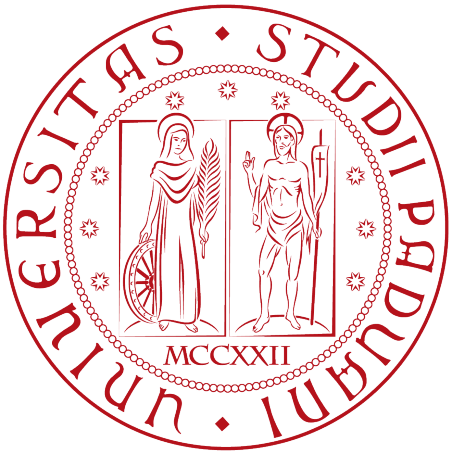
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Developer Portal Hub: una soluzione centralizzata per la gestione sicura ed intuitiva della documentazione API

Tesi di laurea

Relatrice

Prof.ssa Ombretta Gaggi

Laureando

Andrea Meneghello 2009084

Anno Accademico 2022-2023

Sommario

Il seguente documento ha lo scopo di descrivere in modo dettagliato il lavoro svolto durante il periodo di stage, dal laureando Andrea Meneghello, della durata di trecentododici ore, presso l'azienda THRON S.p.A. L'obiettivo principale del progetto di stage è stato realizzare un portale per favorire la consultazione delle *API* pubbliche e private di THRON, attraverso una soluzione centralizzata.
Il portale è stato sviluppato utilizzando il *framework Vue.js* accompagnato da vari strumenti del suo ecosistema, ed è stato protetto con autenticazione seguendo lo standard *Oauth2*, integrandosi con il *provider* aziendale *Azure*. Oltre alla consultazione della documentazione, il portale deve permettere all'utente di provare le *API* direttamente dall'interfaccia in modo intuitivo, permettendo inoltre il download delle stesse in formato *YAML*.
Infine tutte le componenti implementate sono state opportunatamente documentate e il loro corretto funzionamento è stato verificato tramite test di unità e di accettazione.

"Curiosity is the engine of intellect."

— Samuel Johnson

Ringraziamenti

Innanzitutto, desidero esprimere la mia profonda gratitudine alla Prof.ssa Ombretta Gaggi, relatrice della mia tesi, per la sua guida, sostegno e preziosi consigli durante la stesura di questa tesi. Senza il suo prezioso contributo, questo lavoro non sarebbe stato possibile.

Vorrei ringraziare di cuore i miei genitori e la mia famiglia per il loro amore, la loro comprensione e il loro costante supporto incondizionato. Senza di voi questo percorso non sarebbe stato possibile.

Alla mia fidanzata, desidero ringraziarti per la tua pazienza e per aver sempre creduto in me. Sei stata il mio sostegno morale che mi ha spinto a raggiungere questo traguardo.

Vorrei ringraziare tutti i miei amici per essere stati sempre al mio fianco, per avermi incoraggiato nei momenti difficili e per aver condiviso cone me questa avventura accademica.

Padova, Settembre 2023 Andrea Meneghello

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Il seguente capitolo vuole introdurre brevemente l'azienda con il relativo contesto aziendale e l'organizzazione del testo con le norme tipografiche adottate.

1.1 L'azienda

THRON S.p.A è un'azienda italiana con sede a Piazzola sul Brenta specializzata nello sviluppo di *SaaS* (*Software as a Service*). I suoi prodotti principali sono la *THRON DAM Platform* e il *THRON PIM* [24]. Il primo è una piattaforma nata con l'obiettivo di gestire, organizzare e distribuire i contenuti digitali di un'azienda [22]. Il *THRON PIM* invece è una soluzione per la gestione delle informazioni sui prodotti, che si concentra sulla raccolta, organizzazione e distribuzione delle informazioni relative ai prodotti di un'azienda [23].
In sintesi, il primo prodotto si concentra sulla gestione dei contenuti digitali, il secondo invece si focalizza sulla gestione delle informazioni sui prodotti. L'obiettivo è garantire una gestione centralizzata dei contenuti e semplificarne l'adattamento e la distribuzione su diversi canali in modo efficiente. All'interno dell'azienda, l'area *R&D* è suddivisa in due team: il team Prodotti, responsabile della gestione del [G]PIM, e il team Contenuti, focalizzato sulle tematiche legate al [G]DAM.
Durante il mio stage presso l'azienda, sono stato inserito come sviluppatore [G]frontend all'interno dell'area Prodotto, nel team Prodotti.

1.2 Metodologie di sviluppo

Nel contesto aziendale di THRON, vengono adottate metodologie di sviluppo [G]Agile per garantire un'efficace gestione dei progetti. La filosofia *Agile* pone un forte *focus* sulla collaborazione tra i membri del team, sulla capacità di rispondere in modo

rapido ai cambiamenti, nonché sull'orientamento al cliente e alla consegna di prodotti qualitativi [10].
Il [G]framework [G]Scrum parte integrante dell'approccio *Agile*, è utilizzato all'interno dell'azienda per organizzare il lavoro in iterazioni chiamate [G]Sprint della durata di due settimane ciascuna. Questo approccio a breve termine consente al team di concentrarsi su attività specifiche, pianificando e completando le attività in maniera incrementale. Al termine di ogni *Sprint* vengono organizzate molteplici riunioni: '*Sprint Review*', in cui il team presenta i risultati ottenuti durante lo *Sprint*, '*Sprint Planning*', in cui vengono pianificate le attività da svolgere durante lo *Sprint* successivo e '*Sprint Retrospective*', in cui il team riflette sulle prestazioni del team e sulle possibili migliorie. L'importanza della comunicazione e della condivisione delle informazioni è supportata dalla pratica quotidiana della riunione '*Daily Scrum*', durante la quale ogni membro del team esprime il proprio progresso, eventuali ostacoli e le prossime attività pianificate. Questa pratica aiuta a mantenere l'allineamento e la trasparenza, nonché a individuare tempestivamente eventuali problemi da risolvere [19].
Le riunioni settimanali chiamate '*Competence*' rappresentano un'altra componente importante adottata da THRON. Riunendo i membri dello sviluppo che condividono la stessa area di competenza, come frontend o [G]backend, queste riunioni offrono un'opportunità di scambio di conoscenze e di valutazione dei progressi.

1.3 Strumenti di sviluppo

Durante l'esperienza di stage presso THRON, l'utilizzo di una varietà di strumenti di sviluppo è stato fondamentale per assicurare un processo di lavoro efficiente.
Di seguito, sono elencati gli strumenti che sono stati impiegati per lo sviluppo del progetto:

- ***Teams***: per le comunicazione sia asincrone che sincrone con il team;
- ***Microsoft 365*** per la gestione della mail aziendale [11];
- ***Jira*** per il tracciamento e la gestione delle attività assegnate durante il periodo di stage (esempio in figura 1.1) [9];
- ***AWS*** per la gestione della [G]repository del mio progetto, per la [G]build e per il [G]deploy dello stesso [1];
- ***Fork*** per la gestione del versionamento del codice sorgente [7];
- ***Postman*** per la creazione, esecuzione e test delle chiamate verso gli [G]endpoint creati durante lo sviluppo [16];
- ***Confluence*** per la creazione e gestione della documentazione relativa al progetto di stage [6];

- ***StarUML*** per la modellazione dei casi d'uso [20].

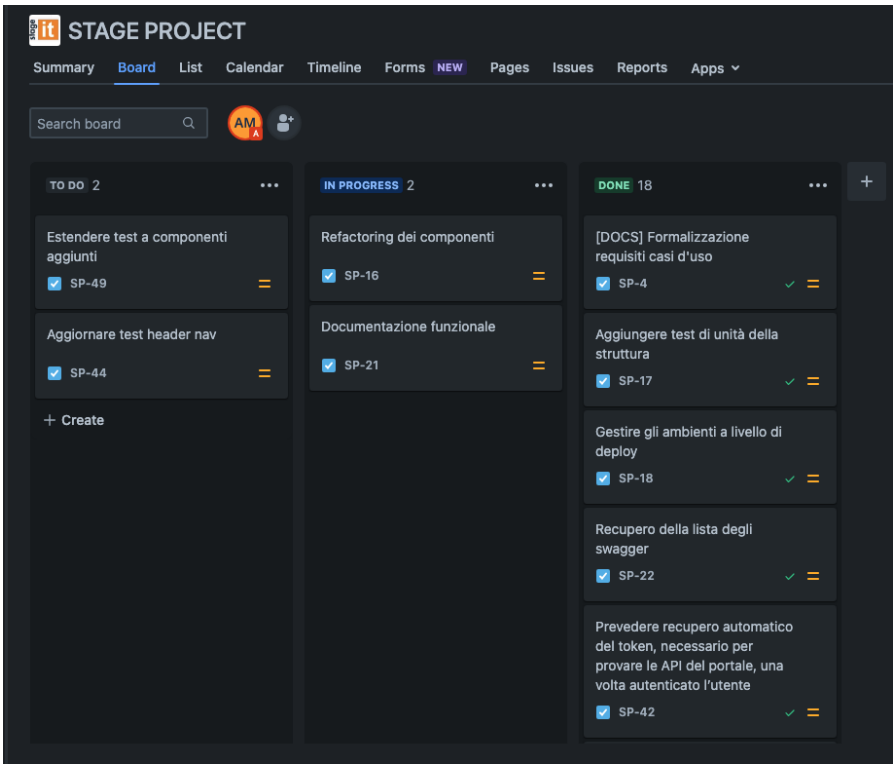


Figura 1.1: Board Jira

1.4 Organizzazione del testo

Il secondo capitolo descrive il progetto svolto durante il periodo di stage, mettendo in risalto gli obiettivi imposti dall'azienda e i possibili rischi.

Il terzo capitolo descrive l'analisi dei requisiti del progetto di stage attraverso la modellazione dei casi d'uso e l'estrapolazione dei requisiti partendo da essi.

Il quarto capitolo approfondisce le tecnologie utilizzate per lo sviluppo del progetto, introduce la sua struttura principale e descrive le scelte progettuali effettuate.

Il quinto capitolo descrive le attività di codifica effettuate durante lo stage, dividendole tra frontend e backend.

Il sesto capitolo descrive le attività di verifica e validazione, descrivendo i test effettuati e i risultati ottenuti.

Il settimo capitolo presenta le conclusioni tratte dallo stage, esponendo le conoscenze acquisite, le competenze sviluppate e le considerazioni personali.

Per quanto riguarda la formattazione del testo nel documento, sono state seguite le seguenti convenzioni tipografiche:

- Le abbreviazioni, gli acronimi e i termini poco comuni menzionati sono definiti nel glossario, situato alla fine del seguente documento;
- I termini in lingua straniera o appartenenti al gergo tecnico sono evidenziati utilizzando il carattere corsivo;
- Ogni termine presente nel glossario, è contrassegnato dalla lettera 'G' come apice.

Capitolo 2

Descrizione del progetto di stage

Il seguente capitolo vuole introdurre brevemente il progetto affrontato durante lo stage, evidenziando gli obiettivi prefissati e i possibili rischi che si potrebbero incontrare.

2.1 Introduzione al progetto

Un punto cardine dell'architettura della piattaforma THRON è la suddivisione in micro servizi, che agevolano la manutenibilità e semplificano le operazioni di sviluppo. Allo stesso tempo rendono però più complessa la consultazione delle [G]API esposte e aumentano la quantità di comunicazione necessaria per mantenere tutti i reparti allineati. Per questo, tra le necessità che stanno avendo le aree di Prodotto e *Revenue* (*Sales*, *Marketing* e *Customer Services*) in THRON, è emersa l'esigenza di avere un unico punto da cui sia possibile consultare in maniera intuitiva, tutte le interfacce delle *API* che metta a disposizione l'area di Prodotto.
Il progetto di stage è consistito nello sviluppo di un portale in *Vue.js* per favorire la consultazione di tutte le *API* esposte, in modo centralizzato. Grazie al portale, la visualizzazione delle *API* è resa più semplice ed intuitiva, e per ogni servizio è possibile visualizzare la documentazione relativa con tutti gli *endpoint* disponibili, i relativi parametri e le risposte attese. Grazie a questa struttura, è possibile che l'utilizzatore del portale possa essere autonomo e facilitato dato che non è più necessario conoscere e cercare le *API*, con conseguente risparmio di tempo e risorse.
All'interno del portale è inoltre possibile provare i relativi *endpoint* delle singole *API* direttamente nell'applicativo, rendendo il progetto una soluzione completa in tutti i suoi aspetti che non necessita di applicazione di terze parti per il suo utilizzo, che comunque potranno essere utilizzate grazie alla possibilità di scaricare lo schema [G]YAML di ogni singola *API*.

2.2 Obiettivi dello stage

In questa sezione vengono elencati gli obiettivi prefissati da raggiungere durante lo stage, suddivisi in obbligatori, desiderabili e opzionali.
Verranno utilizzate le seguenti notazioni per fare riferimento ai requisiti:

- **OB**: per i requisiti obbligatori, vincolanti in quanto costituiscono l'obiettivo principale richiesto;
- **DE**: per i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
- **OP**: per i requisiti opzionali, rappresentanti valore aggiunto non necessariamente competitivo.

Le sigle precedentemente menzionate saranno seguite da una coppia sequenziale di numeri al fine di garantire un'identificazione univoca per ciascun requisito.

Obbligatori

- **OB1**: Realizzazione di un portale che consenta la consultazione degli [G] OpenAPIs schemas dei servizi pubblici e privati offerti da THRON;
- **OB2**: Rendere possibile l'utilizzo delle *API* direttamente dal portale (con inserimento manuale del *token* di autenticazione);
- **OB3**: Documentazione delle funzionalità implementate;
- **OB4**: Realizzazione di test di unità delle funzionalità implementate.

Desiderabili

- **DE1**: Implementare la funzionalità di recupero automatico degli *OpenAPI schemas*;
- **DE2**: Implementare la funzionalità di autenticazione al portale.

Opzionali

- **OP1**: Implementare la funzionalità di *download* dello schema di uno specifico servizio (formato *YAML*);
- **OP2**: Implementare la funzionalità di recupero automatico del *token* di autenticazione per l'utilizzo della *API* direttamente dal portale.

2.3 Analisi preventiva dei rischi

Durante la fase iniziale sono stati individuati dei possibili rischi a cui si potrà andare incontro. Per contrastare ciò, si è cercato di porre rimedio con delle contromisure appropriate.

1. ***Stack* tecnologico**:
   Descrizione: alcune tecnologie utilizzate per lo sviluppo del progetto erano per me nuove o poco conosciute. Ciò poteva portare a un utilizzo scorretto delle tecnologie, non rispettando le *best practice*.
   Soluzione: per ovviare a questo rischio, è stato previsto un periodo di formazione iniziale, durante il quale è stato possibile studiare le tecnologie da utilizzare e sperimentarle in piccoli progetti di prova.

2. **Fattibilità dei requisiti di partenza**:
   Descrizione: alcuni dei requisiti risultavano essere complessi da implementare e non era detto che fosse possibile soddisfarli in modo completo con gli strumenti a disposizione.
   Soluzione: è stato deciso in accordo con il tutor aziendale di effettuare un'analisi settimanale della situazione al fine di valutare l'andamento del progetto e porre rimedio ad eventuali criticità.

3. **Ritardi nello sviluppo**:
   Descrizione: potrebbero verificarsi ritardi nello sviluppo dovuti ad attività esterne al mio progetto (come attività infrastrutturali) necessarie per il suo completamento.
   Soluzione: è stato deciso in accordo con il tutor aziendale di effettuare un'analisi settimanale della situazione ed in caso di criticità è stato previsto un dialogo interno con il team infrastrutturale.

Capitolo 3

Analisi dei requisiti

In questo capitolo viene esposta l'analisi dei requisiti effettuata durante lo stage, dove si vanno ad illustrare le funzionalità tramite casi d'uso e requisiti identificati, con l'obiettivo di creare un'immagine più chiara e definita del sistema.

3.1 Descrizione dell'applicazione

Il progetto consiste nel creare un portale che permetta la consultazione di tutte le *API* THRON con la possibilità di provarle in modo semplice e veloce, direttamente dal portale. Il prodotto verrà utilizzato internamente all'azienda, più precisamente all'interno della *Product Area*, con lo scopo di facilitare attività di sviluppo, di testing e di supporto durante le attività giornaliere aziendali.
Il portale è disponibile in tre ambienti di sviluppo: *development*, *quality* e *production*, ognuno dei quali è identificato da un link diverso, che servirà per accedere al portale in quel determinato ambiente. La differenza principale tra ogni suddivisione è che i *client-id* sono diversi per ogni ambiente, ciò evita che un utente possa utilizzare un *client-id* di produzione involontariamente, che può causare problemi di sicurezza. Inoltre è utile per evitare prove indesiderate, dato che per esempio una chiamata *delete* su un *client-id* di produzione cancellerebbe effettivamente un dato.
Ogni *API* è formata da più *endpoint* al suo interno, che possono essere provati singolarmente. Ogni *endpoint* ha un elenco di possibili risposte che può ritornare, che dipende dai parametri inseriti nella chiamata. Alcune chiamate come le *POST* o *DELETE* avranno dei parametri obbligatori da inserire, mentre altre chiamate come le *GET*, avranno dei parametri opzionali.
In caso l'utente voglia provare le diverse chiamate al di fuori del portale, è possibile scaricare le *API* in formato *YAML*, così da poter usare strumenti di terze parti.

3.2 Casi d'uso

Questa sezione illustra i casi d'uso individuati nel corso dell'analisi dei requisiti del progetto che sono stati definiti utilizzando il linguaggio [G]UML (*Unified Modeling Language*). Ogni caso d'uso offre una panoramica chiara dei diversi attori coinvolti e delle interazioni che essi intraprendono nel contesto del sistema.
Ogni caso d'uso è indentificato da un codice univoco, che segue la seguente notazione:

UC[Codice-padre].[Codice-figlio]

3.2.1 Attori

Gli attori individuati nel sistema sono i seguenti:

- **Utente non autenticato**: è un utente che non è autenticato nel sistema e non può accedere alle funzionalità del portale;
- **Utente autenticato**: è un utente che è autenticato nel sistema e che può accedere alle funzionalità del portale;
- ***Microsoft 365***: è un servizio di autenticazione di *Microsoft* che permette di autenticarsi tramite *account* aziendale.

3.2.2 Descrizione del sistema

Di seguito viene illustrato un diagramma riassuntivo (in figura 3.1) che mostra i casi d'uso individuati nel sistema e le relazioni tra di essi.

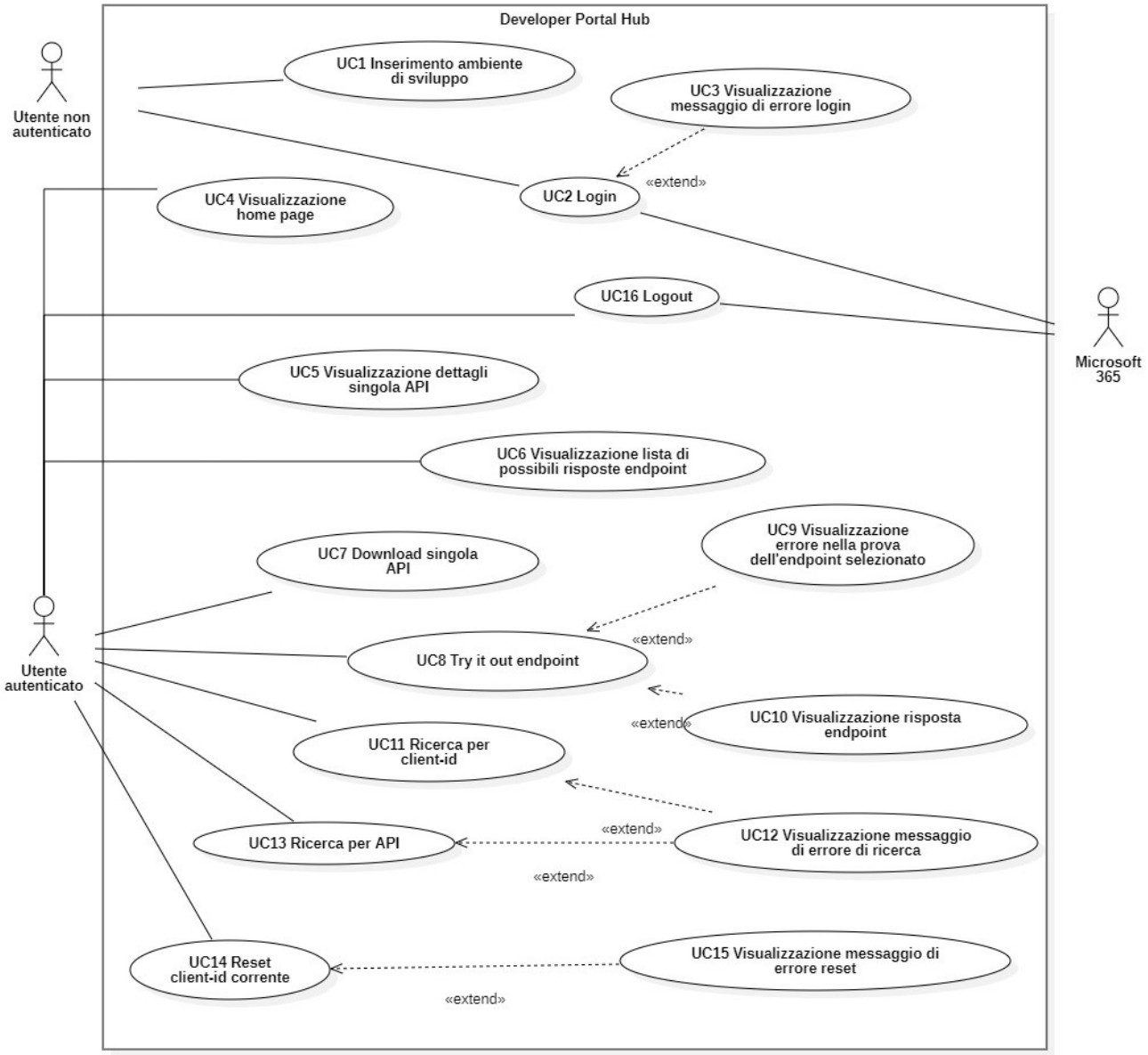


Figura 3.1: Scenario principale

UC1: Inserimento ambiente di sviluppo

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non è autenticato e non ha ancora avviato l'applicazione web.

Descrizione: L'utente vuole avviare l'applicazione web e deve scegliere in che ambiente di sviluppo farlo.

Postcondizioni: L'utente avvia l'applicazione nell'ambiente di sviluppo scelto.

Scenario Principale:

1. L'utente seleziona il link del portale a seconda dell'ambiente di sviluppo che vuole avviare (*development*, *quality* e *production*).

UC2: Login

Attori Principali: Utente non autenticato, *Microsoft 365*.

Precondizioni: L'utente possiede un *account* valido per autenticazione tramite *Microsoft 365* che appartiene al gruppo autorizzato per il login al sistema. Inoltre l'utente non è autenticato e si trova nella pagina di login.

Descrizione: L'utente vuole accedere al sistema e deve inserire le proprie credenziali per accedervi.

Postcondizioni: L'utente è autenticato correttamente e può procedere con l'utilizzo di tutte le funzionalità disponibili all'interno del sistema.

Scenario Principale:

1. L'utente inserisce la propria e-mail;
2. L'utente inserisce la propria password;
3. *Microsoft 365* verifica le credenziali inserite, in base ai permessi configurati.

Estensioni:

1. Visualizzazione messaggio di errore login UC3.

Generalizzazioni: Come da figura 3.2:

1. Inserimento e-mail UC2.1;
2. Inserimento password UC2.2.

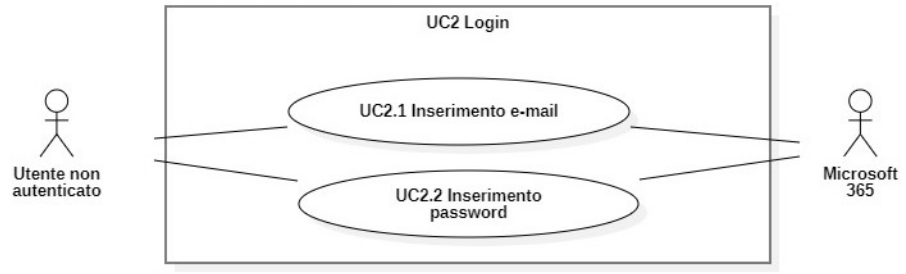


Figura 3.2: UC2 Login

UC2.1: Inserimento e-mail

Attori Principali: Utente non autenticato.

Precondizioni: L'utente possiede un *account* valido per autenticazione tramite *Microsoft 365* che appartiene al gruppo autorizzato per il login al sistema. Inoltre l'utente non è autenticato e si trova nella pagina di login.

Descrizione: L'utente deve inserire la propria e-mail per autenticarsi al sistema.

Postcondizioni: L'utente ha inserito la propria e-mail, può quindi procedere a completare il processo di autenticazione.

Scenario Principale:

1. L'utente inserisce nell'apposito campo la propria e-mail.

UC2.2: Inserimento password

Attori Principali: Utente non autenticato.

Precondizioni: L'utente possiede un *account* valido per autenticazione tramite *Microsoft 365* che appartiene al gruppo autorizzato per il login al sistema. Inoltre l'utente non è autenticato e si trova nella pagina di login.

Descrizione: L'utente deve inserire la propria password per autenticarsi al sistema.

Postcondizioni: L'utente ha inserito la propria password e può concludere il processo di autenticazione.

Scenario Principale:

1. L'utente inserisce nell'apposito campo la propria password.

UC3: Visualizzazione messaggio di errore login

Attori Principali: Utente non autenticato.

Precondizioni: L'utente ha inserito una tra le due credenziali e-mail o password in modo errato.

Descrizione: L'utente deve inserire delle credenziali corrette per poter effettuare il login correttamente.

Postcondizioni: L'utente ha inserito una tra le due credenziali errate.

Scenario Principale:

1. L'utente visualizza un messaggio di errore che lo informa che una delle credenziali che ha inserito per autenticarsi al sistema è sbagliata.

UC4: Visualizzazione *home page*

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato ed è stato reindirizzato alla pagina principale.

Descrizione: L'utente vuole visualizzare la pagina principale.

Postcondizioni: L'utente ha visualizzato la pagina principale.

Scenario Principale:

1. L'utente visualizza la pagina principale.

Generalizzazioni: Come da figura 3.3:

1. Visualizzazione lista *APIs* disponibili UC4.1;
2. Visualizzazione *client-id* di default UC4.2;
3. Visualizzazione dettagli utente autenticato UC4.3.

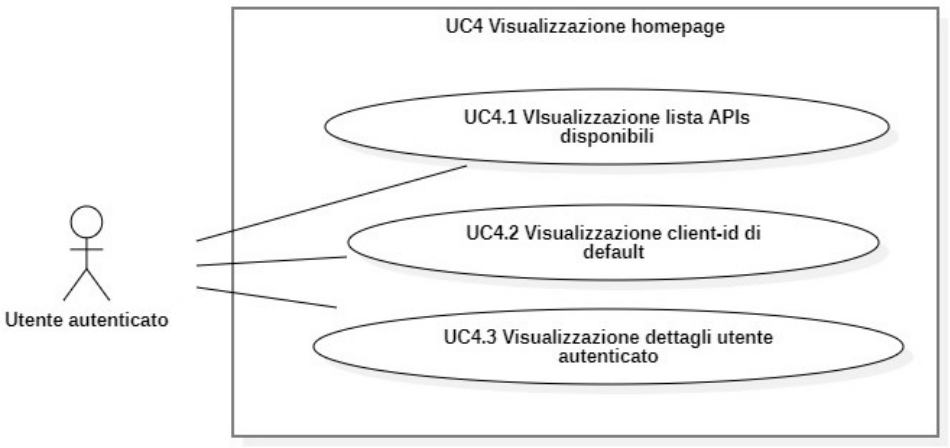


Figura 3.3: UC4 Visualizzazione home page

UC4.1: Visualizzazione lista *APIs* disponibili

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato ed è stato reindirizzato alla pagina principale.

Descrizione: L'utente vuole visualizzare la lista di *API* disponibili per la consultazione all'interno del sistema.

Postcondizioni: L'utente ha visualizzato la lista di *API* disponibili all'interno del sistema.

Scenario Principale:

1. L'utente visualizza la lista di *API* disponibili all'interno del sistema.

UC4.2: Visualizzazione *client-id* di default

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato ed è stato reindirizzato alla pagina principale.

Descrizione: L'utente vuole visualizzare il *client-id* di default impostato nell'ambiente corrente.

Postcondizioni: L'utente ha visualizzato il *client-id* di default impostato nell'ambiente corrente in cui si trova.

Scenario Principale:

1. L'utente visualizza il *client-id* di default impostato nell'ambiente corrente in cui si trova.

UC4.3: Visualizzazione dettagli utente autenticato

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato ed è stato reindirizzato alla pagina principale.

Descrizione: L'utente vuole visualizzare i propri dati personali, ovvero del proprio utente autenticato.

Postcondizioni: L'utente visualizza i dati personali del proprio *account* autenticato nel sistema.

Scenario Principale:

1. L'utente visualizza le informazioni personali del proprio *account* autenticato nel sistema.

UC5: Visualizzazione dettagli singola *API*

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e si trova nella pagina principale.

Descrizione: L'utente vuole visualizzare la pagina di dettaglio di una singola *API*.

Postcondizioni: L'utente ha visualizzato la pagina di dettaglio di una singola *API* tra quelle presenti nel sistema.

Scenario Principale:

1. L'utente clicca su una delle *API* presenti nella lista di *API* disponibili all'interno del sistema.
2. L'utente visualizza la pagina di dettaglio della singola *API* selezionata.

Generalizzazioni: Come da figura 3.4:

1. Visualizzazione lista *endpoint* disponibili UC5.1.

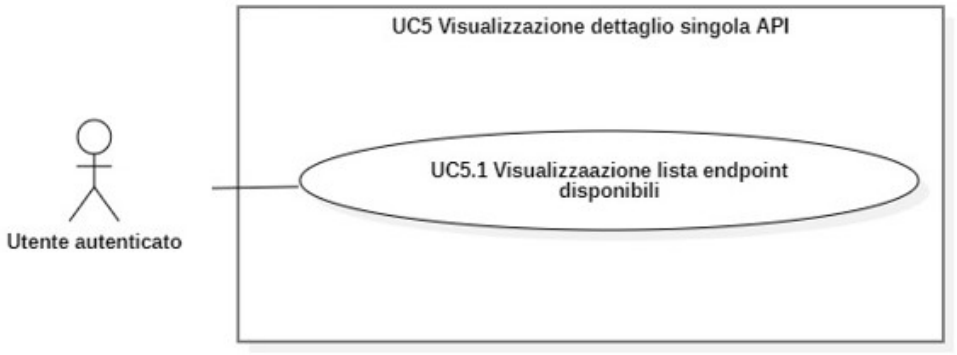


Figura 3.4: UC5 Visualizzazione dettaglio singola API

UC5.1: Visualizzazione lista *endpoint* disponibili

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e sta visualizzando la pagina di dettaglio di una singola *API*.

Descrizione: L'utente vuole visualizzare la lista completa di *endpoint* disponibili per l'*API*.

Postcondizioni: L'utente visualizza la lista completa di *endpoint* disponibili per l'*API*.

Scenario Principale:

1. L'utente visualizza la lista completa di *endpoint* disponibili per l'*API* che ha selezionato.

Generalizzazioni: Come da figura 3.5:

1. Visualizzazione dettaglio singolo *endpoint* UC5.1.1.

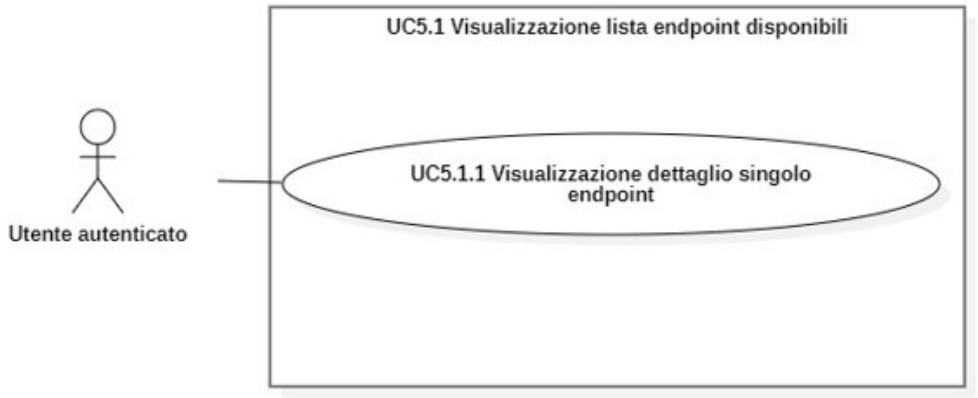


Figura 3.5: UC5.1 Visualizzazione lista endpoint disponibili

UC5.1.1: Visualizzazione dettaglio singolo *endpoint*

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato, sta visualizzando la pagina di dettaglio di una singola *API* contenente la lista di *endpoint* disponibili.

Descrizione: L'utente vuole visualizzare la pagina di dettaglio di un singolo *endpoint*.

Postcondizioni: L'utente visualizza la pagina di dettaglio di un singolo *endpoint*.

Scenario Principale:

1. L'utente clicca sullo specifico *endpoint* che vuole visualizzare;
2. L'utente visualizza i dettagli disponibili per l'*endpoint* selezionato.

Estensioni:

1. Visualizzazione lista di possibili risposte *endpoint* UC6.

UC6: Visualizzazione lista di possibili risposte *endpoint*

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato, sta visualizzando la sezione di dettaglio di un singolo *endpoint*.

Descrizione: L'utente vuole visualizzare la lista dei possibili risultati possibili per l'*endpoint* selezionato.

Postcondizioni: L'utente visualizza la lista delle possibili risposte disponibili per l'*endpoint* che ha selezionato.

Scenario Principale:

1. L'utente visualizza i dettagli riguardanti le possibili risposte che l'*endpoint* selezionato può ritornare.

UC7: Download singola *API*

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato ed ha selezionato una *API* dalla lista di *API* disponibili nel sistema.

Descrizione: L'utente vuole poter scaricare in formato *YAML* un *API* dal portale.

Postcondizioni: L'utente ha scaricato in formato *YAML* un *API* dal portale.

Scenario Principale:

1. L'utente scarica il formato *YAML* di un *API* dal portale, tra quelle disponibili;
2. Una nuova pagina si apre e il download dell'*API* viene eseguito;
3. Il nome del file è già impostato con il nome dell'*API* scaricata.

UC8: *Try it out endpoint*

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato, sta visualizzando i dettagli di un singolo *endpoint* di un *API* disponibile nel sistema.

Descrizione: L'utente vuole poter provare l'*endpoint* selezionato.

Postcondizioni: L'utente ha provato l'*endpoint* selezionato.

Scenario Principale:

1. L'utente ha selezionato una determinata *API*;
2. L'utente ha selezionato un determinato *endpoint* di quell'*API*;
3. L'utente ha cliccato sul pulsante per provare l'*endpoint* selezionato.

Estensioni:

1. Visualizzazione errore nella prova dell'*endpoint* selezionato UC9;

2. Visualizzazione risposta *endpoint* UC10.

Generalizzazioni: Come da figura 3.6:

1. Inserimento parametri per *try it out endpoint* UC8.1;
2. Definire campi aggiuntivi per *try it out endpoint* UC8.2.

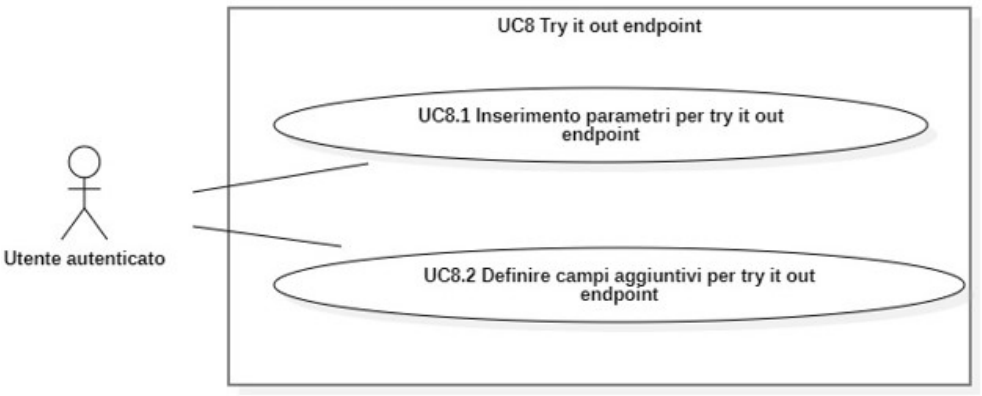


Figura 3.6: UC8 Try it out endpoint

UC8.1: Inserimento parametri per *try it out endpoint*

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato, sta visualizzando la schermata di *try it out* nella sezione di inserimento dei parametri.

Descrizione: L'utente vuole poter inserire i parametri necessari per la prova dell'*endpoint*.

Postcondizioni: L'utente ha inserito i parametri necessari alla chiamata verso l'*endpoint*.

Scenario Principale:

1. L'utente inserisce i parametri richiesti per la chiamata verso l'*endpoint* selezionato.

UC8.2: Definire campi aggiuntivi per *try it out endpoint*

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato, sta visualizzando la schermata di *try it out* nella sezione dei parametri aggiuntivi.

Descrizione: L'utente vuole poter definire dei campi aggiuntivi ai parametri già esistenti, per poi andare a provare la chiamata verso l'*endpoint*.

Postcondizioni: L'utente ha creato dei parametri aggiuntivi per la chiamata verso l'*endpoint*.

Scenario Principale:

1. L'utente definisce dei parametri da aggiungere a quelli già esistenti.

UC9: Visualizzazione errore nella prova dell'*endpoint* selezionato

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato, è nella sezione di prova di un *endpoint* ed ha inserito dei parametri errati.

Descrizione: L'utente deve inserire dei parametri corretti per poter provare l'*endpoint* senza riscontrare errori.

Postcondizioni: L'utente ha inserito dei parametri parzialmente o in modo scorretto e non può procedere con l'esecuzione della chiamata.

Scenario Principale:

1. L'utente visualizza un messaggio di errore che lo avvisa che i parametri inseriti non sono corretti, o risultano incompleti.

UC10: Visualizzazione risposta *endpoint*

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato, è nella sezione di prova di un *endpoint* ed ha inserito dei parametri corretti.

Descrizione: L'utente vuole visualizzare la risposta dell'*endpoint*.

Postcondizioni: L'utente visualizza la risposta adeguata alla chiamata verso l'*endpoint*, in base ai parametri inseriti.

Scenario Principale:

1. L'utente visualizza la risposta dell'*endpoint*, uno tra quelle possibili contenute nella descrizione dell'*endpoint*. La risposta varia in base ai parametri inseriti.

UC11: Ricerca per *client-id*

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e sta navigando all'interno del sistema.

Descrizione: L'utente vuole poter ricercare un *client-id* all'interno del sistema.

Postcondizioni: L'utente effettua una ricerca per *client-id* e il sistema ha restituito i risultati della ricerca.

Scenario Principale:

1. L'utente clicca il bottone per effettuare la ricerca per *client-id*;
2. L'utente inserisce nell'apposito campo il *client-id* che vuole cercare all'interno del sistema;
3. Il sistema ricerca il *client-id* inserito e restituisce i risultati della ricerca;
4. Il sistema aggiunge l'ultima ricerca effettuata alla cronologia delle ultime ricerche.

Estensioni:

1. Visualizzazione messaggio di errore di ricerca UC 12.

UC12: Visualizzazione messaggio di errore di ricerca

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e ha cercato un *client-id* o un *API* inesistente nel sistema.

Descrizione: L'utente deve inserire un *client-id* o un *API* esistente nel sistema per poter effettuare la ricerca. Inoltre il *client-id* deve esistere per l'ambiente selezionato.

Postcondizioni: L'utente ha inserito un *client-id* o un *API* inesistente nel sistema e visualizza un messaggio di errore.

Scenario Principale:

1. L'utente visualizza un messaggio di errore che lo informa che la ricerca effettuata non ha prodotto nessun risultato, indicando che ciò è dovuto al fatto che il *client-id* o l'*API* cercata non è presente nel sistema, oppure non esiste quel *client-id* per l'ambiente selezionato.

UC13: Ricerca per *API*

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e sta navigando all'interno del sistema.

Descrizione: L'utente vuole poter ricercare un *API* all'interno del sistema.

Postcondizioni: L'utente effettua una ricerca per *API* e il sistema ha restituito i risultati della ricerca.

Scenario Principale:

1. L'utente clicca il bottone per effettuare la ricerca per *API*;
2. L'utente inserisce nell'apposito campo l'*API* che vuole cercare all'interno del sistema;
3. Il sistema ricerca l'*API* inserita e restituisce i risultati della ricerca;
4. Il sistema aggiunge l'ultima ricerca effettuata alla cronologia delle ultime ricerche.

Estensioni:

1. Visualizzazione messaggio di errore di ricerca UC12.

UC14: Reset *client-id* corrente

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato, sta navigando all'interno del sistema e ha già selezionato un *client-id* che non è quello di default.

Descrizione: L'utente vuole poter resettare il *client-id* corrente e tornare al *client-id* di default per l'ambiente selezionato.

Postcondizioni: L'utente ha resettato il *client-id* corrente e ora visualizza il *client-id* di default per l'ambiente selezionato.

Scenario Principale:

1. L'utente clicca il bottone per resettare il *client-id* corrente;
2. Il sistema resetta il *client-id* corrente ed imposta il *client-id* al valore di default a seconda dell'ambiente selezionato.

Estensioni:

1. Visualizzazione messaggio di errore reset UC15.

UC15: Visualizzazione messaggio di errore reset

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e ha provato a resettare il *client-id* corrente che è quello di default per l'ambiente selezionato.

Descrizione: L'utente deve aver selezionato un *client-id* diverso da quello di default, per poterlo resettare.

Postcondizioni: L'utente ha provato a resettare il *client-id* corrente, ma risulta essere quello di default.

Scenario Principale:

1. L'utente visualizza un messaggio di errore che lo informa che il *client-id* corrente è quello di default per l'ambiente selezionato e non può essere resettato.

UC16: Logout

Attori Principali: Utente autenticato, *Microsoft 365*.

Precondizioni: L'utente è autenticato e vuole uscire dalla sessione corrente.

Descrizione: L'utente vuole effettuare il logout dal sistema.

Postcondizioni: L'utente effettua il logout dal sistema terminando la sessione corrente, non è più autenticato e viene reindirizzato alla pagina di login.

Scenario Principale:

1. L'utente clicca il bottone per effettuare il logout;
2. *Microsoft 365* effettua il logout dell'utente terminando la sessione.

3.3 Tracciamento dei requisiti

In questa sezione vengono riportati i requisiti individuati durante il progetto di stage. Questo capitolo si sofferma in particolare sulla classificazione dei requisiti in tre categorie principali:

- **Requisiti funzionali**: delineano le funzionalità che il sistema deve offrire. Essi delineano le azioni specifiche che il sistema deve eseguire, le risposte attese a determinati input e le dinamiche generali delle operazioni;
- **Requisiti qualitativi**: definiscono gli aspetti legati alla qualità, all'usabilità e alle prestazioni del sistema;
- **Requisiti di vincolo**: delineano le restrizioni e i parametri che il sistema deve rispettare durante lo sviluppo e l'implementazione.

Inoltre viene fatta una classificazione dei requisiti in base alla priorità assegnata a ciascun requisito.

3.3.1 Notazione

Ciascun requisito è identificato da un codice univoco, che segue la seguente notazione:

R[Priorità][Tipo]-[Codice]

dove:

- **Priorità** indica il livello di priorità assegnato: obbligatorio (**O**), desiderabile (**D**) e opzionale (**Z**);
- **Tipo** indica il tipo di requisito: funzionale (**F**), qualitativo (**Q**) e di vincolo (**V**);
- **Codice** indica il codice identificativo del requisito.

Nelle tabelle 3.1, 3.3 e 3.5 sono riassunti i requisiti tramite una breve descrizione accompagnata dalle fonti da cui è stato individuato il requisito per facilitarne la tracciabilità.

3.3.2 Requisiti funzionali

Tabella 3.1: Tabella del tracciamento dei requisiti funzionali

Requisito	Descrizione	Fonte
RFO-1	L'utente deve scegliere l'ambiente di sviluppo	UC1
RFO-2	Il sistema deve permettere l'autenticazione ad un utente con un *account* valido	UC2
RFO-2.1	L'utente deve inserire la propria mail	UC2.1
RFO-2.2	L'utente deve inserire la propria password	UC2.2
RFO-3	Il sistema deve avvisare l'utente tramite un messaggio di errore che le credenziali inserite nel login sono errate	UC3
RFO-4	Il sistema deve reindirizzare l'utente alla pagina principale, dopo che il login è andato a buon fine	UC4
RFO-4.1	L'utente deve poter visualizzare la lista di *API* disponibili nel sistema	UC4.1
RFO-4.2	L'utente deve poter visualizzare la lista di *client-id* di default impostata nell'ambiente di sviluppo in cui si trova	UC4.2
RFO-4.3	L'utente deve poter visualizzare i dettagli relativi al suo *account*	UC4.3
RFO-5	L'utente deve poter visualizzare i dettagli relativi ad una singola *API*	UC5
RFO-5.1	L'utente deve poter visualizzare la lista di *endpoint* disponibili all'interno del sistema	UC5.1
RFO-5.1.1	L'utente deve poter visualizzare i dettagli relativi ad un singolo *endpoint* di una determinata *API*	UC5.1.1
RFO-6	L'utente deve poter visualizzare l'elenco delle possibili risposte per il determinato *endpoint* selezionato	UC6
RFO-7	Il sistema deve permettere il download all'utente di una singola *API*	UC7
RFO-8	Il sistema deve permettere il *try it out* di un singolo *endpoint* all'utente	UC8
RFO-8.1	Il sistema deve permettere l'inserimento dei parametri necessari per il *try it out* di un *endpoint*	UC8.1
RFD-8.2	Il sistema deve permettere la possibilità di definire dei campi aggiuntivi per il *try it out* di un *endpoint*	UC8.2

Continuazione della tabella 3.1		
RFO-9	Il sistema deve avvisare l'utente tramite un messaggio di errore che i parametri inseriti non sono corretti o incompleti	UC9
RFO-10	L'utente deve poter visualizzare la risposta dell'*endpoint* che ha provato	UC10
RFO-11	Il sistema deve permettere all'utente di poter effettuare una ricerca per *client-id*	UC11
RFO-12	Il sistema deve avvisare l'utente tramite un messaggio di errore che la ricerca effettuata non ha portato a risultati presenti nel sistema	UC12
RFO-13	Il sistema deve permettere all'utente di poter effettuare una ricerca per *API*	UC13
RFD-14	Il sistema deve permettere il reset del *client-id* corrente	UC14
RFO-15	Il sistema deve avvisare l'utente tramite un messaggio di errore che il *client-id* è già di default	UC15
RFO-16	Il sistema deve permettere il logout all'utente, uscendo dalla sessione	UC16

3.3.3 Requisiti qualitativi

Tabella 3.3: Tabella del tracciamento dei requisiti qualitativi

Requisito	Descrizione	Fonte
RQO-1	Il progetto deve essere accompagnato da documentazione tecnica e funzionale	Interno
RQO-2	La parte frontend del progetto deve essere coperta da test di unità	Interno
RQZ-3	L'applicazione web deve avere un'interfaccia responsive	Interno
RQD-5	L'applicativo deve essere accessibile utilizzando i principali browser	Interno

3.3.3.1 Requisiti di vincolo

Tabella 3.5: Tabella del tracciamento dei requisiti di vincolo

Requisito	Descrizione	Fonte
RVO-1	L'applicazione deve essere sviluppata utilizzando il *framework Vue.js* 3, usando *TypeScript* come linguaggio di programmazione	UC1
RVO-2	I componenti devono essere scritti utilizzando le [G]Composition API di *Vue.js 3*	UC1
RVD-3	I componenti di base devono essere implementati utilizzando la libreria *THRON Components*	UC2
RVZ-4	L'interfaccia dell'applicazione deve seguire il *design system* THRON	UC2

Capitolo 4

Struttura principale e progettazione

In questo capitolo sono descritte la struttura principale del progetto e le attività di progettazione dell'applicativo. Inoltre vengono descritte le tecnologie utilizzate durante lo sviluppo del progetto e le scelte architetturali.

4.1 Tecnologie utilizzate

Di seguito viene data una panoramica delle tecnologie utilizzate durante lo sviluppo del progetto di stage.

4.1.1 Frontend

4.1.1.1 Vue.js

Vue.js è un *framework JavaScript* progressivo e reattivo, utilizzato per lo sviluppo di interfacce utente dinamiche e moderne. Creato da *Evan You*, *Vue.js* è apprezzato per la sua semplicità d'uso e flessibilità. Con un sistema di reattività basato su un modello di oggetti e dipendenze, *Vue.js* rende facile il monitoraggio e l'aggiornamento automatico dell'interfaccia utente in base ai cambiamenti di stato dei dati. La sua architettura basata su componenti consente di organizzare il codice in moduli riutilizzabili e autonomi, semplificando la creazione di applicazioni complesse. Grazie alle direttive, è possibile arricchire il [G]DOM con funzionalità reattive, mentre il sistema di *routing* agevola la creazione di [G]single page applications. Con una crescita costante della comunità di sviluppatori, *Vue.js* è diventato un'opzione popolare nel mondo dello sviluppo frontend. Per il mio progetto ho utilizzato la versione 3 di *Vue.js*, insieme allo *script setup*, che è una nuova sintassi per definire componenti, progettata per semplificare la struttura del codice [29].

4.1.1.2 TypeScript

TypeScript è un linguaggio di programmazione [G]open-source sviluppato da *Microsoft*. Si basa su *JavaScript* e offre tipizzazione statica opzionale, consentendo agli sviluppatori di specificare tipi per variabili, parametri di funzioni e oggetti. Questa caratteristica aiuta a individuare errori e a migliorare la manutenibilità del codice [25].
All'interno del mio progetto ho creato per la parte frontend una cartella chiamata *types*, che contiene un file *TypeScript* con al suo interno tutti i tipi utilizzati nel progetto.

4.1.1.3 Vite.js

Vite.js è un *build tool* utilizzato per lo sviluppo di applicazioni web. È stato creato da *Evan You*, lo stesso creatore di *Vue.js*, e si basa su [G]Rollup.js. *Vite.js* è stato progettato per essere veloce, semplice da utilizzare e facile da configurare. La sua velocità è dovuta al fatto che utilizza la tecnica dell'[G]ESM (*ECMAScript Modules*) che permette di caricare i moduli in modo asincrono, riducendo i tempi di compilazione e di [G]hot-reload [26].

4.1.1.4 Sass

Sass è un'estensione di *CSS* che offre funzionalità aggiuntive e avanzate per semplificare e organizzare il modo in cui viene scritto e gestito il codice. Può essere considerato un preprocessore *CSS*, in quanto viene compilato in *CSS* prima di essere interpretato dal *browser*. *Sass* inoltre permette di utilizzare funzionalità non disponibili in *CSS* nativo, offrendo una serie di funzioni, variabili, [G]mixin e altro [18].
Insieme a *Sass*, ho utilizzato *BEM*, ovvero una metodologia di *naming convention* utilizzata nel mondo dello sviluppo web.

4.1.2 Backend

4.1.2.1 Nest.js

Nest.js è un *framework* per applicazioni *server-side* basato su *Node.js*. Si basa su [G]Express.js e *TypeScript* ed è progettato per creare applicazioni scalabili e performanti. Il *framework* in questione combina concetti e caratteristiche provenienti da diversi paradigmi di sviluppo, tra cui la programmazione orientata agli oggetti (OOP), la programmazione funzionale e la programmazione reattiva [12].

4.1.3 Altre tecnologie di supporto

4.1.3.1 Node.js

Node.js è un ambiente di *runtime JavaScript open-source* progettato per eseguire codice lato server [14]. Per gestire le dipendenze del mio progetto, ho deciso di utilizzare

[G]pnpm come gestore di pacchetti. Questa selezione ha portato a un miglior utilizzo delle risorse di sistema e ha notevolmente accelerato il processo di installazione delle dipendenze.

4.1.3.2 Pinia

Pinia è una libreria per la gestione dello stato per applicazioni *Vue.js*. Promuove l'uso di *store* modulari, ognuno dei quali gestisce uno stato specifico dell'applicazione [15].

4.1.3.3 Vue-router

Vue-router è una libreria per la gestione delle *route* per le applicazioni *Vue.js*. Permette di definire le *route* dell'applicazione e di navigare tra le pagine [28].

4.1.4 Versionamento

4.1.4.1 Git

Git è un sistema di controllo versione distribuito e altamente flessibile, utile per tenere traccia delle modifiche apportate al codice sorgente durante lo sviluppo di un progetto [G]Software [8].

4.1.4.2 CodeCommit

AWS CodeCommit rappresenta un servizio di *hosting* di *repository* altamente scalabile, che è gestito all'interno dell'ecosistema di [G]AWS (*Amazon Web Services*). Con *CodeCommit*, è possibile ospitare *repository Git* privati in un ambiente sicuro e flessibile [4].

4.1.5 Verifica

4.1.5.1 ESLint

ESLint è uno strumento *open-source* ampiamente utilizzato per l'analisi statica del codice *JavaScript*. Esso permette di identificare e segnalare potenziali errori o pratiche non conformi durante la fase di sviluppo.

4.1.5.2 Vitest

Vitest è un *framework* per l'implementazione di test di unità, utilizzato prevalentemente in progetti *Vue.js*. Usato in coppia con *Vite* permette di eseguire test di unità in modo più veloce e semplice [27].

4.1.6 Librerie esterne utilizzate

4.1.6.1 THRON Components

La libreria *THRON Components* contiene i componenti utilizzati per creare elementi comuni, come i bottoni del portale, seguendo il *design system* aziendale.

4.1.6.2 Azure MSAL

Azure MSAL è una libreria che permette di integrare il login con [G]Azure AD (*Azure Active Directory*) all'interno di un'applicazione web [5].

4.1.6.3 Swagger UI

Swagger UI è una libreria *open-source* progettata per semplificare la visualizzazione e l'interazione con la documentazione delle *API* [21].

4.2 Struttura principale del sistema

Il sistema è composto da due principali sezioni:

- Frontend, ovvero l'interfaccia utente dell'applicazione web che permette all'utilizzatore di interagire con il sistema. È responsabile di presentare i contenuti in modo visivamente attraente e interattivo, consentendo agli utenti di navigare, inserire dati e svolgere azioni specifiche. Per lo sviluppo di questa parte del sistema è stato utilizzato il *framework Vue.js*;
- Backend, ovvero la parte del sistema che elabora le richieste provenienti dal frontend e restituisce i risultati. È responsabile della gestione dei dati e della logica di *business*. Lo sviluppo di questa parte del sistema è stato realizzato utilizzando il *framework Nest.js*.

4.2.1 Ambienti di sviluppo

Gli ambienti di *staging* sono ambienti di test che vengono utilizzati per testare le funzionalità dell'applicazione prima di rilasciarla in produzione. A livello aziendale sono stati definiti tre ambienti di cui i primi due di *staging*, denominati come segue:

- ***Development***: consente di validare a livello tecnico le funzionalità;
- ***Quality***: consente di validare a livello funzionale o di implementazione le funzionalità;
- ***Production***: ambiente di produzione in cui viene rilasciata la funzionalità.

4.2.2 Configurazione dell'ambiente per lo sviluppo del progetto

Il progetto di stage, necessita di cartelle per la configurazione dell'ambiente di sviluppo, per la configurazione del progetto e per la configurazione del *deploy*. Più precisamente, il progetto segue la seguente struttura:

Buildspec

La cartella *Buildspec* contiene i file di configurazione per la *build* dell'applicazione su [G]AWS CodeBuild. Questo è il primo step del processo di *deploy*, in quanto viene eseguita la *build* sia del *middleware* che del portale. La cartella è formata a sua volta da tre file: *buildspec_development*, *buildspec_quality* e *buildspec_production*. Ognuno di questi file configura la *build* dell'applicazione in base all'ambiente di *deploy*.
Successivamente dopo aver effettuato tutti i comandi specificati nei file *buildspec*, inizia lo step di *deploy*, specificato nella cartella *Infra*. Ognuno dei file *buildspec* è un file *YAML*, che è formato dalle seguenti sezioni:

- La sezione *env* dove vengono specificate le variabili d'ambiente utilizzate nel progetto;
- La sezione di *pre-build* dove vengono specificati i comandi da eseguire prima della *build*;
- La sezione di *build* dove vengono specificati i comandi per la *build* del portale e del *middleware*.

Infra

La cartella *Infra* contiene i file di configurazione per il *deploy* dell'applicazione su *AWS*. È scritta utilizzando il linguaggio di programmazione [G]Python.
La cartella è formata da due file: '*app.py*' e '*stack.py*'. Il primo file contiene la configurazione per il *deploy* dell'applicazione, mentre il secondo file contiene la configurazione per il *deploy* dell'infrastruttura.
Per quanto riguarda l'infrastruttura del progetto, ho utilizzato e configurato due costrutti, anche chiamati *CDK Constructs* o semplicemente *Constructs*, che sono delle classi che rappresentano un componente dell'infrastruttura. Il primo costrutto si chiama *THRONCloudFrontDistribution* e consente di accelerare la distribuzione dei contenuti web statici e dinamici, come file *HTML*, *CSS*, *JS* e immagini, agli utenti. In breve questo costrutto genera il *template* di [G]AWS CloudFront ed esegue il *deploy* degli *asset* statici su un [G]Bucket S3.
Il secondo costrutto che ho utilizzato si chiama *THRONDockerLambda* e consente di creare una funzione [G]lambda in cui il gestore è un'immagine [G]docker. Nel caso del mio progetto, mi è servito per creare una *lambda* che una volta invocata avvia un *docker*

con all'interno il progetto backend.

Middleware

La cartella *Middleware* contiene il progetto backend in *Nest.js*. Nello specifico la cartella è formata dalle seguenti sezioni:

- La cartella *src* che contiene il codice sorgente dell'applicazione, dove vengono specificati i vari *endpoint* che utilizzo sulla parte frontend, lo *script* per la configurazione della *lambda*, una cartella con gli *helpers* e la cartella del *middleware*;
- File *env* che contiene le variabili d'ambiente utilizzate nel progetto;
- Una cartella *node_ modules* che contiene le dipendenze del progetto.

Portal

La cartella *portal* contiene il progetto frontend in *Vue.js*. Nello specifico la cartella è formata dalle seguenti sezioni:

- La cartella *public* contenente il file '*index.html*', ovvero il file principale dell'applicazione;
- Una cartella *node_ modules* che contiene le dipendenze del progetto;
- La cartella *src* che contiene il codice sorgente dell'applicazione, dove vengono specificati i vari componenti che utilizzo, i vari *store*, i vari *router* e le varie *utilities*;
- File *env* che contiene le variabili d'ambiente utilizzate nel progetto.

Dockerfile

Il file *Dockerfile* è un file *docker* che contiene le istruzioni per creare un'immagine che viene utilizzata per eseguire il *middleware* all'interno di una *lambda*.
Utilizzando la struttura *Dockerfile*, ho creato un ambiente isolato in cui il *middleware* può essere configurato e utilizzato come parte della mia funzione *lambda*. Quando la funzione viene invocata, avvia il *Docker* [G]container ed esegue il *middleware* all'interno dell'ambiente containerizzato, garantendo che esso sia parte integrante dell'applicazione [G]serverless.

4.3 Progettazione

4.3.1 Architettura frontend

4.3.1.1 Architettura Vue.js

Vue.js è un *framework* utilizzato nelle *single page applications*, che permette di definire le pagine web in modo modulare, utilizzando componenti riutilizzabili. I componenti costituiscono la base dell'architettura di *Vue*. Essi rappresentano una parte isolata dell'interfaccia, che può contenere il proprio modello, i propri stili e la propria logica, infatti ogni componente ha il proprio *template* scritto in *HTML*, il proprio *script* scritto nel mio caso in *TypeScript* e i propri stili scritti nel mio caso in *Scss*. Come già accennato in precedenza, i componenti sono riutilizzabili all'interno di un'applicazione e possono essere combinati tra loro per creare gerarchie di interfacce ancora più complesse.

L'architettura di *Vue.js* è basata sul pattern architetturale *MVVM* (*Model-View-ViewModel*), che è una variante del pattern [G]MVC (*Model-View-Controller*), dove:

- ***Model***: rappresenta lo stato, i dati e le regole di *business* dell'applicazione, che gestiscono l'accesso e la modifica di tali dati. Lo stato viene definito tramite l'uso di particolari variabili di tipo reattivo, che permettono di aggiornare automaticamente la *View* associata in caso di modifiche;
- ***View***: è l'interfaccia utente, che visualizza i dati contenuti nel *Model* e si occupa di reagire agli input dell'utente. La *View* è definita utilizzando i template *Vue.js* e viene reattivamente aggiornata in base ai cambiamenti del modello. La vista viene definita utilizzando un *template*, ovvero una direttiva dell'*HTML*, arricchita con alcune direttive *Vue.js*. Queste particolari direttive permettono di collegare elementi del *DOM* a proprietà o metodi del modello, in modo che la *View* possa reagire agli input dell'utente e aggiornare automaticamente lo stato dell'applicazione;
- ***ViewModel***: è l'intermediario tra la *View* e il *Model*. Il *ViewModel* gestisce la logica dell'interfaccia utente e mantiene lo stato dell'applicazione sincronizzato con la *View*. Il *ViewModel* è rappresentato da un componente *Vue.js*, infatti esso è un'istanza che collega il modello e la vista. All'interno di un componente è possibile definire metodi, proprietà computate, metodi del ciclo di vita, gestione di eventi e molte altre funzionalità. Questo consente di definire la logica di presentazione e di manipolare i dati all'interno di un contesto definito.

In breve, l'architettura è incentrata sulla creazione e utilizzo di componenti riutilizzabili che al loro interno incorporano sia il modello che la vista. Un aspetto che rende *Vue.js* diverso da altri *framework* è proprio il concetto di reattività, infatti *Vue.js* è in

grado di rilevare automaticamente le dipendenze tra i componenti, in modo da poter aggiornare automaticamente l'interfaccia utente [3].

4.3.2 Architettura backend

4.3.2.1 Architettura Nest.js

L'architettura di *Nest.js* si basa su diversi principi chiave e concetti fondamentali che lo rendono un *framework* efficace per la creazione di applicazioni *server-side*. La caratteristica principale di *Nest.js* è la modularità, che promuove la suddivisione dell'applicazione in moduli, consentendo di organizzare il codice in unità funzionali e riutilizzabili.
Di seguito i concetti base su cui si basa l'architettura:

- ***Module***: rappresenta un'unità organizzativa dell'applicazione che contiene un gruppo di elementi correlati come *Controller*, *Service* e *Provider*. Questa struttura modulare favorisce la separazione delle responsabilità rendendo il codice più leggibile;
- ***Controller***: sono interfacce tra la rete e la logica dell'applicazione responsabili della gestione delle richieste [G]HTTP in ingresso. Ogni *Controller* è associato a un percorso specifico e a uno o più metodi che rappresentano le diverse azioni eseguibili sul percorso;
- ***Service***: contiene la logica di *business* dell'applicazione. I *Service* si occupano della gestione dei dati e dell'interazione con le risorse esterne [2].

Capitolo 5

Codifica

In questo capitolo sono descritti i componenti e le funzionalità sviluppate durante il progetto di stage. Per questioni di chiarezza e di ordine, questo capitolo è suddiviso in due parti: la prima rappresenta la parte frontend, mentre la seconda rappresenta la parte backend.

5.1 Codifica frontend

Il capitolo descrive la fase di codifica relativa al progetto di stage, focalizzandosi sullo sviluppo del frontend. Questa sezione riguarda la realizzazione del portale destinato alla consultazione delle *API*.

5.1.1 Utils

La cartella *Utils* contiene una serie di file *TypeScript* che rappresentano funzioni, *utilities* o moduli che forniscono funzionalità di supporto a varie parti dell'applicazione. Questi file sono progettati per semplificare compiti ripetitivi, astrazioni complesse o per fornire funzionalità condivise in più componenti. Ogni *utility* è accompagnata da un file di test che ne verifica il corretto funzionamento.
Di seguito sono elencate le *utilities* che ho sviluppato durante il progetto di stage.

5.1.1.1 Auth

Il file *Auth* è una *utility* per la gestione dell'autenticazione utilizzando come supporto la libreria *Azure MSAL*. Essa è utile per interfacciarsi all'autenticazione verso il servizio *Azure AD*.
Auth è composto da più metodi, a partire dalla funzione di login fino alla funzione di acquisizione del *token* di accesso.
Una funzione importante che ho implementato con l'aiuto della libreria *Azure MSAL* è

la funzionalità di recupero del *token* in modo automatico con l'implementazione del *refresh token*. Essa segue la procedura seguente:

1. Innanzitutto viene controllata la *cache* nell'archivio del *browser* per verificare se esiste un *token* di accesso ancora valido per la sessione corrente. Se viene trovato, il *token* viene restituito;

2. In caso il *token* sia scaduto o non esista proprio, viene tentato l'utilizzo del *refresh token* per ottenere un nuovo *token* di accesso;

3. In caso siano passate ventiquattro ore, ovvero la validità del *refresh token*, la libreria *Azure MSAL* apre un [G] hidden iframe per richiedere un nuovo codice di autorizzazione usando la sessione corrente dell'utente, ottenendo quindi un nuovo *token* di accesso e di aggiornamento.

Questo procedimento per il recupero del *token* con lo scopo di mantenere la sessione attiva, può non funzionare correttamente nei seguenti casi:

- L'utente ha cambiato la password, oppure
- Il browser blocca i *cookies* di terze parti, che prevengono l'utilizzo del *hidden iframe* per il recupero del *token*.

In questi casi, si verifica un errore che viene catturato tramite un *try catch* e viene utilizzato il metodo di login con *pop-up* per effettuare nuovamente l'autenticazione [17].

5.1.1.2 Debounce

Il file *Debounce* è una *utility* che ho creato per ritardare l'esecuzione di una funzione fino a quando non si verifica un certo intervallo di tempo in cui non vengono eseguite chiamate ad essa.
Questa *utility* è stata utilizzata per ritardare l'esecuzione della chiamata *POST* per il filtraggio dei risultati della ricerca. In questo modo, quando l'utente digita nella barra di ricerca, la chiamata *POST* viene eseguita ogni 300 millisecondi, ignorando le chiamate precedenti.

5.1.1.3 EndpointApiCall

L'*utility EndpointApiCall* è una funzione che ho creato per gestire la chiamata all'*endpoint* del backend per ottenere i dati relativi alle *API* disponibili nel sistema. Questa chiamata necessita di un *token* di autenticazione valido per la sessione corrente, necessario per evitare chiamate non autorizzate.

5.1.1.4 GetClients

L'*utility GetClients* è una funzione progettata per effettuare una richiesta *GET* al backend, al fine di recuperare l'elenco dei *client-id* disponibili nel sistema. Questa funzione è stata utilizzata per popolare la lista dei *clients* disponibili per la ricerca, in modo da poterne selezionare uno tra quelli presenti, in base all'ambiente di sviluppo. La chiamata necessita di un *token* di autenticazione valido per la sessione corrente per evitare chiamate non autorizzate.

5.1.1.5 GetResults

L'*utility getResults* è una funzione progettata per effettuare una richiesta *POST* al backend utilizzata per ottenere i risultati della ricerca. La logica della ricerca nel mio progetto è spostata lato backend, in modo da rispettare la *best practice* usata in azienda. La chiamata è autenticata e viene utilizzata insieme alla funzione di *debounce* per ritardare l'esecuzione della chiamata *POST*. Come *body* della chiamata viene passato il testo inserito dall'utente nella barra di ricerca.

5.1.1.6 MsGraphApiCall

L'*utility MsGraphApiCall* contiene al suo interno due funzioni che ho creato per effettuare chiamate alle *API* di [G]Microsoft Graph. La prima funzione è una chiamata *GET* che viene utilizzata per ottenere i dati relativi all'utente che ha effettuato l'accesso al portale. La seconda funzione è sempre una chiamata *GET* ad un altro *endpoint* di *Microsoft*, per ottenere informazioni secondarie sull'utente, come ad esempio l'immagine del profilo. Entrambe le chiamate necessitano di un *token*, diverso dal *token* utilizzato per le chiamate al backend. Infatti, questo *token* è specifico per le chiamate verso *Microsoft Graph*.

5.1.1.7 SwaggerUtils

L'*utility SwaggerUtils* contiene al suo interno una moltitudine di funzionalità utili per la gestione della documentazione delle *API* e la sua visualizzazione all'interno del *MainContent*, spiegato nella sezione 5.1.6.2. Inoltre, al suo interno ho creato dei *plugin* che semplificano la visualizzazione della documentazione, permettendo di selezionare parti specifiche del file *YAML* da mostrare, come ad esempio la descrizione o la versione delle *API*. L'*utility* gestisce anche il *try it out* di un *endpoint* e le funzionalità relative ad esso, spostando la logica in un unico file centralizzato.

5.1.2 Config

La cartella *Config* contiene i file di configurazione per l'applicazione. Di seguito sono descritti i file che ho sviluppato durante il progetto di stage.

5.1.2.1 ConfigAuth

Il file rappresenta un modulo di configurazione per l'autenticazione. Esso contiene le configurazioni per l'autenticazione verso *Azure AD*, come il *clientId*, il *redirectUri* o l'*authority*.

5.1.3 Stores

La cartella *Stores* contiene i vari *store* utilizzati all'interno del progetto in *Vue.js*. Gli *store* sono utili per gestire lo stato dell'applicazione, in modo da poterlo condividere tra più componenti.
Di seguito sono descritti gli *store* che ho sviluppato durante il progetto di stage.

5.1.3.1 Auth

Lo *store Auth* è stato creato per gestire lo stato dell'autenticazione dell'utente e l'interazione con un servizio di autenticazione esterno. Esso utilizza al suo interno l'*utility Auth* (descritta nella sezione 5.1.1.1), che include variabili reattive per gestire lo stato dell'autenticazione e la gestione dei *token*.
Ho scritto lo *store Auth* utilizzando uno stile di programmazione basato su funzioni [G]closure di *JavaScript*. Questo approccio consente di incapsulare lo stato dell'autenticazione garantendo un maggiore controllo sull'accesso alle variabili e alle funzioni all'interno del modulo.

5.1.3.2 AppState

Lo *store AppState*, a differenza del precedente, utilizza *Pinia*, una libreria apposita per la gestione dello stato. Esso è utilizzato per la gestione delle *API* e dei *client-id* disponibili nell'applicazione. Infatti lo *store* inizializza le variabili reattive per la gestione delle *API* e dei *client-id*, effettuando una chiamata al backend utilizzando le *utilities* apposite definite nel capitolo 5.1.1.

5.1.4 Router

La cartella *Router* contiene le rotte definite nel progetto, ovvero i percorsi che l'utente può visitare all'interno dell'applicazione, e un *helper* per la gestione del *router*.

5.1.4.1 CustomNavigation

Il file *CustomNavigation* rappresenta una classe con due metodi: il primo è utilizzato nel flusso di autenticazione e più precisamente nelle richieste con *pop-up*, mentre il secondo è utilizzato per convertire qualsiasi [G]URI di reindirizzamento completo in un *URI* di reindirizzamento relativo, in modo che il *Router* possa gestire correttamente i *redirect* in modo sicuro.

5.1.5 Views

La sezione *Views* contiene tutte le pagine dell'applicazione. Di seguito sono descritte le *Views* che ho sviluppato durante il progetto di stage.

5.1.5.1 LoginView

La *LoginView* è la prima pagina con cui l'utente interagisce (in figura 5.1). Essa contiene un bottone che permette di effettuare il login tramite *pop-up*, utilizzando un *account Microsoft 365*. La pagina consente di accedere al portale ed è l'unica di tutto il progetto che non richiede l'autenticazione.
Quando un utente non autenticato tenta di accedere a qualsiasi altra pagina del portale, viene sempre reindirizzato automaticamente a questa *View*. La pagina al suo interno contiene un unico componente, ovvero il *LoginButton*, descritto nella sezione 5.1.6.8.

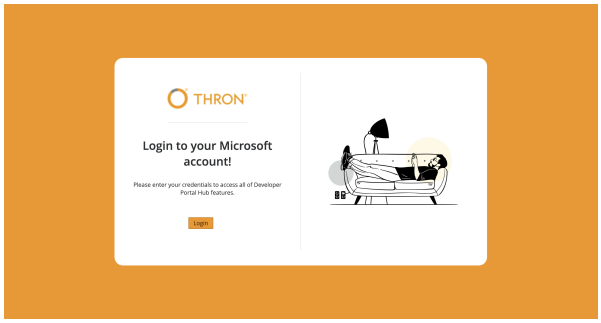


Figura 5.1: LoginView

Inoltre, per permettere un uso comodo anche in schermi di piccole dimensioni, o comunque utilizzando una scheda del *browser* ristretta, ho sviluppato un *design responsive* (in figura 5.2), che permette di visualizzare il contenuto in maniera ottimale anche su spazi ridotti. Ciò secondo me è necessario perché trattandosi di un portale per la consultazione di documentazione, è possibile che l'utente lo utilizzi in combinazione con altre schede del *browser*.

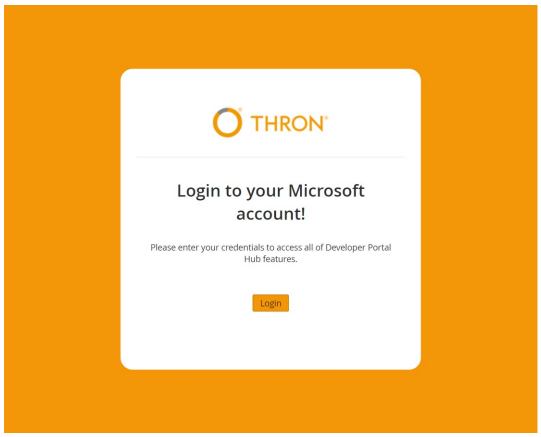


Figura 5.2: LoginView responsive

5.1.5.2 HomeView

La *HomeView* rappresenta la prima pagina visibile dopo aver effettuato il login. Infatti, dopo che l'autenticazione si è conclusa con successo, l'utente viene reindirizzato a questa pagina (in figura 5.3). La *HomeView* è composta da tre sezioni principali: *HeaderNav*, *Sidebar* e *StartPage*. Quest'ultimo è il componente visibile subito dopo aver effettuato il login. Infatti quando l'utente inizia a navigare tra le varie *API*, la *StartPage* viene sostituita con il *MainContent*.

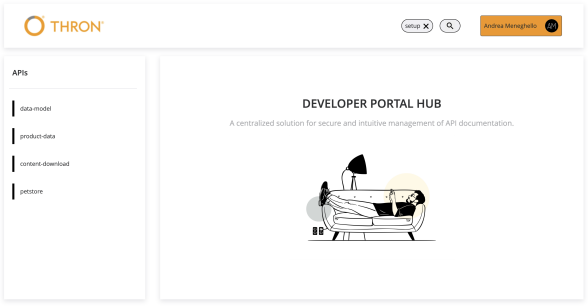


Figura 5.3: HomeView

Come la pagina di login, anche la pagina principale è stata sviluppata con un *design responsive* (in figura 5.4). Come si vede dall'immagine, si può notare che il menù laterale viene nascosto automaticamente ed è visibile cliccando l'apposito bottone a forma di *hamburger*.

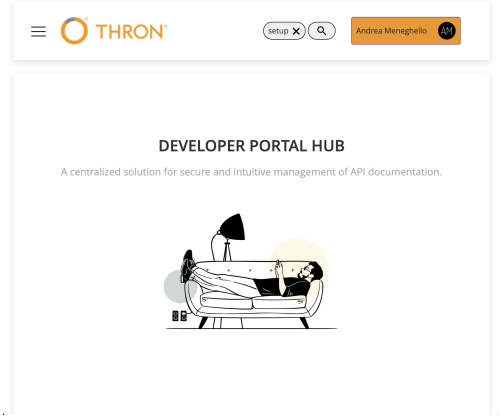


Figura 5.4: HomeView responsive

5.1.5.3 NotFoundView

Quando un utente tenta di accedere ad una pagina non esistente, viene reindirizzato automaticamente alla pagina di errore 404 (in figura 5.5). La pagina consiste in un messaggio di errore con un'immagine, dove viene informato l'utente che la pagina richiesta non esiste. Per tornare alla *HomeView* basta cliccare sul bottone *Home* e l'utente può continuare la navigazione all'interno del portale.

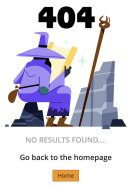


Figura 5.5: NotFoundView

5.1.6 Components

La sezione *Components* contiene tutti i componenti riutilizzabili creati durante lo sviluppo del progetto di stage.

5.1.6.1 HeaderNav

Il componente *HeaderNav* rappresenta la barra di navigazione superiore dell'applicazione (in figura 5.6). Essa contiene il logo dell'applicazione, la *chip* con il *client-id* corrente, il bottone di ricerca e il *popover* di logout. La barra viene sempre visualizzata una volta che l'utente ha effettuato il login all'interno del portale.
In caso di visualizzazione su schermi ridotti, l'*HeaderNav* ha un *design responsive*, dove viene aggiunto, nell'estremità sinistra, un bottone ad *hamburger* che permette di visualizzare la *SideBar* a comparsa.

Figura 5.6: HeadearNav responsive

5.1.6.2 MainContent

Il componente *MainContent* rappresenta il contenuto principale dell'applicazione. Esso infatti contiene la struttura per visualizzare i dettagli di ogni *API* (in figura 5.7), tramite l'aiuto della libreria *Swagger UI*. La struttura segue il *layout* di *Swagger Editor* per motivi di semplicità e uniformità. Inizia con una descrizione iniziale, prosegue con la lista degli *endpoint* disponibili e conclude con l'elenco dei modelli. I colori utilizzati sono uguali a quelli utilizzati in un comune *Swagger Editor* per dare un senso di familiarità all'utente ed è stata una richiesta avanzata dal team. Nell'angolo destro è stato aggiunto un bottone per il download della documentazione, discusso nella sezione 5.1.6.6.

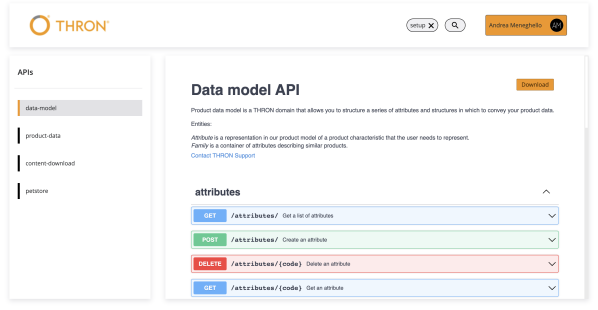


Figura 5.7: MainContent

Cliccando su uno degli *endpoint* disponibili, è presente una sezione dedicata per la visualizzazione dei dettagli di quest'ultimo (in figura 5.8). I parametri e il *try it out* sono gestiti all'interno della *utility SwaggerUtils* per una migliore suddivisione della logica.

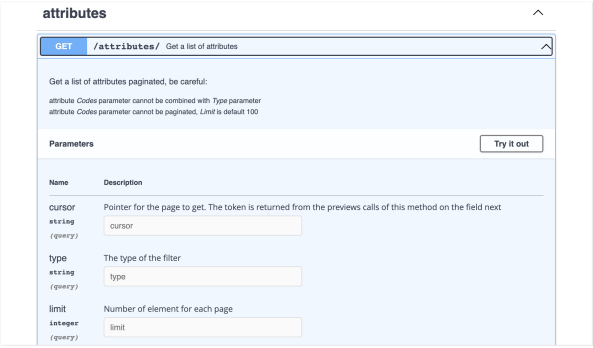


Figura 5.8: Try it out di un endpoint

Inserendo i parametri e facendo quindi il *try it out* viene visualizzato il risultato della chiamata (in figura 5.9).

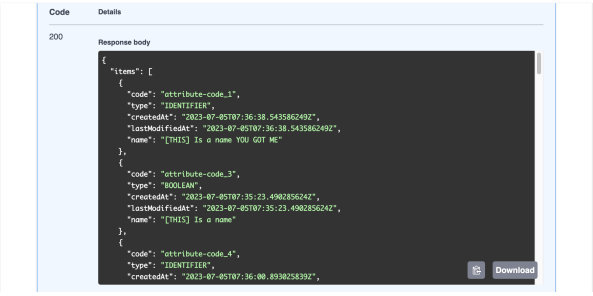


Figura 5.9: Risposta di un endpoint

Inoltre al di sotto è presente anche una lista delle possibili risposte che l'*endpoint* può restituire (in figura 5.10).

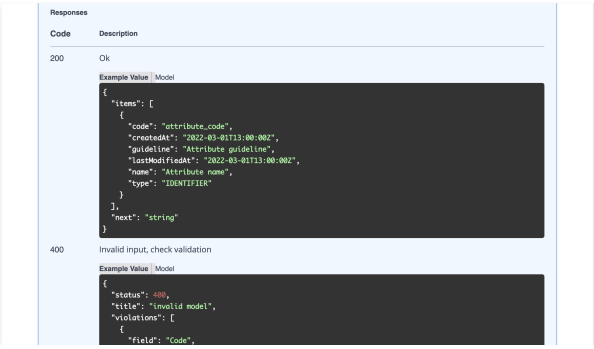


Figura 5.10: Lista di risposte di un endpoint

Nel caso di chiamate con necessità del parametro *client-id*, ho gestito tramite l'*utility SwaggerUtils* l'*autofill* di esso. Infatti, il campo viene compilato con il *client-id* presente nella *chip*, evitando che si inseriscano valori non presenti per quell'ambiente di sviluppo.

5.1.6.3 SideBar

Il componente *SideBar* rappresenta la barra di navigazione laterale dell'applicazione che contiene la lista di tutte le *API* disponibili nel portale (in figura 5.11). Attraverso la barra è possibile navigare tra le varie *API*, andando a visualizzare i dettagli di ognuna di esse. Il componente è stato sviluppato utilizzando un *design responsive*, che permette di visualizzare il contenuto in maniera ottimale anche su spazi ridotti, infatti il menù si nasconde automaticamente e per visualizzarlo basta cliccare l'apposito bottone a forma di *hamburger* situato nell'*HeaderNav*.

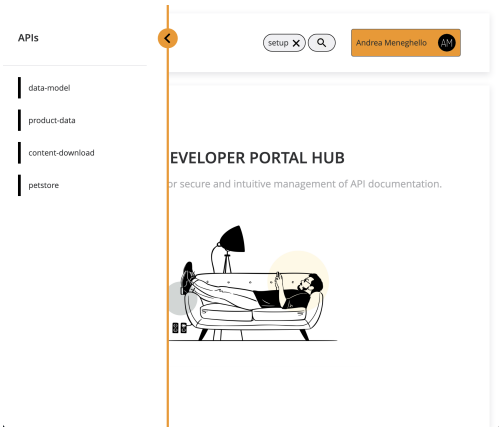


Figura 5.11: Design responsive SideBar

5.1.6.4 StartPage

Il componente *StartPage* rappresenta la pagina iniziale di benvenuto dell'applicazione ed è la prima schermata che l'utente visualizza dopo aver effettuato il login con successo. Essa contiene il nome del portale, una breve descrizione e un'immagine. Per iniziare la navigazione nel portale, basta usare la *SideBar*.

5.1.6.5 SwaggerLoader

Il componente *SwaggerLoader* rappresenta il *loader* di caricamento per il *MainContent* (in figura 5.12). Esso rappresenta un caricamento di stile *skeleton*, ovvero un caricamento di contenuto che simula il contenuto finale, mentre questo viene caricato, attraverso uno sfondo dinamico grigio. Il componente ha la stessa struttura del *MainContent*, quindi segue lo stesso *design responsive*. Il *loader* è visibile ogni volta che l'utente clicca su una delle *API* disponibili nella *SideBar*, e scompare una volta che il contenuto è stato caricato correttamente.

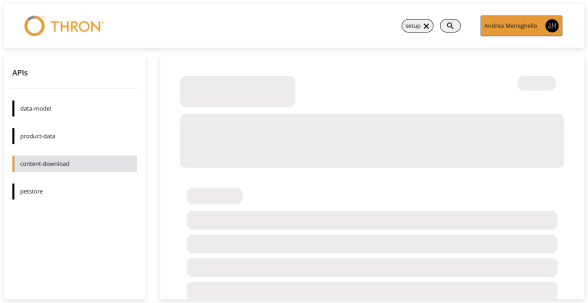


Figura 5.12: SwaggerLoader

5.1.6.6 DownloadButton

Il componente *DownloadButton* rappresenta il bottone di download presente in ogni pagina di dettaglio dell'*API* specifica. Esso è stato sviluppato rispettando il *design system* aziendale, infatti non è un componente *custom* creato da zero, ma è stato sviluppato utilizzando il componente *Button* della libreria *THRON Components*.
Il componente dà la possibilità di scegliere la grandezza, il testo del bottone e le icone da visualizzare. Il componente permette all'utente di scaricare la documentazione dell'*API* in formato *YAML*, con un nome predefinito a seconda del servizio scaricato. Una volta cliccato, il bottone apre una nuova finestra del *browser* e il file viene scaricato automaticamente.

5.1.6.7 LogoutButton

Il componente *LogoutButton* rappresenta il bottone di logout presente nella barra di navigazione superiore dell'applicazione ed è stato sviluppato rispettando il *design system* aziendale, infatti utilizza la libreria *THRON Components* (in figura 5.13). Nello specifico è stato chiamato bottone per comodità, ma in realtà si tratta di un *popover*, ovvero un componente a tendina che si apre tramite l'*hover* del mouse. Esso permette all'utente di effettuare il logout dal portale tramite un *pop-up*, con successivo reindirizzamento alla pagina di login.

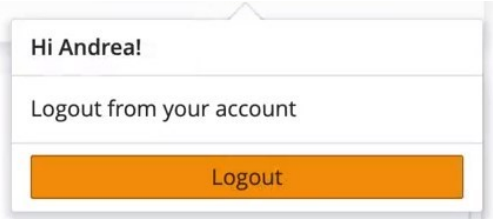


Figura 5.13: LogoutButton

5.1.6.8 LoginButton

Il componente *LoginButton* rappresenta il bottone di accesso al portale presente nella pagina di login dell'applicazione. Esso è stato sviluppato rispettando il *design system* aziendale. Il componente apre un *pop-up* in cui è possibile inserire le credenziali (e-mail e password) per effettuare l'accesso tramite *Microsoft 365*. Una volta effettuato il login, l'utente viene reindirizzato alla pagina principale del portale.

5.1.6.9 SearchButton

Il componente *SearchButton* rappresenta il bottone di ricerca presente nella barra di navigazione superiore dell'applicazione. Si tratta di un componente *custom* creato da me, utilizzato per aprire la barra di ricerca globale del portale.

5.1.6.10 SearchBar

Il componente *SearchBar* rappresenta la barra di ricerca globale dell'applicazione (in figura 5.14). Essa consiste in una barra dove poter inserire il *client-id* o il nome dell'*API* da cercare, con un'icona di ricerca a fianco. Per una migliore esperienza di ricerca, è necessario utilizzare il componente insieme al componente *Autocomplete* descritto nella sezione 5.1.6.11. Infatti i componenti insieme, costituiscono la ricerca all'interno del portale. Inoltre La *SearchBar* è accessibile tramite tastiera, infatti utilizzando il comando *Ctrl + B* è possibile aprire la barra di ricerca.

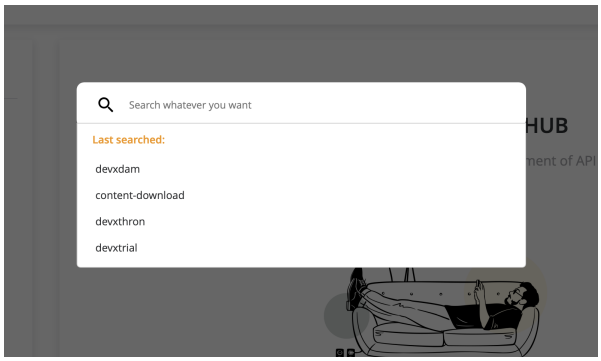


Figura 5.14: SearchBar

5.1.6.11 Autocomplete

Il componente *Autocomplete* rappresenta l'elenco dei risultati visualizzato sotto la barra di ricerca dell'applicazione (in figura 5.15). Essa consiste in una lista di suggerimenti dinamici che vengono visualizzati in base al testo inserito dall'utente. In caso il campo di ricerca della *SearchBar* sia vuoto, viene mostrata la lista degli ultimi quattro termini cercati, se presenti. In caso contrario, quando è presente un termine nella barra di ricerca, viene visualizzata una lista di suggerimenti divisa per categoria. Nel caso in cui la ricerca non abbia prodotto risultati, viene visualizzato un messaggio di errore gestito tramite il componente *SnackBar* descritto nella sezione 5.1.6.16.
La logica della ricerca è completamente sviluppata lato *backend* per estrapolare il componente di ricerca e renderlo riutilizzabile in altri progetti. Essendo un componente *custom*, ho aggiunto la possibilità di navigare il menù e selezionare un suggerimento tramite tastiera, funzionalità obbligatoria per rendere la ricerca accessibile anche a persone con disabilità.
Dopo aver selezionato un suggerimento, viene aperta la pagina di dettaglio dell'*API* corrispondente se l'utente ha scelto un'*API* specifica. In caso contrario, il *client-id* corrente viene impostato e la *Chip* (descritta nella sezione 5.1.6.12) viene aggiornata con il nuovo valore. Il componente infine supporta la gestione di più liste di elementi per la ricerca globale, non limitandosi a soli due elenchi. Questo significa che è possibile avere un numero variabile di liste di elementi con cui effettuare la ricerca globale.

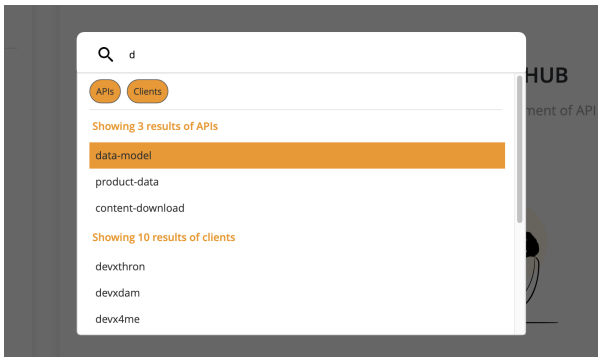


Figura 5.15: Autocomplete

5.1.6.12 Chip

Il componente *Chip* rappresenta la *chip* situata nella barra di navigazione superiore dell'applicazione (in figura 5.16). Essa è utile a visualizzare il *client-id* corrente, ovvero il *client-id* che l'utente ha selezionato tramite la barra di ricerca globale. Se l'utente non ha ancora effettuato alcuna selezione, viene visualizzato il *client-id* predefinito dell'ambiente di sviluppo, ovvero *devxsetup* per l'ambiente di *Develop*, *qaxsetup* per l'ambiente di *Quality* e *setup* per l'ambiente di *Production*. Inoltre è possibile reimpostare il *client-id* corrente cliccando sull'icona di reset a forma di *X*, che riporta la *chip* al valore di default. In caso il *client-id* fosse già al valore predefinito, viene visualizzato un messaggio di errore tramite il componente *SnackBar* (descritto nella sezione 5.1.6.16).

Figura 5.16: Chip

5.1.6.13 Filter

Il componente *Filter* rappresenta il bottone per filtrare la lista dei risultati della ricerca, all'interno del componente *Autocomplete*. Quando il bottone viene cliccato, lo sfondo diventa bianco e la lista corrispondente al filtro cliccato viene nascosta visivamente. Cliccando di nuovo sullo stesso filtro, il bottone torna arancione e la lista corrispondente al filtro viene nuovamente mostrata.

5.1.6.14 OptionList

Il componente *OptionList* rappresenta la lista di opzioni generiche, visualizzate all'interno della *SideBar*. Nel mio progetto, ho utilizzato questo componente per mostrare l'elenco completo delle *API* disponibili nel sistema.

5.1.6.15 OptionListItem

Il componente *OptionListItem* rappresenta un singolo elemento della lista di opzioni generiche, visualizzata all'interno della *SideBar*. Nel mio progetto, un *item* rappresenta una singola *API*. Quando l'utente sceglie un'opzione, viene aperta la pagina di dettaglio dell'*API* selezionata.

5.1.6.16 SnackBar

Il componente *SnackBar* è responsabile della visualizzazione dei messaggi di errore o di successo nell'applicazione (in figura 5.17). All'interno del portale è utilizzata per comunicare all'utente che un termine cercato non ha prodotto risultati, che il *client-id* è stato resettato con successo oppure per informare l'utente che il *client-id* è già al valore di default. Ho creato il componente con l'idea di renderlo riutilizzabile per diversi scopi, infatti è possibile modificare vari aspetti come il colore, il tipo di icona, la durata del messaggio, il testo da visualizzare, la posizione del componente e la dimensione del componente.

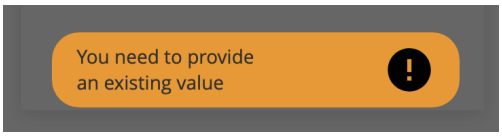


Figura 5.17: SnackBar

5.1.6.17 Loader

Il componente *Loader* rappresenta il sistema di caricamento principale dell'applicazione (in figura 5.18). Esso rappresenta un caricamento stile *skeleton*, ovvero un caricamento che simula la presenza di contenuti mentre questi vengono caricati, attraverso uno sfondo dinamico grigio. Il componente è stato utilizzato per il caricamento della struttura della pagina, ovvero è composto da tre sezioni: *HeaderNav*, *SideBar* e *MainContent*. Il *loader* è visibile subito dopo aver effettuato il login o quando la pagina viene ricaricata, e scompare una volta che i dati sono stati caricati correttamente. Inoltre il componente è stato sviluppato utilizzando un *design responsive*, infatti segue la stessa struttura della pagina, andando a nascondere la sezione *SideBar* in caso di schermi di piccole dimensioni. Ho preferito utilizzare un caricamento di tipo *skeleton*

rispetto ad un classico *loader* circolare, perché secondo me migliora l'esperienza utente conferendo una sensazione migliore durante l'attesa del caricamento dei dati.

Figura 5.18: Loader

5.2 Codifica backend

Il capitolo descrive la fase di codifica relativa al progetto di stage, focalizzandosi sullo sviluppo del backend. Questa sezione è dedicata alla discussione della funzionalità del *middleware* e alla creazione dei servizi necessari per gestire le *API*, i *client-id* e i risultati della ricerca.

5.2.1 Middleware

Il *middleware* è un meccanismo che permette di eseguire delle operazioni specifiche durante il ciclo di vita di una richiesta *HTTP*. Nel mio caso ho utilizzato questa funzionalità per eseguire parte della logica prima che la richiesta venga inoltrata al *Controller* e generi una risposta. Il *middleware* è stato utilizzato per due scopi principali: la validazione del ᴳtoken JWT di autenticazione e l'acquisizione del *supertoken* utile per eseguire delle chiamate. Basandoci sul ciclo di vita, il *middleware* intercetta le richieste, effettua le funzionalità appena descritte e successivamente inoltra la richiesta al *Controller* che si occupa di restituire una risposta [13]. Quindi ogni chiamata verso gli *endpoint* che ho sviluppato passa prima dal *middleware*. Questo perché ho specificato nel modulo principale dell'applicazione tutte le rotte che devono essere gestite dal *middleware*, ovvero i *Controller* di *client-id*, *API* e *search*.

5.2.1.1 Validazione token JWT

La prima funzione implementata nel *middleware* è la validazione del *token JWT* di autenticazione. Questa funzionalità è necessaria perché all'interno del portale è presente un sistema di autenticazione, quindi sicuramente le chiamate verso i servizi *backend* creati sono protetti dal login del portale. La criticità consiste nel fatto che un servizio può essere accessibile direttamente senza la necessità di passare attraverso il portale, poiché ciascun *endpoint* dispone di un *URL* pubblico. La soluzione che ho implementato è stata quella di validare il *token* di autenticazione verificando che qualsiasi chiamata ai miei servizi fosse effettuata da un utente autenticato, avente un token valido nell'*header* della richiesta. Per verificare il *token* ho effettuato i seguenti passaggi:

1. Verifico che il *token* di autenticazione sia presente nell'*header* della richiesta. In caso non lo sia viene restituito un errore 401;
2. Il *token* viene decodificato e viene salvato solo il suo *header*. In caso il risultato salvato non sia valido, viene restituito un errore 401;
3. Estraggo il campo *KID*, che identifica quale chiave pubblica viene utilizzata per la verifica;

4. Eseguo una *GET* verso l'*endpoint* di *Microsoft AAD* per ottenere le informazioni di configurazione di [G]OpenID Connect (*OIDC*). Queste informazioni includono gli *URL* da dove è possibile ottenere le chiavi pubbliche per la verifica del *token*;
5. Estraggo l'*URL* che specifica dove trovare le chiavi pubbliche *JSON Web Key Set* (*JWKS*) e faccio una seconda *GET* verso l'*endpoint* appena descritto;
6. Effettuo un controllo con la chiave pubblica corrispondente all'intestazione del *token JWT* (identificata dal campo '*kid*') nell'elenco delle chiavi ottenute da *Azure AD*. Nel caso non ci sia nessuna corrispondenza viene restituito un errore 401;
7. La chiave pubblica viene convertita e formattata in un [G]certificato X.509 valido;
8. Effettuo la validazione del *token* utilizzando la chiave pubblica convertita. Il risultato contiene il *payload* del *token JWT* se la verifica ha successo o viene generato un errore se la verifica fallisce.

5.2.1.2 GET Supertoken

La seconda funzione implementata nel *middleware* è l'acquisizione del *supertoken*, che avviene solamente se la validazione del *token* appena discussa non genera errori. Per ottenere il *supertoken* è necessario effettuare una chiamata verso un *endpoint* aziendale, che restituisce il *supertoken* in base ad un nome utente passato come parametro. Per prima cosa ho creato dei parametri, aggiungendo l'*username* con il valore che mi è stato fornito dall'azienda. Successivamente effettuo una *POST* verso l'*endpoint* aziendale che mi ritorna il *supertoken*, e in caso di criticità viene restituito un errore.
Il *token* appena ottenuto viene aggiunto all'*header* della richiesta, in modo tale che ogni chiamata verso i miei *endpoint* ha il *supertoken* aggiunto automaticamente.

5.2.2 Endpoints

Le seguenti sezioni descrivono gli *endpoint* che ho creato all'interno del mio progetto. Tutti gli *endpoint* sviluppati sono utilizzati all'interno del progetto *frontend* descritto nel capitolo 5.1. Come descritto nel capitolo dell'architettura (in sezione 4.3.2), ogni *endpoint* creato è formato da un *Module*, un *Controller* e un *Service*.

5.2.2.1 Client-id

L'*endpoint* rappresenta il punto di accesso per ottenere la lista dei *client-id* disponibili nel sistema. È definito dal file *Module*, dove vengono specificati i vari *Controller* e *Service* utilizzati. Nel *Controller* è presente l'*endpoint* '*/clients*', ovvero un metodo *GET* che restituisce la lista dei *client-id* chiamando il metodo '*getAllClients*' del *Service*. All'interno del metodo appena citato, viene chiamato un altro *endpoint* sviluppato

dall'azienda, che restituisce la lista dei *client-id* disponibili per l'ambiente di sviluppo in cui ci si trova. Per chiamare questo *endpoint*, sono state configurate all'interno dell'applicativo le variabili d'ambiente. Inoltre la chiamata necessita del *supertoken*, che grazie al *middleware* è aggiunto automaticamente all'interno dell'*header* della chiamata. Il metodo ritorna un *array* di stringhe, dove ogni stringa rappresenta un *client-id* disponibile nel sistema.

5.2.2.2 API

L'*endpoint* rappresenta il punto di accesso per ottenere la lista di *API* disponibili nel sistema. È definito dal file *Module*, dove vengono specificati i vari *controller* e *Service* utilizzati. Nel *Controller* è presente l'endpoint '*/apis*', ovvero un metodo *GET* che restituisce la lista delle *API* chiamando il metodo '*getAllApis*' del *Service*. All'interno del metodo appena citato, è presente la logica necessaria per restituire la lista completa di *API* disponibili. Il metodo utilizza un *endpoint* aziendale per ottenere la lista di *API* dei servizi THRON. Il metodo restituisce un *array* di oggetti, dove ogni oggetto è rappresentato dal nome del servizio e dall'*URL* corrispondente.

5.2.2.3 Search

L'*endpoint* rappresenta il punto di accesso per ottenere la lista di suggerimenti utilizzata per la ricerca globale del portale. È definito dal file *Module*, dove vengono specificati i vari *Controller*, *Service* e gli *Imports* utilizzati. Quest'ultimi non sono altro che dei moduli che vengono utilizzati all'interno del *Controller*. Nel mio caso ho importato il modulo *APIModule* relativo alle *API* e il modulo *ClientModule* relativo ai *client-id*. Nel *Controller* è presente l'*endpoint* '*/results*', ovvero un metodo *POST* che restituisce la lista di suggerimenti in base ad un parametro di ricerca inserito nel *body* della chiamata. Il metodo in primis ottiene le *API* e i *client-id* utilizzando i moduli importati. Successivamente viene chiamato il metodo '*getResults*' del *Service*, al quale vengono passati come parametri il termine di ricerca, la lista delle *API* e la lista dei *client-id*. All'interno della funziona appena citata, è presente tutta la logica di filtraggio dei risultati.
Il servizio che ho creato restituisce una lista di risultati, ognuno dei quali è un oggetto contenente il nome del gruppo filtrato ed una lista associata ad esso. Inoltre, ho impostato un limite massimo di dieci risultati per ciascun gruppo. Questo limite serve a evitare di avere una lista eccessivamente lunga di elementi, che potrebbe risultare difficile da gestire o visualizzare.

Capitolo 6

Attività di verifica e validazione

In questo capitolo sono descritti i processi di verifica e validazione del prodotto. In particolare sono illustrati i test implementati, i risultati ottenuti e le possibili migliorie future del prodotto.

6.1 Test di unità

Durante lo svolgimento del progetto di stage sono stati implementati dei test di unità, in modo da verificare il corretto funzionamento dei singoli componenti e delle singole funzionalità implementate.
Ciascun test di unità è identificato da un codice univoco, che segue la seguente notazione:

TU-[Codice]

dove:

- **TU**: definisce che il test è di unità;
- **Codice**: definisce un numero progressivo che identifica il test.

Ad ogni test di unità corrisponde un risultato, che può essere di due tipi:

- **S**: il test è stato superato;
- **NS**: il test non è stato superato.

Le tabelle 6.1, 6.3 e 6.5 riportano i test di unità implementati per le *Views*, i *Components* e le *Utilities*.

Views

Tabella 6.1: Tabella del tracciamento dei test di unità delle Views

Codice	Oggetto	Descrizione	Stato
TU-1	LoginView	Verifica del corretto funzionamento dell'autenticazione, con successivo *redirect* alla pagina principale	S
TU-2	HomeView	Verifica la corretta visualizzazione della pagina e la corretta validazione dei parametri dell'*URL*	S
TU-3	NotFoundView	Verifica la corretta visualizzazione della pagina e il funzionamento del reindirizzamento alla *home*	S

Components

Tabella 6.3: Tabella del tracciamento dei test di unità dei Components

Codice	Oggetto	Descrizione	Stato
TU-4	HeaderNav	Verifica la corretta visualizzazione della barra, verificando inoltre la visualizzazione dei dati dell'utente loggato	S
TU-5	MainContent	Verifica la corretta visualizzazione dei dati relativi all'*API* e il corretto funzionamento del *try it out* di un *endpoint*	S
TU-6	SideBar	Verifica la corretta visualizzazione della barra laterale, mostrando le *API* disponibili correttamente	S
TU-7	StartPage	Verifica la corretta visualizzazione della pagina, controllando i parametri nell'*URL*	S
TU-8	DownloadButton	Verifica il corretto funzionamento del download di una singola *API*	S
TU-9	LogoutButton	Verifica il corretto funzionamento del logout con *pop-up* di un utente	S
TU-10	LoginButton	Verifica il corretto funzionamento del login con *pop-up* di un utente	S
TU-11	SearchButton	Verifica il corretto funzionamento del bottone di ricerca	S

Continuazione della tabella 6.3			
TU-12	SearchBar	Verifica la corretta visualizzazione della barra di ricerca	S
TU-13	Autocomplete	Verifica la corretta visualizzazione della lista aggiornata in base all'input corrente nella barra di ricerca. Inoltre verifica l'accessibilità del menù a tendina	S
TU-14	Chip	Verifica la corretta visualizzazione del *client-id* corrente, verificando inoltre la funzionalità di reset	S
TU-15	Filter	Verifica il corretto funzionamento del filtraggio in base al filtro selezionato	S
TU-16	OptionList	Verifica la corretta visualizzazione delle *API* disponibili	S
TU-17	OptionListItem	Verifica il corretto funzionamento del reindirizamento all'*API* selezionata	S
TU-18	SnackBar	Verifica la corretta visualizzazione del messaggio di errore	S

Utils

Tabella 6.5: Tabella del tracciamento dei test di unità delle Utilities

Codice	Oggetto	Descrizione	Stato
TU-19	Auth	Verifica il corretta funzionamento di tutte le funzionalità riguardanti l'autenticazione	S
TU-20	Debounce	Verifica il corretto funzionamento del *delay*, verificando che un'azione sia eseguita solo dopo un determinato periodo di tempo	S
TU-21	EndpointsApiCall	Verifica il corretto funzionamento della chiamata *GET* all'*endpoint* creato lato server	S
TU-22	GetClients	Verifica il corretto funzionamento della chiamata *GET* all'*endpoint* creato lato server	S
TU-23	GetResults	Verifica il corretto funzionamento della chiamata *POST* all'*endpoint* creato lato server	S
TU-24	MsGraphApiCall	Verifica il corretto funzionamento delle chiamate verso gli *endpoint* di *Microsoft Graph*	S

6.2 Test di compatibilità cross-browser

In questa sezione sono introdotti i test di compatibilità *cross-browser*, che sono stati implementati per assicurare il corretto funzionamento del prodotto finale su tutti i browser più utilizzati.
Ciascun test di unità è identificato da un codice univoco, che segue la seguente notazione:

TB-[Codice]

dove:

- **TB**: definisce che il test è di compatibilità *cross-browser*;
- **Codice**: definisce un numero progressivo che identifica il test.

Ad ogni test di unità corrisponde un risultato, che può essere di due tipi:

- **S**: il test è stato superato;
- **NS**: il test non è stato superato.

La tabella 6.7 riporta i test di compatibilità *cross-browser* implementati. Tutti i test sono stati superati con successo, garantendo un'esperienza uniforme per gli utenti su diverse piattaforme e *browser*.

Tabella 6.7: Tabella del tracciamento dei test di compatibilità cross-browser

Codice	Oggetto	Descrizione	Stato
TB-1	Applicazione	Si verifica la corretta visualizzazione e il corretto funzionamento dell'applicazione sul *browser Microsoft Edge*	S
TB-2	Applicazione	Si verifica la corretta visualizzazione e il corretto funzionamento dell'applicazione sul *browser Google Chrome*	S
TB-3	Applicazione	Si verifica la corretta visualizzazione e il corretto funzionamento dell'applicazione sul *browser Mozilla Firefox*	S
TB-4	Applicazione	Si verifica la corretta visualizzazione e il corretto funzionamento dell'applicazione sul *browser Safari*	S
TB-5	Applicazione	Si verifica la corretta visualizzazione e il corretto funzionamento dell'applicazione sul *browser Brave*	S

Continuazione della tabella 6.7			
TB-6	Applicazione	Si verifica la corretta visualizzazione e il corretto funzionamento dell'applicazione sul *browser Opera*	S

6.3 Documentazione

Un obiettivo obbligatorio dello stage era quello di produrre una documentazione sul progetto svolto, sia tecnica che funzionale. La prima delle due è focalizzata sugli aspetti tecnici e implementativi del progetto, andando a rivolgersi principalmente agli sviluppatori. Questo tipo di documentazione infatti, include informazioni dettagliate su componenti e tecnologie utilizzate, in modo che una persona che deve iniziare a lavorare sul progetto possa farlo in autonomia. D'altro canto, la documentazione funzionale è orientata verso gli utenti finali del prodotto, ai clienti o agli *stakeholder*. Essa infatti, fornisce una panoramica degli scenari di utilizzo, delle interazioni dell'utente e delle risposte attese dal sistema, creando un guida che affronta i problemi più comuni.
La validazione della documentazione è stata effettuata tramite un confronto con il tutor aziendale, che ha fornito un *feedback* sulle parti da migliorare e su quelle da approfondire.
Nel dettaglio le documentazioni sono composte dalle seguenti caratteristiche:

- **Documentazione tecnica**: la documentazione in primis espone tutto ciò che è necessario per iniziare a lavorare da subito al progetto, come la locazione della *repository*, i link alle applicazioni distribuite nei vari ambienti di sviluppo o indicazioni su come effettuare il *deploy* del progetto.
  Successivamente vengono affrontati vari punti, come la struttura del progetto, i comandi per avviare il progetto in locale o informazioni ancora più dettagliate, come l'aggiunta di un nuovo *endpoint* all'interno del progetto backend. Grazie a tutte queste informazioni, un nuovo sviluppatore può lavorare in completa autonomia sul progetto;

- **Documentazione funzionale**: la documentazione funzionale è stata scritta in modo da essere comprensibile anche a persone che non hanno conoscenze tecniche. Essa infatti, fornisce una panoramica generale dell'utilizzo del progetto, fornendo una guida completa su come utilizzare le varie funzionalità che possono causare ad un utente dubbi o incertezze.

Infine, entrambe le documentazioni sono disponibili all'interno del [G] Confluence aziendale, in modo che sia facilmente accessibile a tutti i membri del team.

6.4 Collaudo

Durante il periodo di stage, sono stati organizzati degli incontri interni con il tutor aziendale a cadenza settimanale, in cui venivano discussi i progressi raggiunti e le criticità o dubbi riscontrati. Ciò ha permesso di avere un *feedback* costante sul lavoro svolto, garantendo una maggiore trasparenza.
Questi incontri settimanali erano un momento fondamentale per monitorare l'andamento del mio stage e per affrontare eventuali problematiche in modo tempestivo.

6.5 Presentazione finale

Nell'ultima settimana di stage è stata organizzata una presentazione finale, in cui ho illustrato il lavoro svolto e i risultati ottenuti. La presentazione è stata fatta davanti a tutta l'azienda, in modo che tutti i dipendenti potessero avere una panoramica completa del progetto.
L'esito della presentazione è stato più che positivo e non sono state evidenziate criticità a seguito delle domande poste dai presenti.
La presentazione ha contribuito ad una validazione finale del lavoro svolto ad alto livello.

6.6 Migliorie future

Al termine dell'attività di stage, sono state individuate alcune possibili migliorie future, che potrebbero essere implementate in futuro per migliorare il prodotto finale.
In particolare sono state individuate le seguenti:

- **Gestione di *openapi* più vecchie della 2.0**: attualmente il prodotto supporta solo *openapi* versione 2.0 e 3.0, ma sarebbe interessante valutare la possibilità di supportare anche versioni precedenti, in modo da rendere il prodotto più flessibile;
- **Valutare cambio di tecnologia a *Nuxt.js***: attualmente il prodotto è stato sviluppato con *Vue.js* e *Nest.js*, ma sarebbe interessante valutare un cambio di tecnologia verso *Nuxt.js*, che permette di sviluppare applicazioni *server-side rendered*, in modo da non avere due applicazioni separate per il frontend e il backend, ma un'unico progetto che comporta una migliore gestione e manutenibilità.

Capitolo 7

Conclusioni

Il progetto di stage svolto ha avuto come scopo la creazione di un portale per la consultazione di tutte le *API* messe a disposizione da *THRON*, in un'unica soluzione centralizzata. Il portale inoltre doveva permettere ad un utente di visualizzare i singoli *endpoint* di ogni servizio, fornendo la possibilità di provarli direttamente senza l'utilizzo di strumenti esterni. Essendo le *API* informazioni sensibili, il portale per essere navigato in tutte le sue funzionalità, richiedeva un sistema di autenticazione tramite *account Microsoft 365*. Per la realizzazione del progetto sono state affrontate varie fasi, partendo dall'analisi dei requisiti, dove sono stati definiti i casi d'uso e i requisiti del prodotto. In seguito, sono state analizzate le tecnologie da impiegare nella creazione del portale, con uno studio di esse, concentrandosi in particolare sul *framework Vue.js* e il *framework Nest.js*. È stata poi progettata l'architettura del sistema seguita dall'implementazione in tutte le sue parti.

In merito agli obiettivi prefissati per lo stage (in sezione 2.2), sono stati raggiunti tutti quelli obbligatori e desiderabili, mentre per quanto riguarda gli opzionali, sono stati raggiunti tutti ad eccezione di uno, ovvero la funzionalità di recupero automatico del *token* per l'utilizzo delle *API* direttamente dal portale. Più nello specifico, sono stati raggiunti i seguenti obiettivi obbligatori:

- **OB1**: Realizzazione di un portale che consenta la consultazione degli *OpenAPIs schemas* dei servizi pubblici e privati offerti da THRON;
- **OB2**: Rendere possibile l'utilizzo delle *API* direttamente dal portale (con inserimento manuale del *token* di autenticazione);
- **OB3**: Documentazione delle funzionalità implementate;
- **OB4**: Realizzazione di test di unità delle funzionalità implementate.

Sono stati raggiunti i seguenti obiettivi desiderabili:

- **DE1**: Implementare la funzionalità di recupero automatico degli *OpenAPI schemas*;
- **DE2**: Implementare la funzionalità di autenticazione al portale.

Infine, sono stati raggiunti i seguenti obiettivi opzionali:

- **OP1**: Implementare la funzionalità di download dello schema di uno specifico servizio (formato *YAML*).

Come accennato in precedenza, non è stato raggiunto l'obiettivo opzionale *OP2*, ovvero la funzionalità di recupero automatico del *token* di autenticazione per l'utilizzo della *API* direttamente dal portale. Questo obiettivo era presente nonostante non ci fosse ancora una vera soluzione applicabile, poiché anche in azienda non era ancora stata trovata una soluzione definitiva. La sfida principale consiste nel fatto che il *token* di autenticazione per testare le *API* varia a seconda del *client-id* scelto e ha una durata limitata. In azienda al momento, non sono disponibili servizi che permettono di risolvere questo problema. La soluzione adottata è stata l'inserimento manuale del *token*, una pratica utilizzata dal mio team finora e considerata più che accettabile.

7.1 Conoscenze acquisite

Il progetto di stage svolto mi ha permesso di acquisire nuove conoscenze e competenze, sia dal punto di vista tecnico che personale, andando a soddisfare le mie aspettative iniziali.
Innanzitutto, è stato molto stimolante esplorare in dettaglio i due prodotti aziendali *THRON Pim* e *THRON Dam Platform*, di cui non ero a conoscenza prima. Questo mi ha permesso di ottenere una comprensione completa del contesto operativo dell'azienda in cui ho svolto il mio stage.
Tra le competenze più significative acquisite, sicuramente spicca l'approfondimento e l'utilizzo del *framework Vue.js* per la parte frontend e del *framework Nest.js* per la parte backend del progetto. Questi strumenti al giorno d'oggi sono estremamente rilevanti nell'ambito di applicazioni web e la loro richiesta è in costante crescita. È stato interessante anche imparare il concetto di *middleware* del *framework Nest.js* e come questo possa essere utilizzato per la gestione delle richieste *HTTP*.
L'attività di stage mi è stata utile anche per consolidare ulteriormente le mie conoscenze del linguaggio di programmazione *TypeScript*, linguaggio che avevo già utilizzato all'interno del corso di laurea.
Ho scoperto nuove nozioni riguardanti l'infrastruttura che supporta un progetto aziendale e tutto ciò che riguarda la configurazione di essa. Questo mi ha permesso di ottenere una visione più completa del lavoro svolto, anche oltre l'attività dello sviluppo del portale. Ho imparato e utilizzato strumenti infrastrutturali che hanno semplificato

il mio lavoro, come i costrutti, e ho configurato file per la *build* e il *deploy* del prodotto comprendendo il flusso di lavoro utilizzato in azienda per lo sviluppo di progetti.
La competenza più importante acquisita durante l'esperienza di stage è stata la capacità di lavorare in modo efficace in un team e poter partecipare attivamente a tutte le attività, dalle riunioni quotidiane fino alle riunioni di *Sprint Retrospective*. Riguardo a quest'ultima attività, ho approfondito la metodologia *Scrum*, che avevo già utilizzato all'interno del corso di Ingegneria del Software, ma che non avevo ancora sperimentato in un ambiente lavorativo.

7.2 Valutazione personale

L'esperienza di stage rappresenta un capitolo fondamentale nel mio percorso di crescita personale e professionale. Durante questo periodo, ho avuto l'opportunità di acquisire conoscenze e competenze tecniche fondamentali nel campo dello sviluppo web, che saranno un pilastro essenziale per il mio futuro professionale. La possibilità di mettere in pratica le nozioni apprese durante il mio percorso di studi in un contesto lavorativo concreto è stato molto gratificante e ha contribuito in modo significativo al mio apprendimento.
In aggiunta, lavorare a stretto contatto con il team di sviluppo mi ha insegnato l'importanza delle dinamiche di gruppo e della comunicazione efficace tra i suoi membri per il raggiungimento degli obiettivi prefissati. Durante lo stage, ho acquisito una maggiore consapevolezza delle mie capacità e delle mie passioni professionali grazie all'interazione con il team, il che mi ha aiutato a identificare le mie aree di interesse e a delineare con maggior chiarezza la mia futura carriera professionale.
I due mesi trascorsi in azienda sono stati estremamente piacevoli, un fatto che va attribuito in parte all'azienda stessa, che ha creato un ambiente di lavoro stimolante e giovane. È stato un piacere partecipare alle attività ed eventi del team, sentendomi parte integrante di questo gruppo. Ho avuto l'opportunità di conoscere persone competenti e disponibili, che mi hanno sostenuto e guidato lungo tutto il mio percorso in azienda. Questo ha reso il mio team di lavoro un elemento chiave per il successo del mio stage.

In conclusione, valuto l'esperienza di stage in maniera estremamente positiva e gratificante. Sono soddisfatto del risultato che ho ottenuto, del cammino che ho percorso, delle competenze che ho acquisito e delle persone che ho avuto l'opportunità di conoscere. Questa esperienza ha arricchito il mio percorso professionale e sono profondamente grato per questa opportunità, determinato a sfruttarla al massimo nelle sfide future che mi attendono.

Glossario

Agile In ingegneria del software, con il termine *Agile* si indica un approccio metodologico che enfatizza la collaborazione e l'adattabilità nei processi di lavoro. L'*Agile* si basa su un insieme di principi descritti nell'*Agile Manifesto*, che promuove valori come la risposta ai cambiamenti dei requisiti o il coinvolgimento attivo del cliente. 1, 64

API In informatica con il termine *API*, *Application Programming Interface* (ing. interfaccia di programmazione di un'applicazione), si indica un insieme di regole, protocolli e strumenti che consentono a due software o sistemi di interfacciarsi e interagire tra loro in modo standardizzato. 5, 64

AWS *AWS* (*Amazon Web Services*) è una piattaforma di servizi *cloud computing* offerta da *Amazon*. 30, 64

AWS CloudFront *AWS CloudFront* è un servizio di *Content Delivery Network* (*CDN*) gestito da *Amazon*. Offre un modo semplice per distribuire contenuti web, immagini o altri file multimediali a livello globale. 32, 64

AWS CodeBuild *AWS CodeBuild* è un servizio di elaborazione gestito da *Amazon* che consente di automatizzare il processo di compilazione e distribuzione del codice sorgente delle applicazioni. 32, 64

Azure AD *Azure AD* (*Azure Active Directory*) è un servizio di gestione degli accessi fornito da *Microsoft*, all'interno della piattaforma *Azure*. 31, 64

backend In informatica con il termine *backend* si indica la parte di un'applicazione o di un sistema software che non è visibile all'utente e opera in secondo piano per gestire funzionalità come l'elaborazione dei dati, la gestione del database o l'autenticazione degli utenti. 2, 64

Bucket S3 In informatica un *bucket S3* rappresenta un contenitore di oggetti all'interno del servizio di archiviazione *Amazon S3*. È uno spazio di archiviazione dove è possibile archiviare qualsiasi tipo di dato, come file multimediali, documenti o dati di applicazioni. 32, 64

build In informatica con il termine *build* si indica il processo di compilazione e trasformazione del codice sorgente in un eseguibile. 2, 65

certificato X.509 In informatica con il termine *X.509* si indica uno standard per la gestione dei certificati digitali utilizzato per autenticare l'identità di entità digitali come siti web, server e utenti. 53, 65

closure In informatica con il termine *closure* si indica una funzione che è in grado di catturare e memorizzare il contesto in cui è stata dichiarata. Questo significa che una funzione *closure* può accedere alle variabili e dati esterni anche dopo che la funzione che li ha dichiarati è terminata. 39, 65

Composition API Nel *framework Vue.js*, la *Composition API* è un paradigma di sviluppo per la creazione di componenti. È un'alternativa alla *Options API* introdotta con la versione 3 di *Vue.js*. 27, 65

Confluence *Confluence* è una piattaforma di collaborazione e gestione delle conoscenze sviluppata da *Atlassian*. 59, 65

container In informatica con il termine *container* si indica un'ambiente isolato che contiene il codice e tutte le sue dipendenze, garantendo che l'applicazione funzioni in modo identico su qualsiasi ambiente in cui venga eseguita. 33, 65

DAM *DAM* (*Digital Asset Management*) è un insieme di tecnologie utilizzate per organizzare, gestire e distribuire risorse digitali come immagini e video. 1, 65

deploy In informatica con il termine *deploy* si indica l'atto di mettere in produzione o distribuire un'applicazione *software* in un ambiente operativo. 2, 65

docker In informatica con il termine *Docker* si indica una piattaforma di virtualizzazione a livello di contenitore che consente di creare, distribuire e gestire applicazioni e servizi in modo isolato. 32, 65

DOM In informatica con il termine *DOM* (*Document Object Model*) si indica una rappresentazione gerarchica di un documento *HTML*, *XML* o di altre risorse web. 28, 65

endpoint In informatica il termine *endpoint* indica un punto di accesso all'interno di una rete o di un servizio che può essere utilizzato per inviare e ricevere dati. 2, 65

ESM In informatica *ESM* (*ECMAScript Modules*) rappresenta uno standard per la gestione dei moduli in *JavaScript*. Sono un'implementazione comune che consente l'importazione ed esportazione di moduli con il vantaggio di migliorare la modularità del codice. 29, 65

Express.js In informatica *Express.js* rappresenta un *framework* per applicazioni web *Node.js* che fornisce un insieme di funzionalità per semplificare la creazione di applicazioni web e *API*. 29, 66

framework In informatica con il termine *framework* si indica una struttura predefinita e organizzata che fornisce un modello su cui basare e sviluppare applicazioni software. 2, 66

frontend In informatica con il termine *frontend* ci si riferisce alla parte di un'applicazione o di un sistema software che interagisce direttamente con l'utente. Il *frontend* è responsabile per la presentazione dell'interfaccia utente e per tutti gli aspetti visivi e interattivi dell'applicazione, con cui l'utente può interagire. 1, 66

hidden iframe In informatica con il termine *hidden iframe* si indica un elemento *HTML* che risulta invisibile all'utente e consente di caricare contenuti in modo asincrono. 37, 66

hot-reload In informatica con il termine *hot-reload* si indica una funzionalità comune adottata nei *framework*, che consente agli sviluppatori di apportare modifiche al codice sorgente in tempo reale mentre l'applicazione è in esecuzione, senza doverla riavviare. 29, 66

HTTP In informatica con il termine *HTTP* (*Hypertext Transfer Protocol*) si indica un protocollo di comunicazione che permette lo scambio di dati su Internet. 35, 66

lambda In informatica con il termine *lambda* si indica un servizio di elaborazione *serverless* fornito da *Amazon* che consente agli sviluppatori di eseguire codice senza la necessità di gestire server o infrastrutture sottostanti. 32, 66

Microsoft Graph *Microsoft Graph* è un servizio di *Microsoft* che consente di accedere ai dati e alle informazioni di *Microsoft 365*, consentendo l'integrazione con le applicazioni. 38, 66

mixin In *Sass* con il termine *mixin* si indica un costrutto che permette di definire un insieme di regole *CSS* che possono essere riutilizzate in più punti all'interno del foglio di stile. 29, 66

MVC In informatica con il termine *MVC* (*Model-View-Controller*) si indica un design pattern utilizzato nell'ambito dello sviluppo software per organizzare i componenti di un'applicazione. 34, 66

open-source Il termine *open-source* indica un software reso disponibile al pubblico con una licenza che consente l'accesso, la modifica e la sua libera distribuzione. 29, 67

OpenAPI schema In informatica con il termine *OpenAPI schema* si indica un insieme di standard e convenzioni utilizzato per la documentazione di servizi web *RESTful*. 6, 67

OpenID Connect In informatica *OIDC* (*OpenID Connect*) rappresenta un protocollo di autenticazione basato su *OAuth 2.0*, progettato per identificare e autenticare utenti in modo sicuro all'interno di applicazioni web. 53, 67

PIM *PIM* (*Product Information Management*) è un sistema utilizzato dalle aziende per raccogliere, gestire e distribuire informazioni dettagliate sui loro prodotti. 1, 67

pnpm in informatica *pnpm* (*Plug'n'Play Node Package Manager*) indica un sistema di gestione delle dipendenze per progetti *JavaScript* e *Node.js*. 30, 67

Python In informatica *Python* rappresenta un linguaggio di programmazione ad alto livello, orientato agli oggetti e interpretato. 32, 67

repository In informatica con il termine *repository* si indica un ambiente di archiviazione centralizzato dedicato alla gestione e al versionamento del codice sorgente. 2, 67

Rollup.js In informatica *Rollup.js* è uno strumento di *bundling JavaScript* che consente di organizzare e raggruppare più moduli in un singolo file. 29, 67

Scrum In ingegneria del software, *Scrum* è un *framework agile* per la gestione del ciclo di sviluppo di un software. È caratterizzato da una struttura organizzativa flessibile e iterativa che promuove la trasparenza, collaborazione e la continua consegna di valore. 2, 67

serverless In informatica con il termine *serverless* si indica un paradigma di elaborazione *cloud*, utile agli sviluppatori per creare e gestire applicazioni senza la necessità di gestire server o infrastrutture sottostanti. 33, 67

single page application In informatica con il termine *single page application* (*SPA*) si indica un'applicazione web che funziona interamente su una singola pagina web, senza ricaricare la pagina durante l'utilizzo. La *SPA* utilizza tecniche di caricamento asincrono per caricare rapidamente il contenuto della pagina senza causare interruzioni all'utente. 28, 67

software In informatica con il termine *software* si indica l'insieme dei programmi, delle applicazioni, dei dati e delle istruzioni digitali che compongono un sistema informatico. 30, 68

Sprint Nel *framework Scrum* lo *Sprint* rappresenta un periodo di tempo limitato durante il quale un team di sviluppo lavora su un insieme di attività concordate. 2, 68

token JWT In informatica con il termine *token JWT* (*JSON Web Token*) si indica un formato di token standardizzato che consente di rappresentare in modo sicuro informazioni tra due parti come un oggetto *JSON*. Uno dei suoi tipici scopi è quello di verificare l'autenticità di un utente durante l'autenticazione. 52, 68

UML In ingegneria del software *Unified Modeling Language UML* (ing. linguaggio di modellazione unificato) è un linguaggio di modellazione standardizzato e utilizzato nell'ambito della progettazione di sistemi *software*. L'*UML* fornisce un insieme di simboli grafici, convenzioni e notazioni che consentono di rappresentare visivamente i concetti e i comportamenti dei sistemi software. 9, 68

URI In informatica con il termine *URI* (*Uniform Resource Identifier*) si indica una sequenza di caratteri che identifica univocamente una specifica risorsa in Internet. 40, 68

YAML In informatica con il termine *YAML* (*YAML Ain't Markup Language*) si indica un formato di rappresentazione dei dati progettato per essere facile da leggere e comprendere dall'uomo. È basato su un formato di testo semplice che utilizza l'indentazione per rappresentare la struttura dei dati, rendendolo intuitivo per gli sviluppatori. 5, 68

Bibliografia

Siti web consultati

[1] *Amazon Web Services*. URL: https://aws.amazon.com/it/ (cit. a p. 2).

[2] *Architettura di Nest.js*. URL: https://www.mantralabsglobal.com/implementing-a-clean-architecture-with-nest-js/ (cit. a p. 35).

[3] *Architettura di Vue.js*. URL: https://012.vuejs.org/guide/ (cit. a p. 35).

[4] *AWS CodeCommit*. URL: https://aws.amazon.com/it/codecommit/ (cit. a p. 30).

[5] *Azure MSAL*. URL: https://learn.microsoft.com/it-it/azure/active-directory/develop/msal-overview (cit. a p. 31).

[6] *Confluence*. URL: https://www.atlassian.com/it/software/confluence (cit. a p. 2).

[7] *Fork*. URL: https://git-fork.com/ (cit. a p. 2).

[8] *Git*. URL: https://git-scm.com/ (cit. a p. 30).

[9] *Jira*. URL: https://www.atlassian.com/it/software/jira (cit. a p. 2).

[10] *Manifesto Agile*. URL: http://agilemanifesto.org/iso/it/manifesto.html (cit. a p. 2).

[11] *Microsoft 365*. URL: https://www.microsoft.com/it-it/microsoft-365?rtc=1 (cit. a p. 2).

[12] *Nest.js*. URL: https://nestjs.com/ (cit. a p. 29).

[13] *Nest.js Middleware*. URL: https://docs.nestjs.com/middleware (cit. a p. 52).

[14] *Node.js*. URL: https://nodejs.org/it/ (cit. a p. 29).

[15] *Pinia*. URL: https://pinia.vuejs.org/ (cit. a p. 30).

[16] *Postman*. URL: https://www.postman.com/ (cit. a p. 2).

[17] *Refresh token in Azure AD*. URL: https://learn.microsoft.com/en-us/azure/active-directory/develop/scenario-spa-acquire-token?tabs=javascript2 (cit. a p. 37).

[18] *Sass*. URL: https://sass-lang.com/documentation/ (cit. a p. 29).

[19] *Scrum*. URL: https://www.scrum.org/resources/what-is-scrum (cit. a p. 2).

[20] *StarUML*. URL: https://staruml.io/ (cit. a p. 3).

[21] *Swagger UI*. URL: https://swagger.io/tools/swagger-ui/ (cit. a p. 31).

[22] *THRON DAM*. URL: https://www.thron.com/en/platform/dam-software/ (cit. a p. 1).

[23] *THRON PIM*. URL: https://www.thron.com/en/platform/pim-software/ (cit. a p. 1).

[24] *THRON S.p.A.* URL: https://www.thron.com/it/ (cit. a p. 1).

[25] *TypeScript*. URL: https://www.typescriptlang.org/ (cit. a p. 29).

[26] *Vite*. URL: https://vitejs.dev/ (cit. a p. 29).

[27] *Vitest*. URL: https://vitest.dev/ (cit. a p. 30).

[28] *Vue-router*. URL: https://router.vuejs.org/ (cit. a p. 30).

[29] *Vue.js*. URL: https://vuejs.org/ (cit. a p. 28).