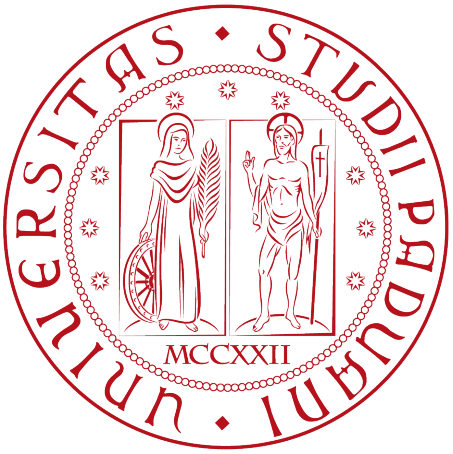
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Sviluppo del frontend di un'applicazione web in React js

Tesi di laurea

Relatore

Prof.Paolo Baldan

Laureando

Mattia Episcopo

Anno Accademico 2022-2023

If a machine is expected to be infallible, it cannot also be intelligent.

Alan Turing

Sommario

Il presente documento descrive l'esperienza di stage del laureando Mattia Episcopo presso l'azienda SAI (Synthema Artificial Inteligence) della durata di circa 300 ore. L'obiettivo dello stage era la realizzazione del frontend di un'applicazione per la gestione di particolari tracce audio e relativi metadati riguardanti le registrazioni dei procedimenti delle aule di tribunale. Gli obiettivi dello stage erano diversi ma lo scopo finale era la realizzazione della parte frontend con l'utilizzo del framework javascript React.

"I'm not crazy, my mother had me tested"

— Dott. Sheldon Lee Cooper

Ringraziamenti

In primis, ringrazio il prof. Paolo Baldan, relatore della mia tesi, per la disponibilità e l'aiuto fornitomi nella stesura di questo elaborato.

Ringrazio l'azienda ospitante, Synthema Artificial Intelligence, e in particolare il tutor interno, Giulio Paci che mi ha guidato in questa esperienza.

Padova, Febbraio 2023 Mattia Episcopo

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'azienda

Figura 1.1: Loso Syntema Artificial Intelligence

Synthema Artificial Intelligence (S.AI) è una start-up[g]innovativa che si occupa di ricerca, progettazione, sviluppo, commercializzazione e manutenzione di prodotti e servizi innovativi ad alto valore tecnologico, basati su tecniche di Intelligenza Artificiale (in particolare reti neurali profonde), per l'analisi integrata e la comprensione di dati multimodali da fonti eterogenee (linguaggio naturale sia scritto che parlato, audio, immagini, video, dati generati da sensori) e per la gestione dei flussi di lavoro, sia nel settore pubblico che privato.

1.2 Introduzione al progetto

Il progetto nasce dalla necessità dell'azienda di creare un sistema semi-automatico per la creazione di particolari ticket. Il sistema è rivolto ai lavoratori della aule di tribunale, ovvero i fonici, che si occupano di registrare su dei supporti CD quello che avviene nell'aula di un tribunale. Dopo aver registrato la loro parte di giornata nel tribunale di competenza, devono inserire tutti i dati relativi ai processi a cui hanno assistito, nel sistema di memorizzazione della loro azienda. Le registrazioni audio dei fonici si articolano in una traccia per ogni microfono presente in aula più una traccia mixer che li riassume tutti. La registrazione di queste tracce audio viene impostata ad inizio giornata, facendo corrispondere ad ogni microfono una traccia, per poi non essere più modificata. Oltre a questo ogni fonico, con il sistema di registrazione presente in aula, può prendere degli appunti in riferimento a quello che succede, magari per alcuni casi particolari che si possono verificare o per ricordarsi i procedimenti a cui ha assistito. Tutti questi metadati[g]vengono salvati sottoforma di file di testo, questo file riporta tutto quello che è successo durante la registrazione in ordine cronologico,

qual'è il microfono attivo e quindi chi sta parlando, se ci sono appunti particolari e così via. Il lavoro del fonico si conclude poi con il caricamento di tutte le informazioni raccolte in aula sull'apposito sistema aziendale, con l'obiettivo di tenere traccia di tutti i procedimenti che avvengono in aula. Questi dati vengono creati come ticket, ogni ticket si compone di un procedimento, che riporta un proprio codice e deve contenere varie informazioni (es: chi è o chi sono i giudici, imputati, PM). Questo processo di creazione e caricamento di ticket al momento è molto laborioso per i fonici che devono sostanzialmente fare tutto a mano, l'obiettivo di questo progetto e del relativo prodotto finale è quello di automatizzare questo processo, in modo che il fonico solo caricando l'intero CD contenente le tracce audio e il file di testo che le riassume visualizzi i vari ticket dei procedimenti a cui ha assistito. Così facendo il nostro prodotto garantirebbe una notevole diminuzione del tempo di lavoro, una minor percentuale di errori e lascerebbe all'incaricato solamente l'onere di ricontrollare i dati relativi ai ticket automaticamente creati prima di caricarli sul sistema.

1.3 Principali problematiche

Per questo progetto sorgono diverse problematiche che elenco di seguito:

* **conoscenza dell'argomento**: le tematiche riguardanti i processi, le udienze e i procedimenti, risultano abbastanza vaghi allo start del progetto;
* **varietà dei metadati**: i metadati generati dal sistema, che coinvolgono lo schema delle tracce e gli appunti del fonico, possono generare grandi varietà di casi e sottocasi;
* **grandezza dei dati**: la mole dei dati da trattare in termini di byte[g], può raggiungere ordini di grandezza molto elevati, basti pensare che ogni ticket può comprendere svariate registrazioni e ognuna di esse può avere una durata molto lunga;

1.4 Soluzione scelta

Il progetto è stato analizzato attentamente per scegliere la soluzione migliore, che sia allo stesso tempo veloce nel caricamento dei dati e nella creazione dei ticket, ma che non perda informazioni importanti, visto la sensibilità dei dati trattati. Per questo progetto è stato scelto di creare un'applicazione inizialmente web ma con la possibilità di renderla desktop in futuro. L'applicazione sarà formata da un frontend[g]che avrà la principale funzione di gestire le informazioni necessarie a creare un ticket, e un backend[g]che servirà a tenerne traccia. In questo modo ogni volta che un CD di registrazioni viene caricato, non viene direttamente mandato al backend, ma rimane in gestione al frontend che ne elabora i dati e li gestisce seguendo le preferenze dell'utente, prima di passare solo i dati necessari al backend. Per la memorizzazione dei file invece la strada pensata in fase di analisi è l'utilizzo di un object storage[g]esterno dove fare l'upload solo delle tracce audio necessarie. L'oggetto del progetto di stage è proprio la parte di frontend di questo sistema, pensato per agevolare il lavoro dei collaboratori delle aule di tribunale. Riassumendo, le soluzioni scelte a fronte delle principali problematiche riscontrate sono:

* **conoscenza dell'argomento**: abbondante fase di studio iniziale e documentazione sulle tematiche poco note;

- **varietà metadati**:
  - libreria[g]**apposita**: creazione di una libreria interna apposita, con la funzione di analizzare e interpretare i metadati, in modo da comprenderne la maggior parte;
  - **editing metadati**: possibilità di editare i metadati dopo che sono stati interpretati dalla libreria e visualizzati nella nostra applicazione, per permettere ai fonici di modificarli in fase di creazione del ticket;
- **grandezza dei dati**:
  - **gestione nel frontend**: i dati caricati vengono gestiti direttamente nel frontend, per poi passare al backend solo quelli realmente necessari alla creazione dei ticket;
  - **object storage**: upload delle sole tracce audio che sono necessarie direttamente su un apposito object storage senza dover passarle al backend;

1.5 Descrizione del prodotto ottenuto

Il prodotto finale ottenuto si discosta leggermente da quello inizialmente pensato, infatti è stata soltanto abbozzata la costruzione del ticket vero e proprio per motivi legati alle tempistiche dello stage e per dare la precedenza agli obiettivi obbligatori e desiderabili dello stage stesso. L'applicazione finale risulta un buon punto di partenza per costruire la restante parte del prodotto. Il prodotto ottenuto ha come inizio l'autenticazione dell'utente con la maschera di login, in modo che l'accesso a tutte le operazioni sia consentito solo agli utenti autorizzati. L'applicazione nello stato finale presenta le seguenti funzionalità:

- **caricamento dei dati**: l'utente può caricare i dati sull'applicazione scegliendo se caricare interi CD di registrazioni o singoli file, questa seconda funzionalità risulta molto utile per gestire manualmente eventuali mancanze.
- **visualizzazione dati**: l'utente ha a disposizione la visualizzazione dei dati caricati in due versioni. Uno è un completo elenco dei file con i relativi metadati associati, l'altro è un elenco dei procedimenti che il sistema ha rilevato interpretando i metadati.
- **compilazione automatica** form[g]: i form con i codici dei procedimenti, risultano automaticamente compilati, dopo che la libreria ha interpretato i metadati in fase di caricamento delle registrazioni.
- **riproduzione audio dei file**: nella pagina di visualizzazione dei file l'utente ha la possibilità di riprodurre ogni traccia audio caricata.
- **visulizzazione metadati del file**: grazie all'interpretazione dei metadati, per ogni file caricato è possibile vedere tutte le informazioni che ha a disposizione, nello specifico, l'utente, oltre a trovare tutti i procedimenti rilevati dalla libreria, può visualizzare tutti gli interventi e le annotazioni che sono state rilevate.
- **caricamento procedimenti**: dopo la revisione dei dati caricati, i procedimenti ritenuti validi possono essere facilmente caricati sul backend, insieme all'upload dei relativi file sull'object storage.

* **riepilogo dati salvati**: l'utente può visulizzare la lista dei procedimenti dei quali è già stato fatto l'upload e andare nel dettaglio del singolo procedimento per visualizzare le informazioni complete.

1.6 Strumenti utilizzati

L'azienda usa un'infrastruttura già collaudata della quale fanno parte sistemi di versionamento del codice, strumenti per l'organizzazione del lavoro e per la comunicazione all'interno del gruppo di lavoro. Di seguito una panoramica dettagliata degli strumenti utilizzati per lavorare nel team di sviluppatori dell'azienda.

Slack

Figura 1.2: Logo Slack

E lo strumento che viene usato per la collaborazione aziendale, permette di vedere lo stato degli utenti (sviluppatori dell'azienda nel nostro caso) se disponibili o assenti al momento. Questa applicazione da la possibilità di comunicare singolarmente con gli altri partecipanti oppure in piccoli gruppi divisi per progetto. Permette in oltre la condivisione dei file, molto utile nelle comunicazioni veloci.

Jira

Figura 1.3: Logo jira

Jira[1] è lo strumento che l'azienda usa per l'organizzazione del lavoro. Permette di creare vari progetti, e per ogni progetto consente la creazione dei relativi task[g]da svolgere. In base alla metodologia di lavoro che si sceglie di utilizzare lo strumento permette di adottarla in tutto. Nel nostro caso la scelta aziendale è il metodo SCRUM e questo strumento consente la creazione degli sprint, è fornito di un backlog, ha una bacheca personalizzata per ogni utente che consente di vedere lo stato di avanzamento dei task di interesse. Inoltre consente di creare report sull'andamento degli sprint oppure su alcuni periodi, cosa molto utile per tracciare la linea guida nel miglioramento aziendale.

[1] *Jira*. URL: https://www.atlassian.com/it/software/jira.

Gitlab

Figura 1.4: Logo Gitlab

Sistema di versionamento dei file di codice usato dall'azienda. Questo strumento si basa su Git[2]. I file vengono condivisi tra tutti gli svilupparori grazie a questo strumento, divisi in repository[g], uno per ogni progetto sul quale l'azienda lavora. La linea guida aziendale per l'utilizzo del sistema Gitlab[3] si basa sul workflow[g]denominato feature branching, questo sistema di lavoro prevede di lasciare il ramo principale (solitamente chiamato main) sempre pulito e con una versione funzionante e testata del prodotto, mentre invece quando si vuole lavorare su una nuova feature si crea un nuovo ramo partendo da quello principale e una volta che questa feature sarà pronta per essere revisionata e integrata con quella principale si chiederà una merge request di questo feature branch sul branch principale. Il mantenimento dei file di codice sul repository si basa sulla regola aziendale 1 modifica - 1 commit - 1 file, ovvero si predilige che ogni branch che lo sviluppatore crea per implementare la relativa feature sia composto da commit che riguardano soltato un file con la relativa modifica.

VS Code

Figura 1.5: Logo VS code

E l'Integrated Development Enviroment (IDE)[g]di lavoro utilizzato per sviluppare in locale. Questo strumento è completo di tutto quello che serve per lo sviluppo del prodotto. Esso infatti consente di sviluppare nei linguaggi che interessano il progetto, aiutando con vari plugin[g]per il riconoscimento del codice lo sviluppatore a lavorare più velocemente e in modo più intuitivo. Inoltre è competamente integrato con i sistemi di versionamento, in particolare nel nostro caso con GitLab. Oltre a queste funzionalità consente di utilizzare plugin per la pulizia del codice, che settati in maniera opportuna permettono di rimanere sempre fedeli alle regole aziendali automaticamente ad ogni salvataggio.

[2] *Git.* URL: https://www.atlassian.com/it/git.
[3] *Gitlab.* URL: https://about.gitlab.com/.

Jitsi

Figura 1.6: Logo Jitsi

E lo strumento che l'azienda utilizza per le riunioni in videocall settimanali ma anche per quelle individuali che dovessero essere necessarie in qualsiasi momento.

SCRUM

Figura 1.7: Logo SCRUM

SCRUM[4] è il framework[g] che l'azienda segue per il lavoro in gruppo sui vari progetti. Questo framework molto diffuso per lo sviluppo software in team si basa su sprint di durata breve che mirano al completamento dei task assegnati, il ripetersi temporale di questi sprint porta l'avanzamento del prodotto. Più precisamente uno sprint è un arco di tempo nel quale si lavora sul completamento di alcune attività che vengono concordate per aggiungere valore al prodotto. Queste attività, solitamente dette task, vengono prese dal backlog, ovvero una lista di attività che si vogliono svolgere per portare avanti il prodotto, il backlog si aggiorna con l'avanzamento del progetto ed è quindi sempre in evoluzione.

Nel nostro caso particolare quella che segue l'azienda è una versione personalizzata di questo framework, il cui funzionamento è spiegato di seguito.

* Durata degli sprint: normalmente una settimana;
* Meeting del lunedì:
  - ognuno da una rapida panoramica per tenere tutti aggiornati su quello che ha svolto nella settimana precendente;
  - si discutono e assegnano i task che si trovavano nel backlog per lo sprint seguente;
  - rapida retrospettiva per valutazioni critiche dello sprint appena terminato;
* Stati dei task da svolgere:
  - da completare: nel corrente sprint ma nessuno ci sta ancora lavorando;
  - in corso: qualcuno sta lavorando a questo task;

[4] *SCRUM*. URL: https://www.atlassian.com/it/agile/scrum.

- pronto per la revisione: il task è stato completato, si attende che l'incaricato revisioni il codice e faccia il merge nel ramo principale;
- completato: il task è stato revisionato e integrato nel ramo principale;
- bloccato: per qualche ragione il task è bloccato, si scrive il motivo per il quale non può essere proseguito e se è necessario parlarne con qualcuno;
- da posticipare: il task deve essere riposto nel backlog e ripianificato per un altro sprint;

1.7 Organizzazione del testo

Il secondo capitolo approfondisce la descrizione dello stage come è stato pianificato, quali sono gli oiettivi ed altri dettagli

Il terzo capitolo approfondisce l'analisi dei requisiti in modo tecnico descrive come è stata svolta la fase di analisi

Il quarto capitolo approfondisce la progettazione del prodotto descrivendo tecnicamente come è stata pensata l'architettura

Il quinto capitolo approfondisce la realizzazione del prodotto con la descrizione della fase di codifica e di testing

Nel sesto capitolo descrive le conclusioni valutando criticamente l'esperienza di stage e il prodotto ottenuto

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

* gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
* per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g];

Capitolo 2

Descrizione dello stage

In questo capitolo viene descritto come si è pianificato e svolto lo stage presso SAI, introducendo il progetto, considerando gli obiettivi e i possibili rischi.

2.1 Analisi preventiva dei rischi

Nella fase di analisi iniziale si sono individuati i principali rischi a cui si poteva andare incontro e si è proceduto a definire le possibili soluzione per farne fronte.

1. Uso di nuove tecnologie
Descrizione: le tecnologie proposte per la gestione e lo sviluppo del progetto erano per lo più nuove o scarsamente conosciute.
Soluzione: è stato previsto un periodo iniziale di studio e formazione su queste tecnologie.

2. Modalità di lavoro smart working
Descrizione: lo stage è stato fatto completamente da remoto, e poteva portare ad una possibile mancanza di comunicazione e ad un incertezza nelle attività da svolgere.
Soluzione: il tutor aziendale si è reso disponibile a vari meeting nelle prime fasi del progetto e nella formazione iniziale, e rimanendo a disposizione per altri meeting in caso di dubbi sulle attività da svolgere.

2.2 Requisiti e obiettivi

Si farà riferimento ai requisiti secondo le seguenti notazioni:

- O per i requisiti obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;
- D per i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
- F per i requisiti facoltativi, rappresentanti valore aggiunto non strettamente competitivo. Le sigle precedentemente indicate saranno seguite da una coppia sequenziale di numeri, identificativo del requisito.

Codice	**Descrizione**
O01	autenticazione mediante server remoto
O02	lettura dati da CD
O03	precompilazione di form con i dati caricati da CD
O04	editing dei dati del form
D01	upload dei dati verso i sistemi esterni
D02	test di unità esaustivi
F01	possibilità di ascoltare le registrazioni
F02	possibilità di modificare i dati mediante interazioni evolute (per es. drag-n-drop)
F03	realizzazione di un'applicazione desktop con Electron
F04	compilazione multipiattaforma dell'applicazione desktop

Tabella 2.1: Tabella del tracciamento dei requisti dello stage

2.3 Pianificazione

La pianificazione, in termini di quantità di ore, sarà distribuita in attività di studio e attività implementative.

Lo stage è stato strutturato in due fasi principali, la prima dedicata alle attività di studio e la seconda alle attività implementative. Di seguito vengono elencate le attività pianificate con la relativa quantità di ore stimata per ogni attività.

* Studio delle tecnologie (80 ore)
  - Setup di un prototipo di applicazione React e del backend (16 ore);
  - Studio delle librerie necessarie (React, Redux, JWT, OAuth) (64 ore);
* Implementazione prototipo (240 ore)
  - Realizzazione dell'interfaccia di login (30 ore);
  - Implementazione lettura e visualizzazione dei dati da CD (70 ore);
  - Implementazione editing dei dati letti da CD (70 ore);
  - Implementazione upload dei dati (30 ore);
  - Testing dei prodotti realizzati (40ore);

Per ogni attività riguardante la fase di implementazione del prototipo è stata prevista anche la relativa attività di testing. Il tutto viene rappresentato dal seguente dal diagramma di Gantt.

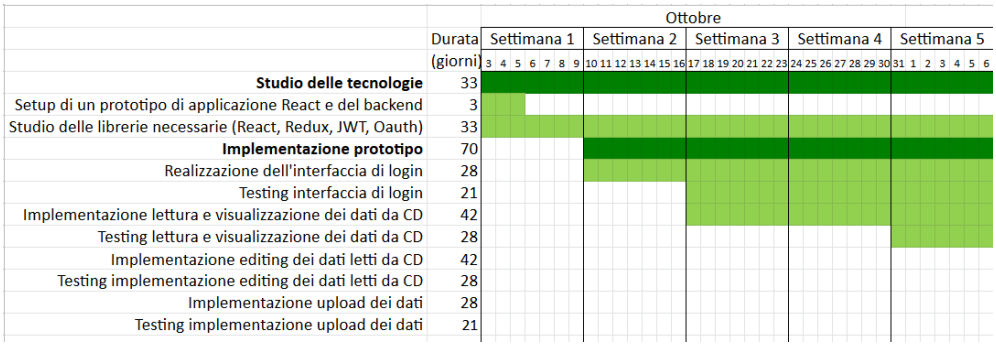


Figura 2.1: Diagramma di Gantt del piano di lavoro dello stage (parte I)

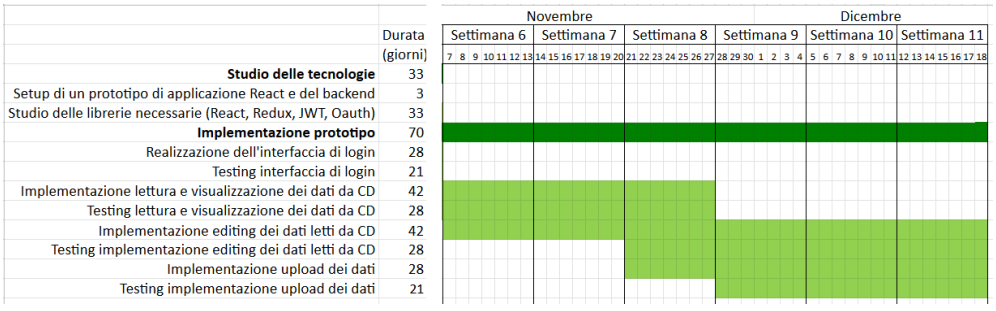


Figura 2.2: Diagramma di Gantt del piano di lavoro dello stage (parte II)

Capitolo 3

Analisi dei requisiti

3.1 Casi d'uso

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi. I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo Unified Modeling Language (UML) dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso. Essendo il progetto finalizzato alla creazione di una applicazione per l'automatizzazione di un processo, le interazioni da parte dell'utente devono essere ridotte allo stretto necessario. Per questo motivo i diagrammi d'uso risultano semplici e in numero ridotto.

3.1.1 UC1 - Autenticazione

* **Identificativo**: UC1
* **Nome**: autenticazione
* **Descrizione grafica**:

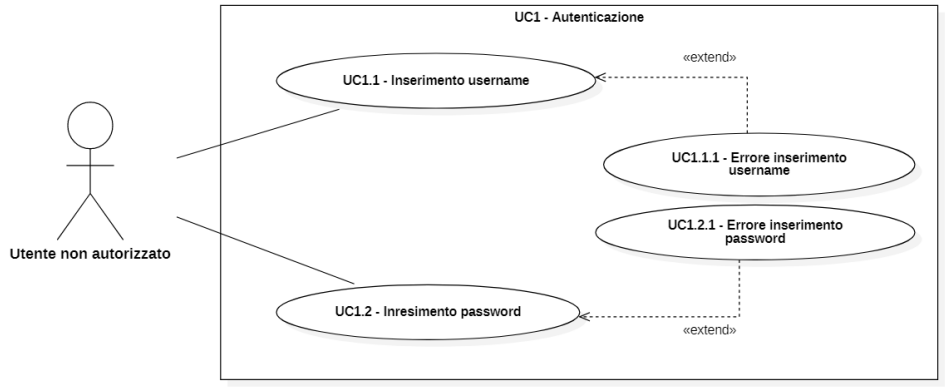


Figura 3.1: Descrizione grafica caso d'uso UC1

* **Attori**
  - *Primari*: utente non autorizzato
* **Precondizione**: l'utente non autenticato si trova sulla pagina di autenticazione.
* **Postcondizione**: l'utente è autenticato.
* **Scenario principale**: l'utente vuole effettuare il login all'applicazione.
* **Scenario secondario**: l'utente non riesce ad autenticarsi a causa di un errore nella procedura. (**UC1.3**)

UC1.1 - Inserimento username

* **Identificativo**: UC1.1
* **Nome**: inserimento username
* **Descrizione grafica**: (approfondita in UC1)
* **Attori**
  - *Primari*: utente non autorizzato
* **Precondizione**: l'utente ha a disposizione una username
* **Postcondizione**: l'utente ha inserito la username.

* **Scenario principale**: l'utente inserisce la username nell'apposito campo di input.
* **Scenario secondario**: l'utente ha inserito una username non corretta che causa un errore. (**UC1.3**)

UC1.1.1 - Errore inserimento username

* **Identificativo**: UC1.1.1
* **Nome**: errore inserimento username
* **Descrizione grafica**: (approfondita in UC1)
* **Attori**
  - *Primari*: utente non autorizzato
* **Precondizione**: la username inserita dall'utente non è correttta.
* **Postcondizione**: l'errore viene mostrato all'utente.
* **Scenario principale**: l'utente inserisce una username non corretta, il sistema segnala l'errore all'utente e mostra nuovamente la maschera di login.

UC1.2 - Inserimento password

* **Identificativo**: UC1.1
* **Nome**: inserimento password
* **Descrizione grafica**: (approfondita in UC1)
* **Attori**
  - *Primari*: utente non autorizzato
* **Precondizione**: l'utente ha a disposizione una password
* **Postcondizione**: l'utente ha inserito la password.
* **Scenario principale**: l'utente inserisce la password nell'apposito campo di input.
* **Scenario secondario**: l'utente ha inserito una password non corretta che causa un errore. (**UC1.3**)

UC1.2.1 - Errore inserimento password

* **Identificativo**: UC1.2.1
* **Nome**: errore inserimento password
* **Descrizione grafica**: (approfondita in UC1)
* **Attori**
  - *Primari*: utente non autorizzato

- **Precondizione**: la password inserita dall'utente non è correttta.
- **Postcondizione**: l'errore viene mostrato all'utente.
- **Scenario principale**: l'utente inserisce una password non corretta, il sistema segnala l'errore all'utente e mostra nuovamente la maschera di login.

3.1.2 UC2 - Lettura dati da CD

* **Identificativo**: UC2
* **Nome**: lettura dati da CD
* **Descrizione grafica**:

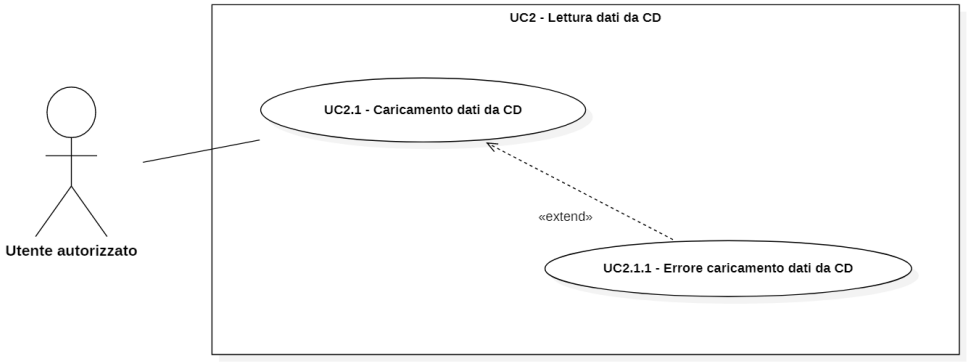


Figura 3.2: Descrizione grafica caso d'uso UC2

* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente autorizzato si trova nella pagina per il caricamento dei dati.
* **Postcondizione**: l'utente ha caricato i dati da CD.
* **Scenario principale**: l'utente premendo sull'apposito bottone può caricare i file contenuti nel CD di interesse e li visualizza (**UC3**).

UC2.1 - Caricamento dati da CD

* **Identificativo**: UC2.1
* **Nome**: caricamento dati da CD
* **Descrizione grafica**: (approfondita in UC2)
* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente ha richiesto il caricamento di una cartella contenente i file del CD.
* **Postcondizione**: i file sono stati caricati nell'applicazione.
* **Scenario principale**: l'utente richiede il caricamento di una cartella contenente i file del CD premendo sull'apposito bottone.

*

* **Scenario secondario**: il sistema riscontra un errore nella procedura di caricamento dei dati. (**UC2.1**)

UC2.1.1 - Errore caricamento dati da CD

* **Identificativo**: UC2.1
* **Nome**: errore caricamento dati da CD
* **Descrizione grafica**: (approfondita in UC2)
* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente ha tentato di caricare i dati.
* **Postcondizione**: l'errore viene mostrato all'utente.
* **Scenario principale**: il sistema non è riuscito a gestire la richiesta di caricamento dei dati da parte dell'utente.

3.1.3 UC3 - Visualizzazione file

* **Identificativo**: UC3
* **Nome**: visualizzazione file
* **Descrizione grafica**:

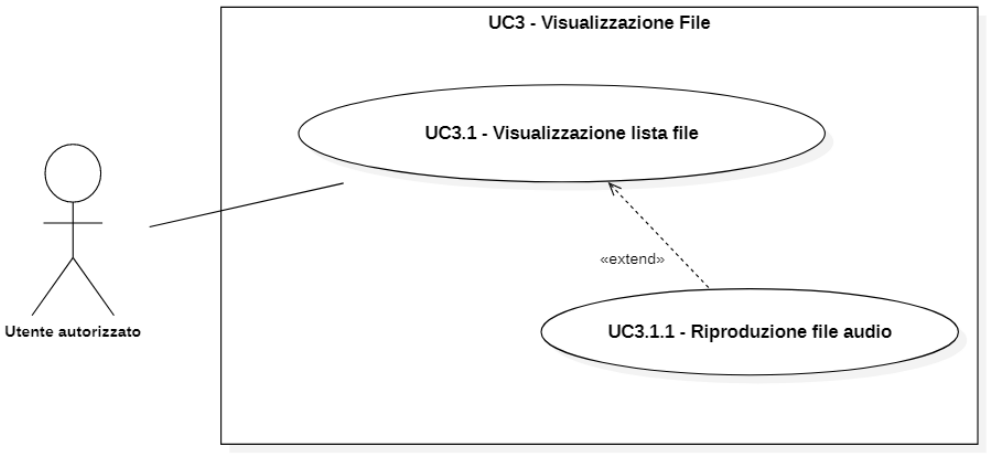


Figura 3.3: Descrizione grafica caso d'uso UC3

* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente si trova sulla pagina per il caricamento dati, che ha già effettuato correttamente.
* **Postcondizione**: l'utente visualizza i file e i relativi metadati caricati.
* **Scenario principale**: l'utente ha caricato correttamente i dati, questi vengono visualizzati con una lista di file ognuno con i relativi metadati.
* **Scenario secondario**: l'utente può visualizzare ed elaborare i metadati relativi ad ogni file **UC5**.

UC3.1 - Visualizzazione lista file

* **Identificativo**: UC3.1
* **Nome**: visualizzazione lista file
* **Descrizione grafica**: (approfondita in UC3)
* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente si trova sulla pagina per il caricamento dati, che ha già effettuato correttamente.

* **Postcondizione**: l'utente visualizza la lista dei file ognuno con i relativi metadati.
* **Scenario principale**: l'utente visualizza la lista dei file e può visualizzare la lista dei metadati associati.
* **Scenario secondario**: l'utente può riprodurre i file audio **UC3.1.1**.

UC3.1.1 - Riproduzione audio file

* **Identificativo**: UC3.1.1
* **Nome**: riproduzione audio file
* **Descrizione grafica**: (approfondita in UC3)
* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente si trova sulla pagina per il caricamento dati, che ha già effettuato correttamente.
* **Postcondizione**: il file audio viene riprodotto.
* **Scenario principale**: l'utente comanda il riproduttore audio con gli appositi comandi play/pause.

3.1.4 UC4 - Visualizzazione procedimenti

* **Identificativo**: UC4
* **Nome**: visualizzazione procedimenti
* **Descrizione grafica**:

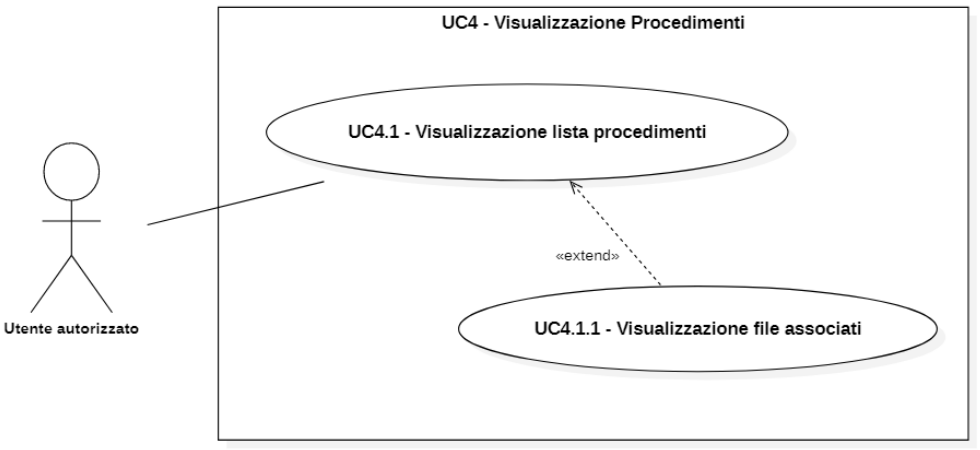


Figura 3.4: Descrizione grafica caso d'uso UC4

* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente ha già effettuato correttamente il caricamento dati da CD.
* **Postcondizione**: l'utente visualizza i procedimenti relativi ai dati caricati.
* **Scenario principale**: l'utente da questa vista, può visualizzare la lista dei procedimenti e i file relativi ad ognuno.
* **Scenario principale**: l'utente può modificare i metadati di ogni procedimento. (**UC5.1**)

UC4.1 - Visualizzazione lista procedimenti

* **Identificativo**: UC4.1
* **Nome**: visualizzazione lista procedimenti
* **Descrizione grafica**: (approfondita in UC4)
* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente si trova sulla pagina di elenco procedimenti.

* **Postcondizione**: l'utente visualizza la lista dei procedimenti ognuno con i relativi file.
* **Scenario principale**: l'utente visualizza la lista dei procedimenti e può visualizzare la lista dei file associati.
* **Scenario secondario**: l'utente può visualizzare la lista dei file associati ad ogni procedimento.

UC4.1.1 - Visualizzazione file associati

* **Identificativo**: UC4.1.1
* **Nome**: visualizzazione file associati
* **Descrizione grafica**: (approfondita in UC4)
* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente si trova sulla pagina di elenco procedimenti.
* **Postcondizione**: l'utente visualizza la lista dei fie associati ad un procedimento.
* **Scenario principale**: l'utente visualizza la lista dei procedimenti e con un apposito bottone può visualizzare la lista dei file associati.

3.1.5 UC5 - Elaborazione metadati

* **Identificativo**: UC5
* **Nome**: elaborazione metadati
* **Descrizione grafica**:

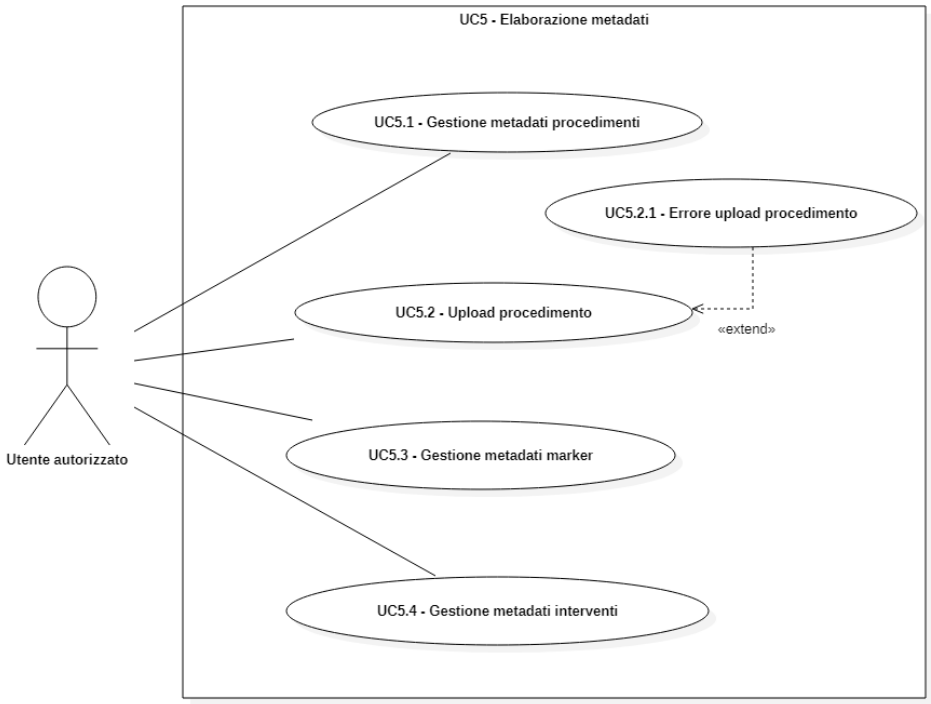


Figura 3.5: Descrizione grafica caso d'uso UC5

* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente si trova sulla pagina di visualizzazione dei file con i relativi metadati.
* **Postcondizione**: l'utente ha visualizzato ed eventualmente elaborato i metadati desiderati.
* **Scenario principale**: l'utente può visualizzare i metadati relativi ad ogni file caricato.
* **Scenario secondario**: l'utente può gestire (modificare aggiungere eliminare) i procedimenti relativi ad ogni file. (**UC5.1**)
* **Scenario secondario**: l'utente può gestire (modificare aggiungere eliminare) i marker relativi ad ogni file. (**UC5.3**)
* **Scenario secondario**: l'utente può gestire (modificare aggiungere eliminare) gli interventi relativi ad ogni file. (**UC5.4**)

UC5.1 - Gestione metadati Procedimenti

* **Identificativo**: UC5.1
* **Nome**: getione metadati Procedimenti
* **Descrizione grafica**: (approfondita in UC5)
* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente vuole elaborare procedimenti relativi ad un file.
* **Postcondizione**: l'utente ha elaborato i procedimenti.
* **Scenario principale**: l'utente può modificare, aggiungere o eliminare i procedimenti relativi ad un file.

UC5.2 - Upload procedimento

* **Identificativo**: UC5.2
* **Nome**: upload procedimento
* **Descrizione grafica**: (approfondita in UC5)
* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente vuole fare l'upload dei metadati riguardanti un procedimento relativi ad un file.
* **Postcondizione**: l'utente ha fatto l'upload del procedimento desiderato.
* **Scenario principale**: l'utente può tramite un apposito bottone fare l'upload del file e i relativi metadati del procedimento che desidera.
* **Scenario secondario**: si è verificato un errore nella richiesta di upload del procedimento. (**UC5.2.1**)

UC5.2.1 - Errore upload procedimento

* **Identificativo**: UC5.2.1
* **Nome**: errore upload procedimento
* **Descrizione grafica**: (approfondita in UC5)
* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: il sistema non ha gestito correttamente la richiesta di upload procedimento.
* **Postcondizione**: l'errore viene visualizzato sull'applicazione.
* **Scenario principale**: la richiesta di upload procedimento non va a buon fine e l'errore viene mostrato all'utente.

UC5.3 - Gestione metadati Marker

* **Identificativo**: UC5.3
* **Nome**: getione metadati Marker
* **Descrizione grafica**: (approfondita in UC5)
* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente vuole elaborare i marker relativi ad un file.
* **Postcondizione**: l'utente ha elaborato i marker.
* **Scenario principale**: l'utente può modificare, aggiungere o eliminare i marker relativi ad un file.

UC5.4 - Gestione metadati Interventi

* **Identificativo**: UC5.4
* **Nome**: getione metadati Intervetni
* **Descrizione grafica**: (approfondita in UC5)
* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente vuole elaborare gli interventi relativi ad un file.
* **Postcondizione**: l'utente ha elaborato gli interventi.
* **Scenario principale**: l'utente può modificare, aggiungere o eliminare gli interventi relativi ad un file.

3.1.6 UC6 - Visualizzazione jobs

* **Identificativo**: UC6
* **Nome**: visualizzazione jobs[g]
* **Descrizione grafica**:

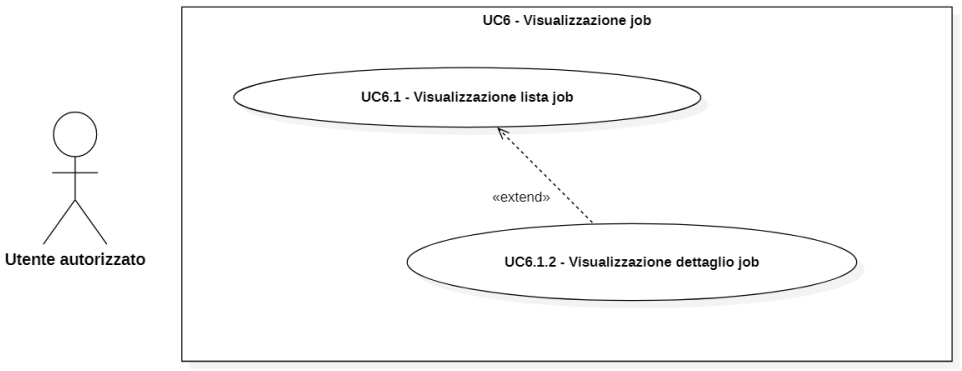


Figura 3.6: Descrizione grafica caso d'uso UC6

* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente si trova all'interno dell'applicazione e vuole visualizzare la lista dei jobs.
* **Postcondizione**: l'utente visualizza la lista dei jobs.
* **Scenario principale**: l'utente può visualizza la lista dei jobs (**UC6.1**) con un piccolo riassunto delle informazioni principali di ognuno.
* **Scenario secondario**: l'utente può visualizzare il dettaglio di un singolo job premendo sull'apposito link. (**UC6.2**)

UC6.1 - Visualizzazione lista dei job

* **Identificativo**: UC6.1
* **Nome**: visualizzazione lista dei job
* **Descrizione grafica**: (approfondita in UC6)
* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente ha premuto sull'apposito link per visualizzare la lista dei job.
* **Postcondizione**: la lista dei job viene visualizzata.

* **Scenario principale**: l'utente visualizza la lista dei job paginata e ordinata per ultimo caricato.

UC6.1.2 - Visualizzazione dettaglio job

* **Identificativo**: UC6.2
* **Nome**: visualizzazione dettaglio job
* **Descrizione grafica**: (approfondita in UC6)
* **Attori**
  - *Primari*: utente autorizzato
* **Precondizione**: l'utente ha premuto sull'apposito link per visualizzare il dettaglio di un job.
* **Postcondizione**: il dettaglio del job viene visualizzato.
* **Scenario principale**: l'utente visualizza tutte le informazioni riguardanti il job desiderato.

3.2 Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e degli use case effettuata sul progetto è stata stilata la tabella che traccia i requisiti in rapporto agli use case.
Sono stati individuati diversi tipi di requisiti e si è quindi fatto utilizzo di un codice identificativo per distinguerli.
Il codice dei requisiti è così strutturato R(F/Q/V)(O/D/F) dove:

R = requisito

F = funzionale

Q = qualitativo

V = di vincolo

O = obbligatorio

D = desiderabile

F = facoltativo

Nelle tabelle ?? e ?? sono riassunti i requisiti e il loro tracciamento con gli use case delineati in fase di analisi.

Requisito	**Descrizione**	**Use Case**
RFO-1	autenticazione	UC1
RFO-2	lettura dati da CD	UC2
RFO-3	visualizzazione dati ordinati per file	UC3
RFO-4	visualizzazione dati ordinati per procedimenti	UC4
RFO-5	gestione dei Marker relativi al file	UC3
RFO-6	gestione degli Interventi relativi al file	UC3
RFD-1	upload procedimento	UC5
RFF-2	visualizzazione jobs	UC6
RFF-1	riproduzione audio file	UC3.1

Tabella 3.1: Tabella del tracciamento dei requisti funzionali

Requisito	**Descrizione**	**Use Case**
RQD-1	test di unità esaustivi	-

Tabella 3.2: Tabella del tracciamento dei requisiti qualitativi

Capitolo 4

Progettazione

In questo capitolo vengono spiegate in modo dettagliato le tecnologie utilizzate nella realizzazione del progetto, inoltre viene data una spiegazione generale di come è stata progettata l'architettura generale del sistema e una più approfondita riguardante l'architettura del frontend, che era oggetto del progetto di stage.

4.1 Tecnologie

Di seguito viene data una panoramica delle tecnologie utilizzate.

Javascript

Figura 4.1: Logo Javascript

JavaScript è un linguaggio di programmazione orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione web lato client per la creazione di applicazioni web. L'intera applicazione è stata scritta con questo linguaggio.

React

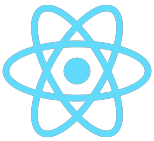


Figura 4.2: Logo React

React[1] è una libreria Javascript utilizzata per implementare User Interface (UI)[g]lato frontend. React si basa sul concetto di component, idealmente è una libreria che permette di costruire i propri component come fossere degli elementi HyperText Markup Language (HTML)[g]del Document Object Model (DOM)[g]per poi poterli riusare nell'intera applicazione.

Redux & Redux RTK

Figura 4.3: Logo Redux

Redux[2] è un contenitore dello stato dell'applicazione[g]per le applicazioni Javascript. Viene usato per la gestione centralizzata dello stato delle applicazioni sviluppate in React Javascript. In particolare con la sua libreria Redux-Toolkit, permette una gestione dello stato semplice ed efficente.

MUI

Figura 4.4: Logo MUI

Material UI[3] è una libreria React open-source[g]che permette di implementare i Google's Material Design. Essa comprende una collezione di componenti React precostruiti che possono essere facilmente adattati e messi in uso nella UI dell'applicazione.

React Router

Figura 4.5: Logo React router

React Router è la libreria standard per il routing[g]in React. Questa libreria permette la navigazione tra le varie viste dell'applicazione, permette di gestire le Uniform Resource Locator (URL)[g], e mantenere la sincronizzazione tra URL e viste.

[1] *React.* URL: https://reactjs.org/.
[2] *Redux.* URL: https://redux.js.org/.
[3] *MUI.* URL: https://mui.com/.

4.2 Architettura dell'applicazione

L'architettura generale dell'applicazione era già stata progettata prima dell'inizio dello stage in oggetto, e in parte già esistente. Il sistema si divide principalmente in tre parti:

* **backend**
* **frontend**
* **object storage**

La comunicazione tra le diverse parti del sistema invece avviene con l'utilizzo di **Application Program Interface (API)**[g].

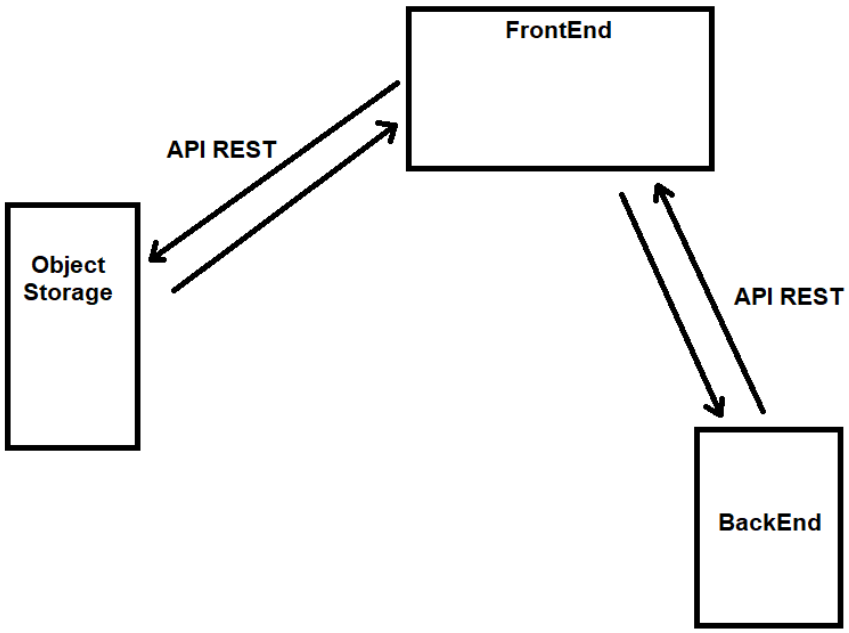


Figura 4.6: Descrizione grafica dell'architettura generale dell'applicazione

4.2.1 Backend

Il backend è un'applicazione a se stante realizzata interamente in Python, con architettura Model View Controller (MVC)[g]. Questa architettura permette di separare completamente la logica del prodotto dal modello dei dati, le viste non sono state approfondite in quanto lo scopo principale del backend è gestire e memorizzare i dati che vengono passati dal frontend e non visualizzarli. Il backend espone delle particolari URL come endpoint[g]per le chiamate REpresentational State Transfer (REST)[g]del client frontend. Il backend utilizza il modello (Model) per rappresentare i dati di interesse, il controller per gestirli, e la vista (View) per rappresentarli. Nel nostro caso, come già menzionato, non dovendo rappresentare i dati (operazione che spetta al frontend), esso espone dell URL come endpoint per le chiamate REST proprio del

frontend. In questo modo ogni volta che viene fatta una richiesta su una corretta URL al backend, esso la interpreta con l'apposito controller, gestendo i dati strutturati come nel modello, e dopo aver completato la gestione della richiesta, ne restituisce la risposta al frontend.

4.2.2 Frontend

Il frontend dell'applicazione è generalmente la parte di interfaccia per l'utente, ossia quello che l'utente visualizza del nostro sistema e che gli da l'opportunità di interagire con il backend. Nel nostro caso il frontend non fa solo da interfaccia utente ma gestisce anche dei dati nel proprio stato fin tanto che questi rimangono in sessione[g], in modo da non fare continue richieste al backend che rallenterebbero molto il sistema a causa della mole dei dati da gestire e visto che non tutti i dati devono essere passati al backend. Il frontend è stato scritto in JavaScript e in particolare usando il framework React, gestendo però lo stato esternamente usando Redux, questo porta ad un'architettura leggermente più complessa dell'intero frontend ma da grandi benefici in quanto aiuta a mantenere per quanto possibile la separazione tra logica dell'applicazione e reppresentazione dei dati. In generale l'architettura è quella della single page application, cioè una pagina che non necessita di essere totalmente ricaricata ad ogni modifica ma che va a modificare soltanto la porzione interessata da uno specifico cambiamento.

4.2.3 Object storage

L'object storage che si è deciso di utilizzare è minIO (scelta aziendale, legata anche ad altri progetti) e nel caso della nostra applicazione serve in particolare come stumento per memorizzare i file con delle semplici richieste tramite API. Questo ci permette essenzialmente due cose molto importanti, non doverci preoccupare troppo dei dettagli implementativi della memeorizzazione e allo stesso tempo poter reperire agevolmente i file che ci servono. Per il caricamento di un file è necessario fare una richiesta PUT, ad una apposita URL di minIO passando come parametro il file che si desidera salvare. Questa URL viene concordata tra backend e minIO stesso in modo da essere sempre univoca per ogni nuovo file caricato, questo comporta che l'operazione finale di salvataggio del file comprenda tre richieste:

* **frontend -> backend**: con questa richiesta il frontend comunica quale file vuole salvare al backend che può così memorizzarne i dati, in modo da tenerne traccia qualora si voglia in futuro reperire il file caricato;

* **backend -> minIO**: il backend richiede su un'apposita URL a minIO di comunicargli la vera e propria URL univoca sulla quale fare la richiesta di salvataggio del file, dopo che l'ha ottenuta la da in risposta al frontend;

* **frontend -> minIO**: il frontend dopo aver ricevuto la corretta URL sulla quale effettuare la richiesta PUT può eseguirla passando il file come parametro

Per la versione finale del progetto, non è escluso di rivedere questa parte che si prevede una delle più importanti, ma allo stesso tempo difficile da eseguire in modo efficace ed efficiente. Per lo stato che il prodotto deve avere alla fine dello stage, invece, questa gestione è più che sufficiente.

4.2.4 API rest

Le API rest sono il tramite tra le due applicazioni, backend e frontend, permettendo la comunicazione e il passaggio di dati. Vengono gestite nella parte frontend con Redux-tolkit, uno strumento apposito per la gestione delle richieste API con Redux. Le richieste del frontend interrogano il backend sugli appositi endpoint, e dopo aver ricevuto la risposta ne interpretano il risultato. Le richieste http rest possono essere di vario tipo, ognuno dei quali rappresenta una particolare operazione che il client richiede al server di fare, nel nostro caso per la comunicazione tra backend e frontend sono state utilizzate richieste di tipo GET e POST. Inoltre una particolare richiesta di tipo PUT viene effettuata dal frontend verso l'object storage minIO per l'upload dei file. Il linguaggio scelto per la comunicazione tramite API è JavaScript Object Notation (JSON)[g], tra i più diffusi per questo tipo di operazioni per la sua semplicità di utilizzo.

4.3 Architettura frontend

L'architettura del frontend è stata progettata per essere poi implementata con le tecnologie che avevamo già in mente di usare e cioè in primis React e Redux. La progettazione del frontend è molto semplice e si basa sull'interazione tra queste due tecnologie. React realizza tutte le nostre viste, tramite dei component che possono essere integrati tra loro e riutilizzati, lo stato dell'applicazione invece è contenuto nello store di Redux, queste due entità comunicano in due modi:

* **da component a store**: gli eventi Javascript/React che hanno bisogno di utilizzare lo stato dell'applicazione richiamano lo store con appositi metodi;

* **da store a component**: lo store invia ai component React le parti di stato che hanno richiesto o modificato così possono essere aggiornate;

Per capire nel dettaglio l'architettura del frontend ricorriamo al seguente schema di approfondimento.

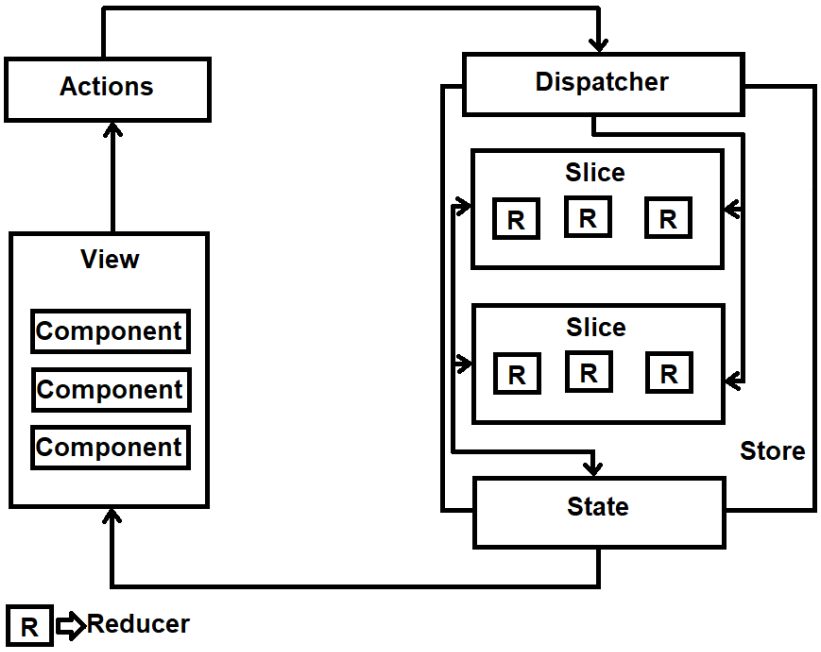


Figura 4.7: Descrizione grafica dell'architettura del frontend dell'applicazione

Il frontend dell'applicazione è stato progettato come una single page application, che mantiene completa seprarazione tra lo stato e la user interface, questo grazie proprio all'utilizzo delle tecnologie scelte. Lo store Redux implementa quello che può essere visto come un modello di dati per i componenti React, essi invece svolgono sia la funzione di vista che quella di controller, in quello che può essere idealizzato come un pattern Model-View-ViewModel (MVVM)[g], la natura delle tecnologie in causa però fa si che questo pattern non sia stretto e vincolante per tutta l'applicazione. In generale l'applicazione risulta più semplicemente governata dal principio azione-reazione, infatti con la logica dei component React risulta immediato associare ad ogni interazione dell'utente con la UI un evento. L'evento associato ad un azione dell'utente viene gestito in vari modi, può coinvolgere una semplice modifica della UI, può implicare una modifica dello store e quindi una rielaborazione della UI (solo nelle parti coinvolte) oppure può lanciare una richiesta tramite API al backend.

4.4 Design Pattern utilizzati

Alcuni design pattern utilizzati li abbiamo già citati, andiamo di seguito ad elencare quelli utilizzati dall'applicazione, alcuni sono nativi o intrisechi delle tecnologie utilizzate altri invece pensati per la nostra specifica applicazione, l'utilizzo di tante tecnologie diverse integrate tra loro porta ad un utilizzo di alcuni design pattern in modo spurio, questi quindi non sono presenti e implementati pedissequamente però risultano comunque in alcune forme o per alcuni aspetti chiave.

* **MVVM**: come già citato in precedenza, l'integrazione tra React e Redux da vita ad una forma di questo pattern, se consideriamo lo store Redux come modello e i component React come viste con logica al proprio interno;
* **MVC**: non presente nel frontend ma bensi nel backend realizzzato in Python;
* **Observer**: nel frontend i componenti React sono dei subject sottoscritti allo stato dell'applicazione, quando questo cambia per qualsiasi motivo, i componenti vengono modificati

4.5 Progettazione dell'applicazione

Nel dettaglio della progettazione della nostra applicazione abbiamo pensato ad un funzionamento lineare, semplice ed intuitivo per l'utente finale, in modo da continuare a perseguire le caratteristiche di velocità e facilità di utilizzo dell'applicazione, che ci eravamo prefissati in fase di analisi del progetto. L'architettura del sistema permetterà il funzionamento che possiamo riassumere nei passi che spieghiamo di seguito.

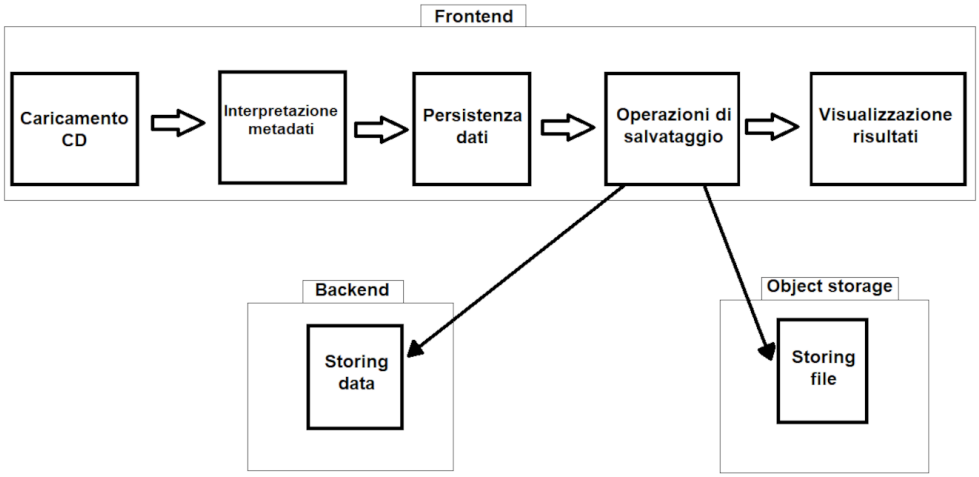


Figura 4.8: Schema del funzionamento della architettura per la nostra applicazione

Caricamento CD

La prima operazione da fornire all'utente sarà il caricamento del CD contenete file audio e metadati che sono l'input della nostra applicazione.

Interpretazione metadati

I metadati dovranno essere interpretati da una libreria di nostra creazione in modo da renderli disponibili all'interno dell'applicazione nei formati che possiamo trattare agevolmente.

Persistenza dati/metadati sul frontend

I metadati e i file caricati dall'utente e interpretati dalla nostra libreria ora sono pronti per essere memorizzati nel frontend, qui procederemo al trattamento di questi dati con tutte le operazioni che metteremo a disposizione dell'utente, con lo scopo di creare i ticket che sono necessari.

Passaggio dati al backend

Dopo che l'utente ha svolto tutte le operazioni sui dati che riteneva opportune, creando i ticket, i dati vengono passati al nostro backend per essere memorizzati in modo permanente, contestualmente a questa operazione vengono trattate le traccie audio coinvolte nei ticket, come spieghiamo di seguito.

Storing file

Nel momento in cui avviene la richiesta per il caricamento dei dati necessari alla creazione dei ticket verso il backend, una richiesta parallela verrà inviata all'object storage per salvare i file audio coinvolti nella creazione di questi ticket. La risposta di queste due operazione positiva o negativa verrà data all'utente che nel caso una delle due o entrambe non siano andate a buon fine potrà ripeterle.

Visualizzazione risultati frontend

Per concludere, dopo che le operazioni di memorizzazione dei dati sul backend e salvataggio dei file sull'object storage sono andate a buon fine, questi dati con i relativi file vengono inviati in risposta al frontend in modo che l'applicazione possa esporli all'utente, in una sorta di riepilogo dei dati coinvolti nell'operazione che è avvenuta con successo.

Capitolo 5

Realizzazione

In questo capitolo viene spiegato come si è agito in fase di codifica e testing, andando nel dettaglio dell'implementazione e dell'uso delle tecnologie.

5.1 Sviluppo

5.1.1 Dati e metadati

Prima di addentrarci nella descrizione dell'implementazione del codice, vediamo una descrizione accurata dei dati e metadati a cui il prodotto fa riferimento.

Dati

I dati del progetto sono i file che l'utente carica nell'applicazione, in particolare come già visto, l'utente può caricare sia singoli file che, come procedura normale per il flusso dell'applicazione, una cartella di file che rappresenta il CD registrato. Questi file seguono una precisa nomenclatura che viene data in automatico dal sistema di registrazione.

Cronologia E il file che contiene i metadati generati automaticamente dal sistema di registrazione o inseriti dal tecnico in aula. Esempio di una porzione di file cronologia:

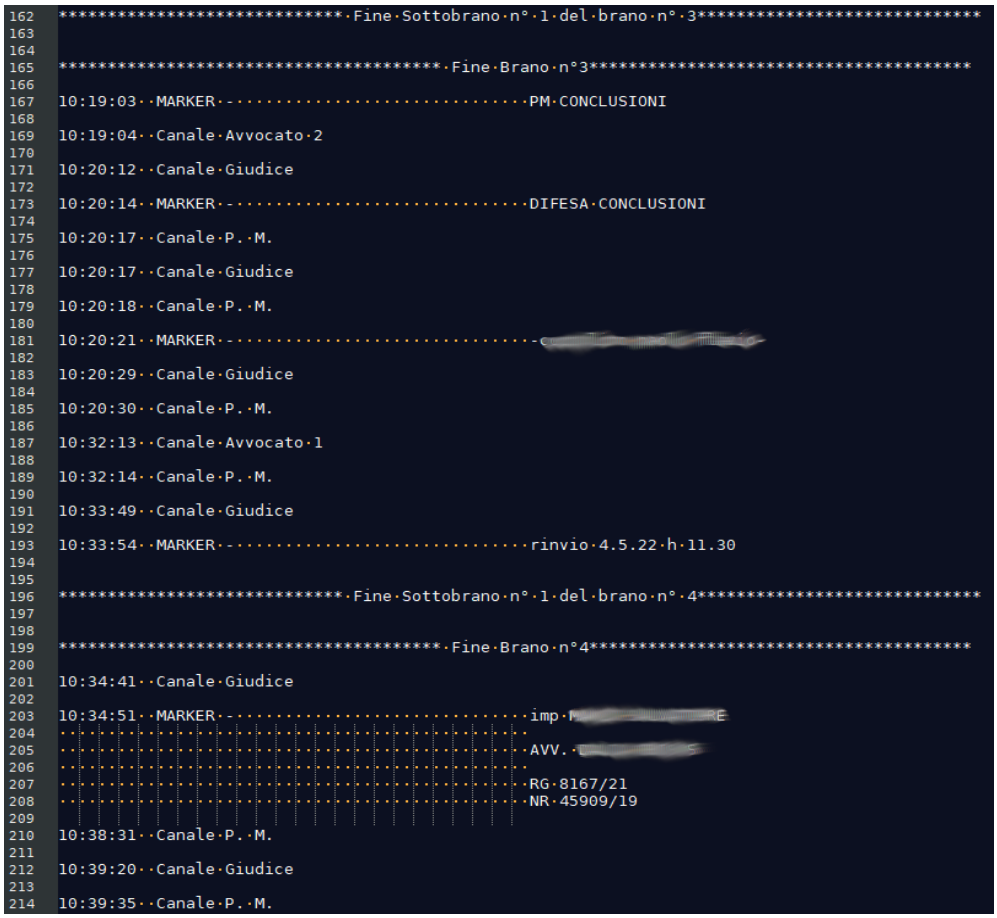


Figura 5.1: Porzione di file cronologia

Come possiamo vedere dall'immagine all'interno del file cronologia troviamo i metadati. Nello specifico possiamo vedere in completo il sottobrano[g] numero 1 (in questo caso l'unico) del brano[g] numero 4. All'interno di questo sottobrano troviamo i vari canale audio[g] con gli orari in cui in aula sono stati fatti degli interventi, con il nome di riferimento dei canali dedicati si può facilmente intuire chi abbia preso parola in un dato istante. Inoltre possiamo vedere vari marker che sono gli appunti che il fonico prende nel corso della registrazione e che interpretati come avviene nella libreria creata appositamente formano i vari procedimenti. Nel caso specifico di questo sottobrano abbiamo una chiusura di un procedimento, come si può vedere dal marker che dichiara il "rinvio". Scorrendo in giù il nostro file cronologia possiamo vedere come inizi il sottobrano numero 1 del brano numero 5 dove troviamo, grazie al marker che contiene i codici (RG e RGNR), l'inizio di un nuovo procedimento che continuerà fino ad un nuovo marker di chiusura.

Traccia audio E un file mp3 che contiene la registrazione di un canale audio in particolare, che può fare riferimento ad un solo microfono, oppure all'insieme dei microfoni descritto come traccia mixer[g]. Esempio di file all'interno di una cartella:

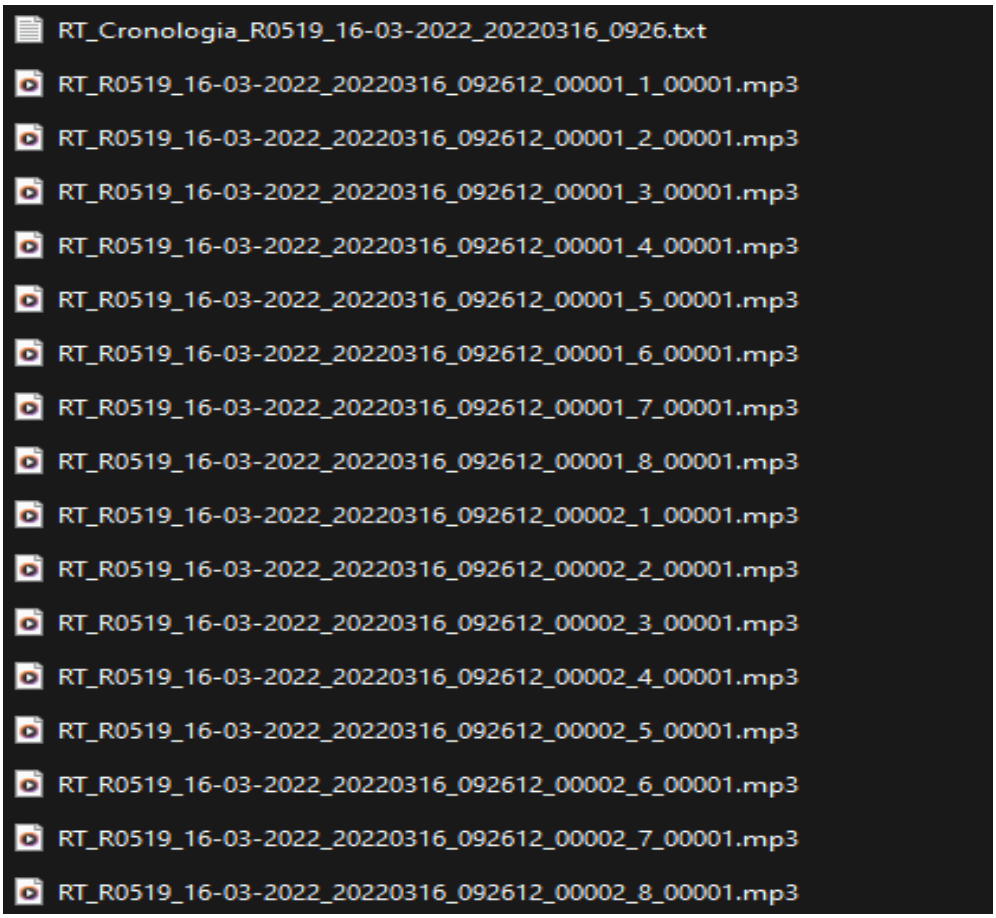


Figura 5.2: I file all'interno di un CD

* `RT_Cronologia_R0519_16-03-2022_20220316_0926.txt`: file **cronologia** che regola tutte le tracce audio presenti nella cartella CD;

* `RT_R0519_16-03-2022_20220316_092612_00001_1_00001.mp3`: file audio **traccia mixer**, contiene la registrazione completa di tutti i microfoni presenti in aula per il Brano 1 e Sottobrano 1;

* `RT_R0519_16-03-2022_20220316_092612_00001_2_00001.mp3`: file audio **giudice**, contiene la registrazione completa del microfono dedicato al giudice per il Brano 1 e Sottobrano 1;

* `RT_R0519_16-03-2022_20220316_092612_00001_3_00001.mp3`: file audio **P.M.**, contiene la registrazione completa del microfono dedicato al Pubblico Ministero per il Brano 1 e Sottobrano 1;

* `RT_R0519_16-03-2022_20220316_092612_00001_4_00001.mp3`: file audio **avvocato 1**, contiene la registrazione completa del microfono dedicato all'avvocato 1 per il Brano 1 e Sottobrano 1;

* `RT_R0519_16-03-2022_20220316_092612_00001_5_00001.mp3`: file audio **avvocato 2**, contiene la registrazione completa del microfono dedicato all'avvocato 2 per il Brano 1 e Sottobrano 1;
* `RT_R0519_16-03-2022_20220316_092612_00001_6_00001.mp3`: file audio **imputato**, contiene la registrazione completa del microfono dedicato all'imputato per il Brano 1 e Sottobrano 1;
* `RT_R0519_16-03-2022_20220316_092612_00001_7_00001.mp3`: file audio **testimone**, contiene la registrazione completa del microfono dedicato al testimone per il Brano 1 e Sottobrano 1;
* `RT_R0519_16-03-2022_20220316_092612_00001_8_00001.mp3`: file audio **ausiliario**, contiene la registrazione completa del microfono ausiliario per il Brano 1 e Sottobrano 1.

Metadati

I metadati in questo progetto fanno riferimento a ciò che è scritto all'interno del file di cronologia, che viene interpretato dalla libreria appositamente creata per il parsing[g] del file. Questi metadati possono essere di svariati tipi, dipendono dal sistema che li scrive su questo particolare file e dal tecnico che appunta ciò che succede in aula. Di seguito una lista dei tipi di metadati che sono stati ritenuti rilevanti per l'applicazione e che vengono cercati all'interno del file cronologia per poi essere interpretati:

* **Procedimenti**: questi sono il cuore dei metadati, sono appuntati manualmente dai tecnici e per questo non sempre precisi e presenti. Per ovviare a queste possibilità la decisione è stata quella di lasciare la possibilità all'utente di modificarli dopo il caricamento e l'interpretazione da parte della libreria apposita. Il dato più rilevante e di interesse per l'applicazione è la combinazione dei due codici **RG** e **RGNR** che identifica in modo univoco il procedimento.
* **Interventi**: questi metadati sono di supporto alle registrazioni, e sono automaticamente generati e scritti sul file cronologia dal sistema di registrazione dei file. Come facilmente intuibile dal nome scelto per loro riportano gli interventi che sono stati fatti in aula, specificando con delle informazioni da quale microfono e a che ora;
* **Marker**: questi metadati invece non sono generati automaticamente dal sistema ma vengono scritti dai tecnici e inseriti nel file cronologia mediante il sistema di registrazione. Essi riportano informazioni di qualsiasi tipo, inizio di un procedimento, fine, parole che magari non si sentono bene in aula o informazioni sui partecipanti alla sessione giudiziaria. Vengono riassunti con due campi, uno per l'ora nella quale è stato riportato l'appunto e un altro per la descrizione testuale dell'appunto.

5.1.2 Libreria per il parsing

Data la particolare natura dei dati e metadati trattati dall'applicazione si era pensato in sede di analisi e poi di progettazione di realizzare una libreria apposita per interpretarli. In particolare quello che rendeva indispensabile la creazione di una libreria apposita è la varietà di casi che si possono presentare nei metadati del file cronologia, in questo file infatti troviamo dei casi abbastanza standard che riguardano gli interventi al microfono

dei partecipanti alle udienze, ma troviamo anche i marker scritti dai fonici mentre l'udienza scorre in aula che assomigliano molto a degli appunti, che possono essere quindi i più disparati possibili, ma che hanno grande importanza nella nostra applicazione. Questi marker, infatti, contengono delle parole chiave che possono determinare molte cose di grande importanza tra cui l'inizio di un procedimento, la sua fine, il suo esito e molto altro. Nonostante la grande rilevanza di questi marker per la logica della nostra applicazione, che si basa interamente su questi metadati, essi sono per la maggior parte scritti manualmente dai tecnici delle aule di tribunale e soggetti quindi ad errori di vario tipo. La libreria che abbiamo creato prende in input questi metadati caricati dall'utente e dopo una serie di interpretazioni, di vario genere, da in output i dati che l'utente visualizza all'interno dell'applicazione. Il suo funzionamento si basa su varie funzioni create per l'interpretazione del testo, in particolare per trasformare il linguaggio naturale (testo scritto dai fonici come i marker) in oggetto javascript[g] da poter usare all'interno dell'applicazione. La libreria è divisa in due parti principali, la prima prende il testo semplice del file cronologia, lo interpreta e ne restituisce una rappresentazione molto complessa, strutturata in oggetti javascript, la seconda prende questa complessa rappresentazione ad oggetti del file cronologia e la trasforma in array di oggetti[g] (appositamente strutturati) da passare allo store Redux. Da qui i dati saranno disponibili all'interno dell'applicazione per l'utente finale.

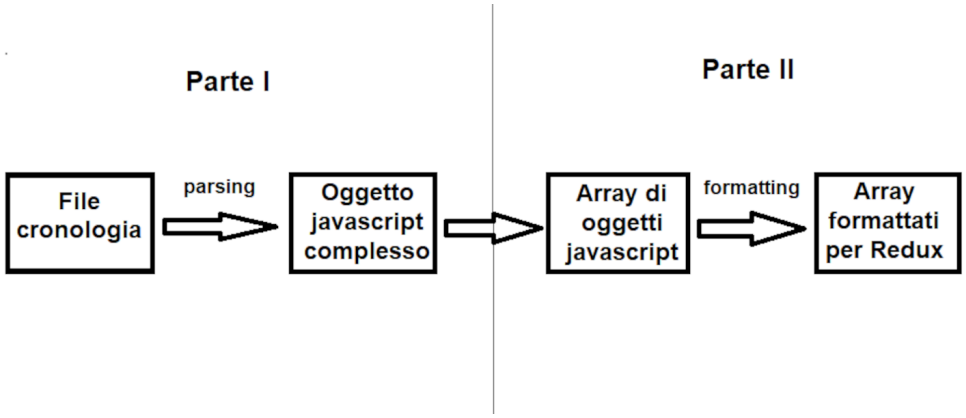


Figura 5.3: Struttura della libreria

Di seguito riassumiamo il corso dei dati/metadati e il funzionamento della libreria:

* **input file**: in input la libreria riceve un file cronologia nel formato che abbiamo già visto nel precedente capitolo;
* **parsing del file**: il file viene interpretato e trasformato in un oggetto javascript di struttura complessa (come in figura 5.4);
* **parsing dell'oggetto**: l'oggetto viene trasformato in una serie di array di oggetti javascript che contengono i metadati del file facilmente utilizzabili in qualsiasi applicazione javascript (come in figura 5.5);
* **output array**: gli array di oggetti vengono formattati e messi a disposizione per essere caricati e utilizzati nello store Redux che abbiamo scelto come gestore dello stato per il lato frontend della nostra applicazione;

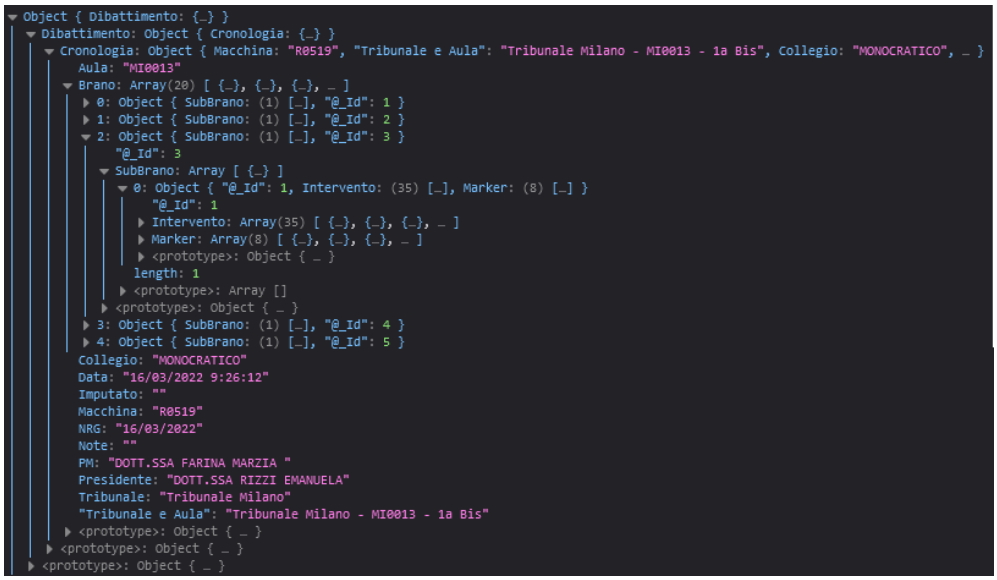


Figura 5.4: L'oggetto javascript risultato dell'interpretazione da parte della libreria del file cronologia

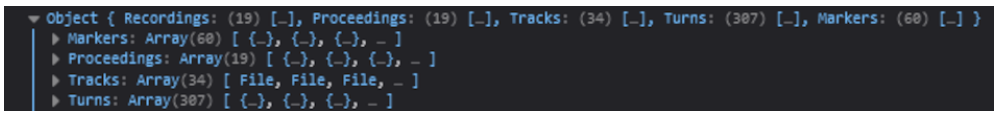


Figura 5.5: Gli array di oggetti javascript risultato della trasformazione da parte della libreria dell'oggetto in figura 5.4

Difficilmente la libreria sarà riutilizzabile perchè la natura dei dati di input è molto particolare però la sua realizzazione con varie funzioni suddivise in due parti ben distinte è stata appositamente pensata, per permetterci eventualmente di utilizzare i dati in uscita dalla prima parte dell'interpretazione del file, anche su un' altra piattaforma che non sia necessariamente la nostra applicazione web scritta in javascript.

5.1.3 Redux

Store

Lo store è l'oggetto che contiene tutto lo stato dell'applicazione, viene creato e gestito con Redux e Redux-toolkit ed è completamente separato dalla user interface. Nel nostro caso questo oggetto è molto complesso, perchè i dati possono diventare numericamente molto grandi in modo veloce e perchè la loro rappresentazione in alcuni casi può essere non serializzata, cioè dati non standard, come Redux prevede. Per ovviare a questo inconveniente che è stato riscontrato per tutte le operazioni che coinvolgono i dati di tipo File, centrali nello sviluppo del nostro frontend, abbiamo sfruttato la possibilità che da Redux di disabilitare il controllo sulla serializzazione per alcuni dati. Con le funzioni che Redux mette a disposizione la creazione e la manutenzione dello store risulta molto facile, una volta che è stato compreso come questo si comporta. Lo store rappresenta per la nostra applicazione lato frontend il punto dove vengono mantenuti

tutti i dati e le operazioni che possono essere eseguite su questi. Lo store, infatti, non rappresenta solo un modello di dati ma un vero e proprio stato dell'applicazione, cioè un insieme di dati e azioni che in un dato momento è possibile compiere. Per facilitare l'implementazione ma soprattutto la manutenzione del codice lo store può essere diviso in **slice**, ovvero fette di dati, nel nostro caso abbiamo suddiviso lo store in due slice principali, una riguardante i dati caricati dall'utente e un altra con le operazioni che coinvolgono chiamate alle API. La caratteristica principale di uno store Redux e di come questo strumento gestisca lo stato dell'applicazione è che i dati all'interno dello store sono immutabili, cioè non possono essere cambiati o sovrascritti direttamente ma bisogna sempre passare per le apposite funzioni chiamate **action** che Redux permette di creare per implementare il modo di cambiare lo stato. Queste action sono la regola per gestire lo stato con Redux perchè lavorando in questo modo si permette a questo strumento di avere sempre il controllo dello stato ad ogni modifica e in particolare di ricalcolare lo stato solo nella misura in cui una action prevede. Questo nuovo stato viene poi aggiornato automaticamente nei componenti React dopo il ricalcolo. Tutte le nostre action sono create all'interno di particolari funzioni della libreria Redux chiamate **reducer** che ricevono come parametri lo stato e l'azione e permettono il calcolo del nuovo stato. Ogni reducer può essere richiamato all'interno dell'applicazione con il metodo Redux **dispatch** che scatena l'azione di riferimento sullo stato attuale. Infine per rendere disponibili i dati presenti nello store si possono creare dei **selector** che selezionano porzioni di dati dallo store e le rendono disponibili nei componenti React.

Creazione dello Store

Lo store implementato con Redux, viene creato in un apposito file javascript all'esterno dell'applicazione React, in modo da mantenere la separazione, e viene passato all'applicazione in un unico punto, che per convenzione è il file index.jsx dove tutta l'applicazione risiede. Lo store viene creato con un particolare metodo di Redux che ha la funzione di combinare tutte le slice che abbiamo creato ma non solo, permette di definire anche casi particolari, infatti è proprio in questa fase di creazione dello store che abbiamo "richiesto" a Redux di non verificare la serializzazione di alcuni dati, come spiegato in precedenza.

```
import { combineReducers, configureStore } from '@reduxjs/toolkit';
import config from './config';
import apiSlice from '../features/api/rest/apiSlice';
import trialSlice from '../features/trialSlice';

const rootReducer = combineReducers({
  [apiSlice.reducerPath]: apiSlice.reducer,
  [trialSlice.name]: trialSlice.reducer,
});

const store = configureStore({
  reducer: rootReducer,
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware({
      serializableCheck: {
        // Ignore these field paths in all actions. Default reported: 'meta.arg', 'meta.baseQueryMeta'
        ignoredActionPaths: ['meta.arg', 'meta.baseQueryMeta', 'payload.Dibattimento.Cronologia.Files'],
        // Ignore these paths in the state
        ignoredPaths: ['trial.Files'],
      },
    }).concat([apiSlice.middleware]),
  devTools: config.debug,
});

export default store;

```

Figura 5.6: Creazione dello store dell'applicazione

Slice e Reducer

Una slice contiene una parte del nostro store, nel caso della nostra apllicazione si è deciso di dividere lo store in due slice una per la parte dove vengono salvati i dati da visualizzare e modificare all'interno dell'applicazione (procedimenti, registrazioni e tracce) e l'altra per la parte che gestisce le richieste tramite API al backend, per questa seconda slice è necessario l'utilizzo della libreria Redux-toolkit che ha una serie di funzioni che permette di facilitare le richieste, e di integrare le risposte direttamente nello store. Ogni slice ha i propri reducer, ovvero le funzioni che prendendo in input lo stato attuale e una action e restituiscono il nuovo stato modificato secondo quello che l'azione prevede. Nel nostro caso per mantenere più ordinata possibile la gestione dello stato ogni slice presenta il proprio reducer che contiene le varie action.

```
1   import { createSlice } from '@reduxjs/toolkit';
2
3   const initialState = {
4     Brano: [],
5     Proceedings: [],
6     Files: [],
7   };
8   function nextId(array) {
9     let maxId = 0;
10    array.map((elem) => {
11      maxId = elem['@_Id'] > maxId ? elem['@_Id'] : maxId;
12      return null;
13    });
14    return maxId + 1;
15  }
16  const trialSlice = createSlice({ ···
124 });
125
```

Figura 5.7: Slice che rappresenta i nostri dati all'interno dell'applicazione

Action

Le action modificano lo stato dell'applicazione senza farlo direttamente ma demandando la vera e propria modifica dei dati a Redux. I dati come accennato in precedenza sono immutabili quando si implementa uno store con Redux per la gestione dello stato di un applicazione, questo significa che operazioni come arrayDiDati[chiave] = valore non sono possibili direttamente dal codice. Per mantenere lo stato logicamente corretto, efficente e valido i dati vanno modificati creando delle action, che prevedono il passaggio dello stato attuale e dei paramentri aggiuntivi come input e restituiscono un nuovo stato dell'applicazione.

```
4   const apiProceedingSlice = apiSlice.injectEndpoints({
5     endpoints: (builder) => ({
6       getProceeding: builder.query({
7         query: (id) => `file_processing_job/${id}`,
8       }),
9       getProceedings: builder.query({
10        query: (params) => ({
11          url: `file_processing_job/`,
12          params,
13        }),
14      }),
```

Figura 5.8: Actions con chiamate al backend

Selector

Per reperire una porzione di dati come per esempio un array di registrazioni o un singolo procedimento all'interno del nostro store abbiamo bisogno dei selector, questi sono delle piccole funzioni che fanno un'azione di filtro sull'intero store restituendo solo quello che abbiamo richiesto. La vera funzione di un selector oltre a filtrare lo store è quella di mantenere questo filtro attivo sempre all'interno dell'applicazione finchè non cambia lo stato, infatti i dati selezionati con un selector, vengono ricalcolati soltanto se una modifica allo store (tramite action) coinvolge almeno uno dei dati stessi.

```
128
129  // Proceeding list
130  export const getProceedingsList = (state) => state.Proceedings;
131
132  // Proceeding by Id
133  export const getProceedingById = (state, proceedingId) => getProceedingsList(state).find((proceeding) => proceeding['@_Id'] === proceedingId);
134
```

Figura 5.9: Selectors per selezionare i procedimenti dallo store

5.1.4 React

Component

La scelta di react in fase di analisi del progetto ci permette di utilizzare i component tipici di questa libreria javascript. Il concetto di component è molto intuitivo quando ci si approccia ad usare React, essi sono come delle funzioni javascript che accettano in input dei particolari parametri chiamati **props**, e ritornano elementi React che descrivono cosa si dovrà vedere sullo schermo. Usando i components possiamo facilmente dividere la user interface in piccole parti indipendenti tra loro e riusabili, in modo da doverci occupare di una parte alla volta pensando solo a quella in modo isolato prima di integrarla con il resto della UI. I component secondo React possono essere di due tipi class component o function component, nel nostro caso abbiamo scelto di costruire tutti function component principalmente per avere un codice più leggibile e perchè non c'è la necessità di avere i vantaggi dei class component (in particolare non ci serve nessun metodo costruttore all'interno dei component che abbiamo creato). I component possono quindi essere una qualsiasi parte della user interface che vogliamo realizzare, da un bottone a un form oppure un intera vista. Sviluppando la nostra applicazione abbiamo pensato inizialmente ai component più "esterni", cioè quelli che fanno da contenitore agli altri, andando poi a raffinare sempre di più fino ai component più "interni", ossia quelli riguardanti le parti più piccole dell'applicazione, usando in questo modo una sorta di metodo top-down per lo sviluppo delle componenti necessarie. Una vola che abbiamo creato i vari component, utilizzando questo modello top-down, siamo passati all'integrazione. React permette molto agevolmente, proprio come punto di forza nell'uso di questa libreria, l'integrazione tra i vari componenti. Basta importare nella pagina il componente che ci serve, richiamarlo nel codice come se fosse un normale tag HTML e passargli le props di cui ha bisogno, non dobbiamo preoccuparci di altro, React svolgerà per noi il corretto rendering dell'insieme dei componenti.

```jsx
import { forwardRef } from 'react';
import { Stack } from '@mui/material';
import { useSelector } from 'react-redux';
import { getFilesList } from '../features/trialSlice';
import AudioPlayer from './AudioPlayer';
import ProceedingTabs from './ProceedingTabs';
import ProceedingList from './ProceedingList';

const FileView = forwardRef((props, ref) => {
  const Files = useSelector((state) => getFilesList(state.trial));

  return (
    <Stack component="section" id="FileView" direction="column" spacing={7}>
      {Files?.map((file) => {
        return (
          <Stack key={file.name} component="section" direction="column" spacing={1}>
            <AudioPlayer file={file} />
            <ProceedingTabs subbranoId={file.SubBranoId} branoId={file.BranoId} />
            <ProceedingList subbranoId={file.SubBranoId} branoId={file.BranoId} file={file} ref={ref} />
          </Stack>
        );
      })}
    </Stack>
  );
});

export default FileView;
```

Figura 5.10: Codice del component fileView per la vista dei file caricati

Vediamo nel dettaglio come viene reralizzato un component nella nostra applicazione, nello specifico il component fileView, che realizza un intera vista occupandosi di visualizzare la lista di tutti i file dopo che sono stati caricati dall'utente. Questo componente non necessita di alcuna props ma si serve di uno dei selector Redux che abbiamo creato per reperire la lista dei file di cui abbiamo bisogno dallo store. (riga 10) La lista dei file recuperata grazie all'apposito selector viene snocciolata con la funzione nativa javascript map (riga 14), che permette di eseguire delle operazioni per ogni oggetto della lista, nel nostro caso per ogni file. In questo component per ogni file si desidera richiamare altri tre component che si occuperanno di alcune parti della user interface che riguarda ogni file, e sono:

* **AudioPlayer**: che provvede a fare il render di un player per riprodurre il file;
* **ProceedingTabs**: tramite una tendina permette di visualizzare e gestire i metadati relativi al file (Marker e Interventi);
* **ProceedingList**: recupera e visualizza tutti i procedimenti relativi a questo file.

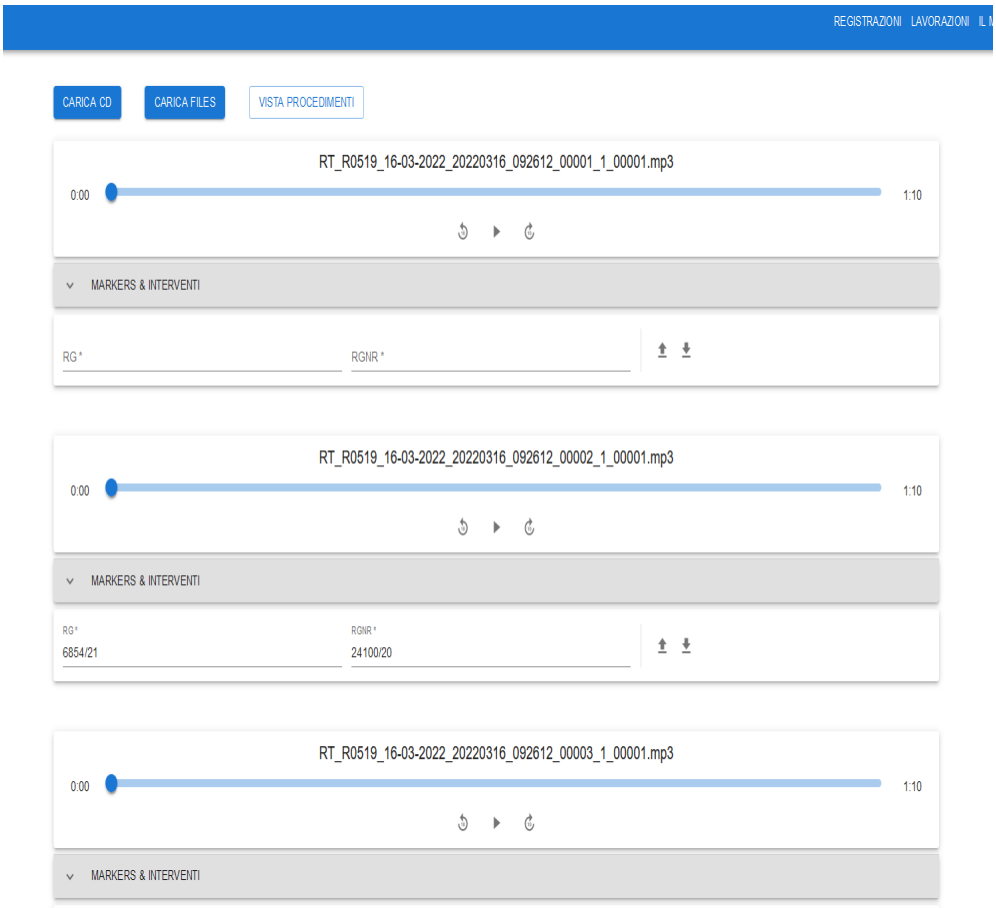


Figura 5.11: Render grafico del component fileView per la vista dei file caricati

Hooks

Gli Hook sono delle funzioni javascript che permettono di entrare nel ciclo usato da React per renderizzare i componenti, e in questo modo descriverne il comportamento. Queste funzioni non possono essere usate nei componenti di tipo classe e hanno delle regole di invocazione abbastanza rigide, per proteggere lo stato dell'applicazione e il ciclo di rendering. Gli hook infatti non possono essere chiamati all'interno di cicli, istruzioni condizionali o funzioni che non siano componenti react. La libreria React mette a disposizione degli Hook generici che risultano molto utili per gestire il rendering dei componenti, ma se questi non dovessero bastare possiamo anche crearne di personalizzati. Di seguito vediamo un hook appositamente creato per recuperare dal backend la lista dei jobs, è stato creato per essere riusato qualora si rendesse necessario da un altra parte dell'applicazione ma anche perchè sfruttando le proprietà del linguaggio e in particolare di queste speciali funzioni possiamo facilmente gestire il filtraggio e la paginazione della lista dei jobs.

```
import { useSearchParams } from 'react-router-dom';
import { useGetProceedingsQuery } from '../features/api/rest/apiProceedingSlice';

export default function useJobsResults() {
  const [search] = useSearchParams();
  const params = {};

  for (const [key, value] of search.entries()) params[key] = value;

  const { currentData, isFetching, isSuccess, isError } = useGetProceedingsQuery(params);

  return {
    jobs: currentData?.results,
    jobsCount: currentData?.count,
    isFetching,
    isSuccess,
    isError,
    params,
  };
}

```

Figura 5.12: Codice del hook creato per la visualizzazione dei procedimenti caricati

5.1.5 MUI

UI style

Lo sviluppo della parte grafica, in particolare per quanto riguarda lo stile dell'applicazione, non aveva nessun vincolo, si è scelta una soluzione che possa essere intuitiva, veloce da implementare e generalmente riconosciuta. La scelta fatta è stata l'utilizzo di **MUI - Material UI** che è una libreria open-source di componenti React già precostruiti seguendo le regolole di Google Material Design. Material è un sistema di design creato da Google per velocizzare lo sviluppo dei componenti. MUI permette di non doversi preoccupare di aspetti chiave quali responsive-design, integrazione grafica dei nuovi componenti, stile generale dell'applicazione, perchè tutti questi aspetti vengono automatizzati usando stili globali e appunto componenti già preimpostati, semplicemente da configurare in base alle necessità.

```
76  return (
77    <Stack margin={2} spacing={2} direction="row" alignItems="center" component="form" onSubmit={handleSubmitProceeding}>
78      <FormControl sx={{ width: 2 / 6 }}>
79        <TextField required onChange={(e) => dispatch(changeRG({ ... }))} value={Proceeding.RG} label="RG" variant="standard" />
80      </FormControl>
81      <FormControl sx={{ width: 2 / 6 }}>
82        <TextField required onChange={(e) => dispatch(changeRGNR({ ... }))} value={Proceeding.RGNR} label="RGNR" variant="standard" />
83      </FormControl>
84      <Divider orientation="vertical" flexItem />
85      <Typography sx={{ width: 2 / 6 }}>
86        <IconButton type="submit" variant="contained">
87          <UploadIcon />
88        </IconButton>
89        <IconButton variant="contained" onClick={handleAudioProceeding}>
90          <DownloadIcon />
91        </IconButton>
92      </Typography>
93    </Stack>
```

Figura 5.13: Codice del componente per visualizzare un procedimento

In questo esempio preso direttamente da uno dei componenti della nostra applicazione, possiamo vedere vari MUI component utilizzati e personalizzati appositamente per il nostro funzionamento. Tra questi c'è per esempio **<Stack>** che costruisce una pila di sottocomponenti, che possono essere direzionati in colonna o in riga (come nel nostro caso). Altri utilizzi sono **<FormControl>** e **<TextField>** che costruiscono una porzione di un form con un campo di tipo testo al suo interno.

Figura 5.14: Render grafico del componente per visualizzare un procedimento

5.2 Testing

Parte importante del nostro processo di sviluppo, sono i test. Secondo regola aziendale ma anche seguendo il piano di progetto dello stage è stata pianificata una buona parte di ore di testing. Le ore di testing sono state ripartite in modo proporzionale alle ore di sviluppo di ogni funzionalità. Nella fase iniziale di studio delle tecnologie da utilizzare per il progetto sono state incluse anche varie librerie per i test su funzioni javascript e componenti React che sono la grande parte del prodotto da realizzare. Dopo un confronto con il tutor si è deciso di intraprendere la strada di **React-testing-library**[1] per il testing dei componenti e appoggiarci a **jest**[2] per la parte javascript dei test.

5.2.1 Test javascript function

Il framework jest è uno dei più diffusi per il test di applicazioni javascript, è focalizzato sulla semplicità nella scrittura dei test e risulta comunque in grado di testare tutte le parti di nostro interesse. All'interno dei nostri test è stato usato in supporto a React-testing-library per il testing dei componenti React e in prima battuta per il test di tutte le utility javascript che abbiamo creato, in particolare si è reso molto importante nel testing delle funzioni che leggono i metadati caricati dall'utente e li manipolano per renderli usabili nell'applicazione. Il funzionamento di jest è molto semplice per i nostri casi, ma può essere esteso a molti altri tipi di test. Per noi è semplice e molto intuitivo creare un test javascript con questo framework, basta creare le condizioni per l'utilizzo di una determinata funzione che vogliamo testare, chimarla come se dovessimo realmente utilizzarla e vedere se i risultati in usicta da questa chiamata sono quelli che ci aspettiamo.

```
3  describe('parseFileName', () => {
4    it('should parse a file name', () => {
5      const fileName = 'RT_R0519_16-03-2022_20220316_092612_00001_1_00001.mp3';
6      const parsed = parseFileName(fileName);
7      // Assert
8      expect(parsed).toEqual({
9        Brano: 1,
10       SubBrano: 1,
11       Traccia: 1,
12     });
13   });
14 });
15
```

Figura 5.15: Codice del test per una funzione javascript

Nella prima parte del test come si può vedere abbiamo creato i dati fittizzi (fileName riga 5) che ci servono da passare alla funzione da testare (in questo caso parseFileName riga 6), in seguito abbiamo effettuato la chiamata di funzione e per concludere il test abbiamo scritto delle particolari funzioni (expect, toEqual riga 9) con le quali andare a vedere se i risultati ottenuti sono quelli che ci aspettiamo passando i nostri dati fittizzi.

[1] *React Testing Library*. URL: https://testing-library.com/docs/react-testing-library/intro/.

[2] *jest*. URL: https://jestjs.io/.

5.2.2 Test React component

La libreria usata è React-testing-library, questa si appoggia completamente su un altra libreria di testing chiamata DOM-testing-library e fornisce una abbastanza completa suite di testing per i componenti React. Il principio che guida questa libreria e anche i nostri test sui componenti è che i test di integrazione hanno maggior valore degli unit test nel caso di applicazioni React. In particolare con queste librerie (React-testing e jest) usate in modo combinato si creano test su parti di applicazione dopo il rendering, ovvero la logica dei nostri test è quella di andare a verificare funzionamenti e presenza di elementi nel DOM dopo che è stato fatto il rendering dell'applicazione. Questo tipo di test ovviamente è un test di integrazione in quanto non va a testare singolarmente ogni componente e ogni funzionalità del componente ma esegue un test della UI dal punto di vista dell'utente, cioè controlliamo che l'applicazione in ogni momento si comporti come abbiamo progettato che lo faccia. La libreria prevede di descrivere un test passando ad un apposito metodo il componente di cui fare il render, e una volta che questo è stato completato con altri metodi specifici si va a controllare cosa è nel DOM. Altre situazioni possono esssere sottoposte a test in questo modo, come scatenare in modo controllato un evento (per esempio il click su un bottone) e vedere se il DOM si modifica come ci si aspetta. Ci sono molte altre possibilità per casi più particolari come per testare chiamate ad API, nella nostra applicazione però questo tipo di test non è stato implementato per motivi di tempo e in accordo con l'azienda che prevede di mettere appunto una specifica suite di test per questo ambito per vari prodotti che segue.

```
21  it('it should render Trial blank page', () => {
22    // Arrange
23    render(
24      <Provider store={store}>
25        <Router>
26          <Routes>
27            <Route path="/" element={<Trial />} />
28          </Routes>
29        </Router>
30      </Provider>
31    );
32    // Assertions
33    expect(screen.getByText('recordings.loadCd')).toBeInTheDocument();
34  });
35
```

Figura 5.16: Codice del test per un componente React

Nel nostro piccolo esempio che abbiamo riportato qui di test di un componente si può vedere come sia facile e veloce implementare un test di questo tipo, ma allo stesso tempo come verifichi esattamente quello che ci aspettiamo dalla nostra applicazione. In questo caso dopo il rendering del nostro componente testato (Trial) andiamo a controllare se nel DOM sono presenti gli elementi che ci aspettiamo, in questo caso un semplice bottone con testo preso dal file delle traduzioni (recordings.loadCD).

Capitolo 6

Conclusioni

6.1 Consuntivo finale

Sett	Da - A	Attività
1	dal 17/10/2022 al 23/11/2022	Studio tecnologie
2	dal 24/10/2022 al 30/11/2022	Studio tecnologie e inizio sviluppo prototipo applicazione
3	dal 31/10/2022 al 06/11/2022	Studio tecnologie e sviluppo prototipo applicazione
4	dal 07/11/2022 al 13/11/2022	Sviluppo dell'interfaccia di login
5	dal 14/11/2022 al 20/11/2022	Sviluppo dell'interfaccia di login e relativi test
6	dal 21/11/2022 al 27/11/2022	Implementazione lettura dati da CD
7	dal 28/11/2022 al 04/12/2022	Implementazione lettura dati da CD e relativi test
8	dal 05/12/2022 al 11/12/2022	Implementazione editing dei dati e relativi test
9	dal 12/12/2022 al 18/12/2022	Sviluppo react audio player component
10	dal 19/12/2022 al 25/12/2022	Implementazione upload dei dati
11	dal 26/12/2022 al 01/01/2023	Implementazione upload dei dati e relativi test
12	dal 02/01/2023 al 08/01/2023	Inizio creazione dei ticket e studio sistemi di modifica dati tramite interazione evolute

Tabella 6.1: Tabella del consuntivo orario dello stage

6.2 Raggiungimento degli obiettivi

Codice	Descrizione	Stato
O01	autenticazione mediante server remoto	soddisfatto
O02	lettura dati da CD	soddisfatto
O03	precompilazione di form con i dati caricati da CD	soddisfatto
O04	editing dei dati del form	soddisfatto
D01	upload dei dati verso i sistemi esterni	soddisfatto
D02	test di unità esaustivi	soddisfatto
F01	possibilità di ascoltare le registrazioni	soddisfatto
F02	possibilità di modificare i dati mediante interazioni evolute (per es. drag-n-drop)	non soddisfatto
F03	realizzazione di un'applicazione desktop con Electron	non soddisfatto
F04	compilazione multipiattaforma dell'applicazione desktop	non soddisfatto

Tabella 6.2: Tabella del tracciamento della soddisfazione dei requisti dello stage

6.3 Valutazione obiettivi

Valutando lo stage sulla base degli obiettivi che erano stati prefissati in sede di pianificazione posso dire che è stato svolto un buon lavoro e che gran parte di quelli che erano stati individuati come più facili da raggiungere sono stati soddisfatti. In particolare gli obiettivi obbligatori (O01, O02, O03, O04) sono stati soddisfatti agevolmente nel corso della prima parte dello stage in seguito alla fase di studio delle tecnologie. Per quanto riguarda invece gli altri obiettivi soddisfatti, che si dividono tra desiderabili (D01, D02) e facoltativi (F01), sono stati svolti prevalentemente nella seconda parte dell'esperienza di lavoro, anche se va fatta una menzione particolare per D02 che trattava i test di unità, che ha coinvolto l'intera durata dello stage, dalla fase di studio alla fine dell'implementazione dell'ultimo requisito sviluppato. Per quanto riguarda gli obiettivi dello stage che non sono stati soddisfatti (F02, F03, F04), va constatato che erano stati classificati come facoltativi in fase di pianificazione delle attività, questo perchè era stato previsto che potessero essere difficili da raggiungere per vari motivi. In particolare l'obiettivo F02, che riguarda la modifica di dati mediante drag-n-drop è stato studiato durante il periodo di stage in quanto era una delle attività che poteva essere sviluppata nel corso delle ultime settimane, però oltre alla fase di studio delle tecnologie da utlizzare e qualche prova non sono riuscito a concludere molto di più perchè è arrivata la fine dello stage, probabilmente era un attività che necessitava di essere pianificata prima nel percorso di lavoro. Gli altri due obiettivi non soddisfatti

(F03, F04) durante lo stage invece non sono mai stati pianificati tra le attività da svolgere, e in questo caso i motivi sono molteplici e non riguardano solo le tempistiche dello stage. La prima cosa che va osservata per questi due obiettivi è che sono sequenziali nella loro realizzazione, in particolare l'idea pre-stage era quella di portare prima l'applicazione sul web, poi su desktop tramite Electron e infine su altri sistemi desktop, non avendo completato la creazione dell'applicazione web gli altri due obiettivi non potevano ovviamente essere realizzati. Posso quindi dire che l'inserimento di questi due obiettivi nel piano di lavoro è stata un po troppo pretenziosa, nel senso che sono stati giustamente classificati come facoltativi però dopo aver lavorato su questo progetto ho capito che era abbastanza impensabile poter realizzare un'applicazione web, per quanto minima, nelle tempistiche dello stage, e poi cercare di spostarla su desktop. Quello che a posteriori posso dire che ha portato via la maggior parte del tempo e quindi allungato tutte le tempistiche dello stage è stato lo studio del corretto modo di trattare i dati e metadati che sono coinvolti nell'applicazione, la natura di questi dati e la loro varietà ha protratto le fasi di studio e di progettazione dell'applicazione molto più avanti nel tempo di quanto ci si poteva aspettare e questo di conseguenza ha ristretto le tempistiche per lo svolgimento degli obiettivi facoltativi.

6.4 Prodotto ottenuto

Alla fine dell'esperienza di stage il prodotto, come già accennato nella parte introduttiva della relazione, si discosta leggermente da quello che sarà il prodotto finale da mettere in produzione. Si è deciso di realizzare le funzionalità che erano oggetto dello stage in prima battuta e svolgere in seconda fase tutto quello che riguarda l'integrazione tra le varie parti del frontend, in particolare la creazione del ticket che alla fine dell'esperienza risulta non ancora completata. Il prodotto non è ancora in produzione, ma è in un buono stato di avanzamento, a giudizio dell'azienda ospitante lo stage, e guardando al futuro si prevede che con qualche altro mese di lavoro, potrà passare alla fase di collaudo.

6.5 Conoscenze acquisite

Questa esperienza di stage ha avuto grande valore per quanto riguarda l'acquisizione delle conoscenze tecniche e tecnologiche, infatti grazie all'aiuto in particolare del tutor ma anche di altri sviluppatori del team dell'azienda ospitante lo stage, posso dire di aver raggiunto una buona conoscenza degli strumenti utilizzati ma anche a lato pratico delle tecnologie impiegate per lo sviluppo e i test. Guardando nel particolare, per qunato riguarda l'organizzazione aziendale e la gestione del progetto, ho acquisito una conoscenza approfondita nell' utilizzo del framework SCRUM. Inoltre sempre per quanto riguarda la gestione del codice, un altro personale grande passo in avanti è stato fatto nella comprensione approfondita di git nella maggior parte delle sue funzionalità, in modo specifico per quanto riguarda il workflow utilizzato dall'azienda, il feature branch. A lato sviluppo e testing ovviamente l'apprendimento è risultato più difficile perchè le tecnologie utilizzate erano completamente nuove in particolare Redux ma anche React, solo per citare le principali, ma in una valutazione retrospettiva posso dire che il migliramento, nella comprensione dei paradigmi di sviluppo, e delle buone norme di codifica per le tecnologie utilizzate, è stato molto soddisfacente.

6.6 Valutazione finale prodotto

Il prodotto finale come detto in precedenza non risulta ancora utilizzato, ma essendo un progetto reale che l'azienda porterà avanti per mettere in seguito in produzione, continuerà ad essere sviluppato per raggiungere quelli che sono gli obiettivi che l'azienda ha posto nell'analisi iniziale. Alla fine dell'esperienza di stage da parte mia sicuramente resta una piccola parte di insoddisfazione per non aver raggiunto anche gli obiettivi posti come facoltativi che l'azienda porterà a termine nel prossimo futuro. La principale problematica riscontrata alla fine dell'esperienza penso di poter dire che è la non conoscenza delle tecnologie utilizzate per lo sviluppo, nonostante un iniziale periodo di studio e prova, prima di iniziare a sviluppare il prodotto vero e proprio, quando sono arrivato alla fine dello stage la mia conoscenza teorica e pratica è molto migliorata e questo mi fa guardare alle prime fasi di codifica con insoddisfazione perchè con le conoscenze ora acquisite sicuramente queste parti si potevano svolgere in modo migliore, ecco perchè prima di ultimare lo stage parlando con il tutor ho proposto varie soluzioni per migliorare la codifica del prodotto, per renderlo più solido, consistente, efficiente, efficace e manutenibile. Tutto questo costerà tempo all'azienda ma probabilmente sarà del tempo che in futuro verrà risparmiato per eventuali feature o manutenzioni da fare. Un altro punto su cui si è discusso con il referente aziendale è quello delle funzionalità del prodotto, perchè come è normale che sia, in corso di sviluppo le funzinalità inizialmente previste vengono riviste e rivalutate. Per il nostro prodotto tutto quello che era stato pensato in fase di analisi è stato tenuto come funzionalità da preservare anche per il futuro del prodotto, ma qualche aggiunta è stata pensata per renderlo più veloce e pratico da utilizzare per i fonici, visto che l'obittivo del prodotto è proprio quello di facilitare il loro lavoro.

6.7 Valutazione finale stage

L'esperienza di stage è stata assolutamente positiva, mi ha dato modo di vedere a lato pratico molte delle cose che avevo visto in modo teorico nel corso degli studi. Ho effettuato lo stage completamente da remoto, in modalità smart-working, e anche da questo punto di vista sono rimasto molto soddisfatto perchè ho potuto apprezzare il potenziale di questa modalità di lavoro, che secondo la mia opinione, spinge maggiormente lo studente in stage a rendersi autonomo dal punto di vista lavorativo, ne migliora le capacità organizzative e consente la flessibilità rispetto agli orari. In un modo di lavoro dove si guarda più agli obiettivi da raggiungere che al tempo effettivo di lavoro si incorre nel rischio di stimare male le tempistiche per le attività da svolgere ma allo stesso tempo si rende maggiormente responsabile il lavoratore, e nel lungo periodo si valutano in modo più oggettivo le competenze e le capacità, è una scelta che continuerei a fare anche nel mio futuro lavorativo. Un grande aiuto l'ho ricevuto dal mio tutor e da altri sviluppatori che con grande pazienza e conoscenze degli argomenti molto approfondite mi hanno guidato in questo stage, senza mai lasciarmi ad affrontare i problemi del progetto da solo ma rimanendo sempre a disposizione per qualsiasi mio dubbio, e mi ritengo molto soddisfatto anche per questo dell'azienda scelta per questa esperienza. L'unica vera difficoltà che ho incontrato in questo stage è la poca conoscenza iniziale delle tecnologie, in particolare di React e ma in parte anche di git, che mi hanno costretto ad un inizio un po lento fatto più che altro di studio e approfondimento delle conoscenze, questo ha comportato un rallentamento di tutte le attività del progetto, andando un po ad influire sulle tempistiche e il prodotto finale.

Acronimi e abbreviazioni

API Application Program Interface. 33

DOM Document Object Model. 32

HTML HyperText Markup Language. 32

IDE Integrated Development Enviroment. 5

JSON JavaScript Object Notation. 35

MVC Model View Controller. 33

MVVM Model-View-ViewModel. 37

REST REpresentational State Transfer. 33

UI User Interface. 32

UML Unified Modeling Language. 13

URL Uniform Resource Locator. 32

Glossario

array di oggetti sono delle collezioni non ordinate di oggetti javascript, sulle quali possiamo eseguire operazioni di vario tipo, aggiungere/rimuovere/modificare dati per esempio. 45

backend nell'architettura di un applicazione si intende la parte di software con la quale l'utente non interagisce direttamente, che generalmente gestisce l'archiviazione dei dati e la logica di busness. 2

brano nelle tracce audio del nostro sistema di registrazione, è la traccia più esterna che può contenere vari sottobrani. Possono essercene più di uno per CD.. 42

byte è generalmente una sequenza di 8 bit. Con i multipli del byte si identifica l'unità di misura della capacità di memoria. 2

canali nell'aula di tribunale, corrisponde alla registrazione di uno dei microfoni disponibili in aula nei quali uno dei presenti parla. Ad ogni canale audio corrisponde una traccia.. 42

endpoint nella comunicazione frontend-backend tramite API è il punto che il server espone per ricevere richieste ed inviare risposte. 33

form è la parte di interfaccia utente di un applicazione che consente all'utente l'iserimento di dati. 3

framework è l'architettura logica sulla quale il software viene progettato e realizzato, utilizzato per facilitare lo sviluppo e il mantenimento del codice. 6

frontend nell'architettura di un applicazione si intende la parte di software visibile all'utente. 2

jobs nella nostra applicazione con questo termine si intende i procedimenti per i quali è già stato fatto l'upload verso il backend. 26

libreria è un insieme di funzioni e strutture dati non direttamente eseguibili utilizzati a supporto di un software. 3

metadati sono le informazioni che permettono la costruzione dei dati. 1

object storage è un architettura per la memorizzazione dei dati, dove questi vengono gestiti come oggetti. 2

oggetti javascript sono delle strutture dati che possiamo definire all'interno del nostro codice.. 45

open-source indica una particolare la licenza di un software, che consente la libera distribuzione la modifica e lo studio. 32

parsing significa letteralmente analisi, nel nostro contesto si riferisce all'analisi e all'interpretazione che la libreria creata fa sui metadati che si trovano nel file cronologia. 44

plugin è un programma a supporto di un altro che ne estende le funzionalità. 5

repository è un sistema di archiviazione di file e dati utilizzato per gestire il codice di un progetto. 5

routing indica la parte di gestione dell'indirizzamento delle URL e dei percorsi (path) in un applicazione web. 32

sessione nelle applicazioni web, indica l'attività svolta dall'utente da quando si collega all'applicazione tramite browser a quando il collegamento viene chiuso. 34

sottobrano nelle tracce audio del nostro sistema di registrazione, è la traccia più interna che è figlia di un brano. Possono essercene più di una per ogni brano.. 42

start-up è un'azienda che è nella fase iniziale di avvio delle attività. 1

stato è l'insieme di input di dati e azioni che determinano l'output in un dato momento.. 32

task in un sistema di organizzazione del lavoro è una specifica attività che fa parte di un progetto. 4

traccia mixer nel nostro sistema di registrazione, è la traccia risultato del mixaggio delle tracce corrispondendi ai singoli canali. In particolare ci sono 7 microfoni in aula, ognuno viene registrato su una traccia audio, la traccia mixer è quella che registra tutto l'insieme.. 42

workflow nella gestione dei progetti è il flusso di lavoro che un attività deve seguire dalla sua creazione al suo completamento. 5

Bibliografia

Siti web consultati

Jira. URL: https://www.atlassian.com/it/software/jira (cit. a p. 4).

Git. URL: https://www.atlassian.com/it/git (cit. a p. 5).

Gitlab. URL: https://about.gitlab.com/ (cit. a p. 5).

SCRUM. URL: https://www.atlassian.com/it/agile/scrum (cit. a p. 6).

React. URL: https://reactjs.org/ (cit. a p. 32).

Redux. URL: https://redux.js.org/ (cit. a p. 32).

MUI. URL: https://mui.com/ (cit. a p. 32).

React Testing Library. URL: https://testing-library.com/docs/react-testing-library/intro/ (cit. a p. 55).

jest. URL: https://jestjs.io/ (cit. a p. 55).