Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di laurea in Informatica

Test Report

Relatore: Prof. Gianluca Della Vedova

Correlatore: Dott. Antonio Riva

Relazione della prova finale di:
Davide Cologni
Matricola 830177

Anno Accademico 2020-2021

Indice

Elenco delle figure

Capitolo 1

Introduzione

L'oggetto della tesi é una relazione dettagliata sul progetto di stage universitario svolto presso DDX, una software house che si occupa dello sviluppo di software CAD/CAM [1].

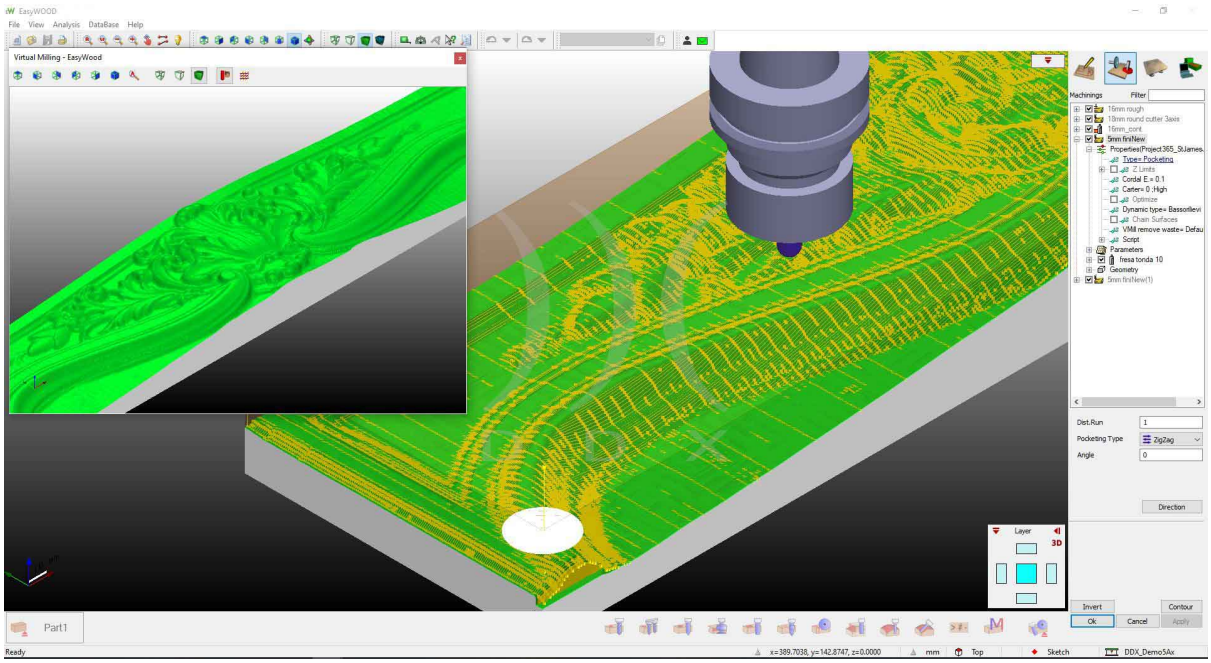

Figura 1.1: Uno dei Software di CAD/CAM sviluppato internamete

Il risultato questo periodo di sviluppo, durato in totale 3 mesi, é un report web dinamico per la consultazione e la modifica dei test. In questa tesi verranno discusse le tecnologie, le scelte tecniche e la struttura dell'applicazione.

Per essere meglio compreso il funzionamento dell'applicativo verrá illustrata nello specifico la struttura del sistema di test.
Successivamente verrá descritta una panoramica del flusso di sviluppo e come il report in oggetto si integri con esso.

[1] **Software CAD/CAM** - Un software CAD/CAM è un programma che consente la progettazione (CAD) e la realizzazione (CAM) di manufatti in diversi materiali, come il legno, il vetro o la pietra, tramite l'uso di macchine a controllo numerico (CNC).

Capitolo 2

Processo di sviluppo

Ogni programma sviluppato internamente si occupa in modo specializzato della manifattura di un materiale specifico (come il legno, la pietra o il vetro).
Lo sviluppo dei software CAD/CAM é stato suddiviso in diversi moduli.
Alcuni per esempio si occupano delle funzionalitá comuni di disegno e di manipolazione delle geometrie, oppure del calcolo della posizione degli utensili durante la lavorazione del materiale.
Questo tipo di moduli che provvedono funzionalitá comuni sono utilizzati da multipli software.
Altri moduli (detti plugin) sono opzionali e usano le funzionalità del software per automatizzare processi industriali come, nel caso del legno, la produzione di travi o di mobili da cucina.

Nel ciclo di vita del software viene eseguita una fase di testing.
Questo procedimento é atto a garantire la correttezza e l'affidabilitá del software.
Nella fase di testing ogni modulo é testato separatamente in una test suite a lui dedicata.
Le diverse suite contengono una serie di input d'esempio e i loro risultati aspettati, in questo modo viene definita una specifica del comportamento del programma.

2.1 Flusso di sviluppo

Per eseguire una modifica al software e renderla pubblica gli sviluppatori seguono un flusso di sviluppo prestabilito.
L'ultima versione pubblica viene scaricata in locale usando il sistema di controllo di versione.
Lo sviluppatore carica i moduli necessari all'interno dell'IDE.
Ora vengono scritte o modificate le porzioni di codice necessarie per la correzzione dell'errore o l'aggiunta della funzionalitá.
Dal sorgente modificato viene compilata una versione di debug del software che verrá usata per lo sviluppo.
Al termine della fase di scrittura del codice, per testare l'effettiva correttezza in ogni caso d'uso viene compilato il modulo modificato in una versione *release* sia a 32 che 64 bit. La versione di release differisce da quella di debug perché non dispone di tutte le operazioni di log e telemetrica garantendo delle maggiori performance del software.

I test sono suddivisi in caso d'uso.
Pertanto, nella la maggior parte dei casi, lo sviluppatore sa quali test controlleranno la correttezza della sua modifica.
Generalmente viene eseguito in locale solo un sottoinsieme di test al fine di ottenere una conferma preliminare della correttezza dello sviluppo.

Essendo la test suite molto ampia e il processo di testing molto dispendioso computazionalmente, la sessione di test non viene lanciata in locale dal singolo sviluppatore, Bensí le modifiche del software già precedentemente compilate (sostituendo l'exe o una libreria dinamica .dll) vengono caricate su un server apposito, il quale eseguirá ogni test della test suite.

Il processo di testing su server é descritto piú dettagliatamente nel capitolo 7.2.

Al termine dell'esecuzione dei test gli sviluppatori interessati vengono notificati con un mail contentente un link al report.
Attraverso l'uso del report il programmatore decide se i test hanno prodotto il risultato sperato.
Nel miglior caso le modifiche verranno revisionate nel sistema di controllo di versione.
Altrimenti eńecessario debuggare i test con esito negativo per capire quali sono i comportamenti introdotti che non seguono le specifiche.

2.2 Necessitá alla base del nuovo sviluppo

Lo scopo del nuovo applicativo é quello di snellire il flusso di sviluppo, in particolare la fase di consultazione dei test.

Prima dello sviluppo era giá presente un tool web a supporto della fase di analisi e accettazione dei test.
Questo strumento aveva peró le seguenti limitazioni:

- Il sistema di versione GIT non è supportato.
- il tool è svilupparto con tecnologie di rendering server-side e ogni operazione richiede il ricaricamento della pagina e di conseguenza l'intera scansione della test suite.
- Il rendering server side non rendeva riusabili le informazioni riguardo hai test.
- L'interfaccia grafica non é interattiva e non è possibile nascondere o filtrare i test.

Per questi motivi é stato scelto di sviluppare un applicativo con un architettura client-server (Descritta dettagliatamente nel capitolo 4).

Capitolo 3

Sistema di Testing

3.1 Meccanismo dell'oracolo

Per la verifica del test viene sfruttato il meccanismo dell'oracolo.
Esso consiste nell'accoppiare ogni test all'output corrispondente (chiamato appunto oracolo) e successivamente nell'effettuare un confronto tra il risultato del test e il suo riferimento.
Il test viene dichiarato fallimentare se ci sono differenze tra gli outcome.

Ogni test deve riferirsi solo a una funzionalitá o ad un insieme di esse strettamente correlate tra loro.
Questo consente allo sviluppatore di individuare eventuali errori con maggiore precisione nel caso in cui un test fallisca.
Se vi fosse la necessitá di cambiare le specifiche del software o di aggiungerne funzionalitá, é possibile rispettivamente modificare un oracolo esistente o crearne uno nuovo.

Nel nostro sistema l'oracolo è salvato in un file *.reference* e il risultato prodotto dal test in un file *.current*.
Il file current viene salvato nel file system solo nel caso in cui esso sia effettivamente diverso dal suo riferimento.

Salvare il nuovo stato può consentire allo sviluppatore di confrontare i due file per capire quale sia l'errore introdotto nel software.
Nella sezione successiva vengono illustrati gli elementi dal quale é composto un test e in seguito la loro funzione.

3.2 Anatomia di un test

Un test consiste in un insieme di file locati per comoditá nella stessa cartella.

Ogni test consiste in:

- Un file CAD di partenza
- Dei file per descrivere una configurazione specifica di un determinato cliente
- Un file SCL [1] o un file python
- Uno o piú riferimenti (file .reference)

3.3 Ciclo di vita del test

Il software, per lanciare un test, importa il file CAD e le configurazioni del cliente.
Successivamente esegue lo script SCL o Python, che con opportune librerie, si interfaccia con il software e simula le operazioni utente.

Tra le diverse routine alcune si occupano di creare le reference esportando un risultato parziale dal programma come uno screenshot, un gruppo di entitá geometriche o un file testuale.

Durante la esecuzione di un test i suoi outcome vengono salvati in una cartella di cache.
Nel caso sia giá presente una reference esso viene salvato come current.

Se il file prodotto dalle operazioni é uguale all'oracolo, esso viene rimosso dal file system.
In caso contrario il test é in stato di errore e l'output resta disponibile per il confronto.

3.4 Versioning dei tests

I test si trovano sotto un sistema di versioning.
Quando uno sviluppatore decide di cambiare il comportamento di un caso d'uso o di aggiungere una funzionalità al software, sará necessario modificare o creare un test.
Esso verrà committato nel sistema di versioning in modo che gli altri sviluppatori possano testare il software secondo le nuove specifiche definite dal test.

[1] **Sculptor Common Language** é un linguaggio di scripting proprietario interpretato con un insieme di routine per il disegno e la manifattura dei materiali

Capitolo 4

Struttura dell'applicazione

L'applicazione é stata sviluppata seguendo un'architettura client-server.

Questa infrastruttura è composta da diversi programmi che interagiscono tra loro usando messaggi di rete, nel nostro caso codificati secondo il protocollo HTTP. Comunemente, le componenti principali sono il backend e il frontend.

4.1 Elementi dell'applicazione

La prima componente di *backend* si occupa della parte di logica di dominio dell'applicazione.
La seconda, denominata *frontend*, rappresenta graficamente i dati e di rende le interazioni tra essi e l'utente fruibili. Decuplicare le parti dell'applicazione consente di rendere i singoli componenti riusabili.

L'esempio piú comune consiste nel condividere la logica dell'applicazione usando un backend e multiple rappresentazioni grafiche a seconda della piattaforma di utilizzo. In questo modo si può creare a un'applicazione web o un app mobile, senza duplicare la logica di dominio che verrà gestita dell'altra componente.

Nel nostro caso in esame, la parte di backend si occuperà di leggere i test da file system e di esporli tramite un API.
É stato sviluppato un frontend web per interagire graficamente con i test.

4.2 Intercomunicazione tra gli elementi

La comunicazione avviene tramite messaggi di rete codificati secondo il protocollo HTTP. La condivisione del protocollo consente agli attori di comunicare senza conoscere i dettagli implementativi dell'altro.
Ció implica un ulteriore vantaggio: gli elementi possono essere eseguiti su piattaforme diverse o essere scritti in linguaggi differenti, a seconda delle diverse necessità del singolo componente.

Per il caso in oggetto la comunicazione viene iniziata dall'applicazione web e l'API REST si occupa di rispondere alle richieste.
Il formato di interscambio dei dati usato é *JSON*[9].

4.3 Vantaggi, svantaggi dell'architettura e alternative

I vantaggi di questa architettura, come precedentemente detto, sono la decuplicazione dei diversi elementi, il loro riutilizzo e la separazione delle responsabilità.

In un caso futuro, sarà possibile sviluppare un altro frontend per visualizzare i test su un altra piattaforma riutilizzando il backend e garantendo la stessa logica applicativa.
Viceversa la parte di frontend non conosce i dettagli implementativi usati dal backend come la gestione delle reference tramite file system e il sistema di versioning.
Questo consente di cambiare le gestioni di basso livello lasciando invariato l'uso del programma agli occhi dell'utente.

Dall'altra parte, di contro, la separazione degli elementi porta a una piccola duplicazione di quelle che sono le parti comuni tra il client e il server, come i modelli di dominio.

Per il tool aziendale precendente era stata adottata un'architettura diversa: un unico attore si occupava di consultare le reference e creare l'interfaccia grafica.
Questo sistema non é peró estensibile e riusabile come quello attuale, essendo la logica di dominio e la generazione della pagina eseguita dallo stesso componente.

Capitolo 5

Backend

Il backend è stato sviluppato in Python [12] usando il framework Flask [3].
É stato scelto Python perchè è un linguaggio già usato internamente per lo sviluppo di plugin e tool interni.

5.1 Specifica e documentazione

Prima della fase di sviluppo è stata predisposto un periodo di design dell' API.
Questo processo ha prodotto un file di specifica (descritta nel paragrafo 5.1.1).

Questa fase preliminare allo sviluppo è durata circa una settimana ed è servita per individuare difficoltà tecniche in modo preliminare che sono sate discusse con il team.
In questo modo é stato risparmiato tempo durante lo sviluppo, evitando di dover ristrutturare l'applicazione successivamente.
Questa specifica come vedremo in seguito è servita anche a produrre automaticamente una pagina di documentazione (5.1.2).

5.1.1 Specifica in linguaggio OpenAPI

La Specifica dell'API è stata scritta in formato OpenAPI [11].
Questo file può essere scritto in linguaggio JSON o YML, io ho scelto il secondo perchè sintatticamente più leggero e usa l'indentazione per separare le sezioni, facilitandone dunque la scrittura.
Questo formato ci consente di descrivere tutte le chiamate che il server mette a disposizione, specificando:

- Il formato dell'URI
- Il metodo
- Il formato di interscampio dei dati (json, xml ecc)
- La struttura della risposta
- Come deve essere strutturata la richiesta perché essa sia valida
- I codici di errori e la loro semantica specifica nel contesto dell'applicazione

5.1.2 Documentazione

Inoltre da questa specifica è possibile generare la documentazione, in pagina statica html, tramite l'utilizzo di diverse CLI[1].
Io ho scelto ReDoc, perchè mette a disposizione diverse vendore-extension, cioè sezioni che se specificate provedono una maggiore customizzazione della pagina generata.
Queste estensioni sono state usate per sezioni di codice di esempio riguardo a come effettuare una chiamata al server, e per aggiungere riferimenti a file markdown esterni da includere nella documentazione.

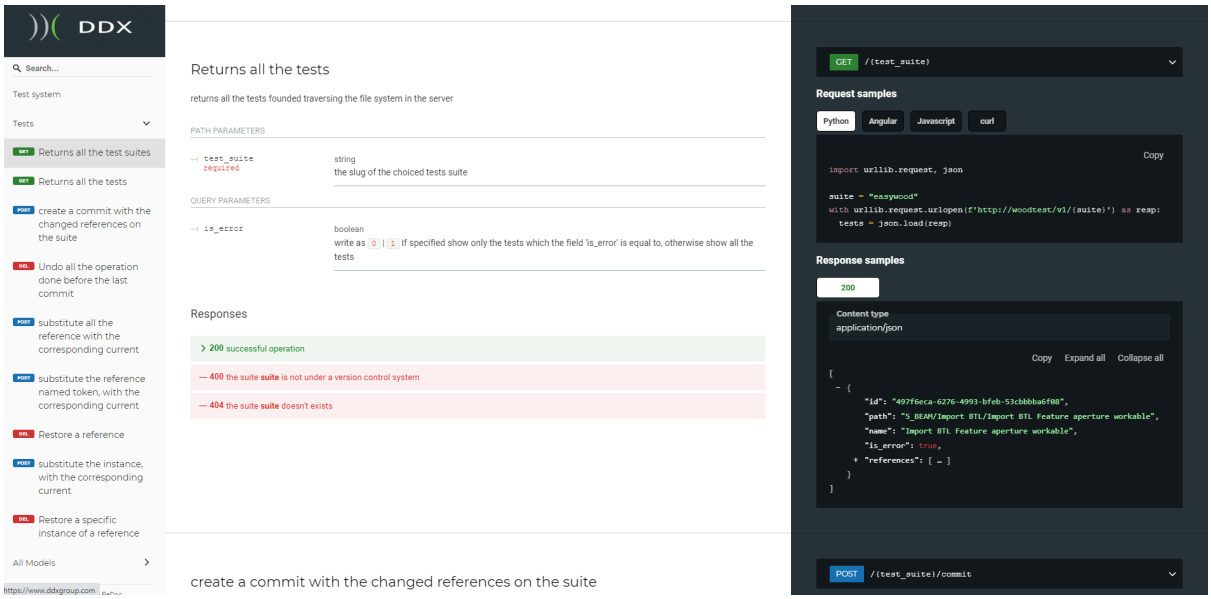

Figura 5.1: Uno screenshot della pagina di documentazione generata dal processo di CI

5.1.3 Continous integration

Il processo di scrittura della documentazione è stato aggiunto a una pipeline di CI [2].
Questo processo si occupa, solo se il file .yml di specifica viene modificato, di rigenerare il file html di documentazione.
Questo file viene successivamente servito in modo da rendere la documentazione disponibile online e sempre aggiornata con le specifiche.

[1] **Command Line Interface** - Interfaccia a linea di comando

[2] **Continous Integration** - L'integrazione continua consiste nell'integrare gli ambienti di sviluppo dei diversi sviluppatori tramite processi di build, testing ecc... eseguiti da un server condiviso

5.2 Tecnologie Utilizzate

5.2.1 Python

Python é un linguaggio di scripting, interpretato ad alto livello, ovvero la gestione della memoria viene effettuata da un sistema di garbage collection e non delegata allo sviluppatore.
Questo linguaggio è stato scelto siccome é usato per lo scripting all'interno di altri applicativi esistenti e dunque giá familiare a molti sviluppatori.

Tra le varie librerie prodotte é presente anche un modulo per lanciare i test e inviare i risultati agli sviluppatori via e-mail.
Questo modulo non è stato utilizzato all'interno del test report ma l'uso dello stesso linguaggio ha contribuito a mantenere la code-base piú uniforme.

5.2.2 Flask

Flask é un *micro*Framework per la creazione di servizi web, esso consente di creare applicazioni complete o API (come nel nostro caso).
La sua peculiaritá é la minimalitá e consente si scrivere applicazioni in modo rapido riducendo al minimo la parte di codice che si occupa della gestione dell'interfaccia HTTP [7].
Questo é possibile grazie agli strumenti di introsprezione del codice offerti dal linguaggio come i decoratori.

Esso é stato scelto perché l'applicazione, essendo destinata all'uso interno, non necessitava di gestioni di rete di basso livello per coordinare grandi quantitá di richieste.
L'attenzione principale é stata invece posta sulla riusabilità dell'API.

5.3 Interazione con i sistemi di controllo di versione

Un sistema di controllo di versione è un software distribuito che si occupa di tenere traccia di tutti i cambiamenti effettuati su un progetto.
Lo sviluppatore dopo ogni modifica si occuperà di registrarla nel sistema, creando quello che viene definito un commit.
Questi software consentono di ripristinare versioni precedenti e avere uno storico dei cambiamenti effettuati.
Le tests suite sono revisionate con un sistema di versioning.

L'API oltre alla modifica, aggiunta o rimozione dei test si occupa anche di versionare queste azioni nel sistema usato dalla suite.
Per sapere quali sono le modifiche alla suite effettuate, il servizio si avvale di *staging area*.
Sono aree intermedie salvate in locale che consentono di formattare e rivedere le modifiche prima di creare il commit.
La creazione del commit avviene tramite un endpoint specifico.

Internamente vengono usati sia Git [4] che SubVersion (SVN) [1] e una specifica del report è supportare entrambi in modo trasparente allo sviluppatore.
Questa funzionalità nasce da una necessità futura di effettuare una migrazione delle test suite revisionate attualmente con Svn a Git.

5.4 Struttura dell'API

Nel processo di design dell'API si è scelto di adottare i principi architetturali REST [13]. Il primo punto di questa filosofia consiste nel poter assegnare un indirizzo (detto *URI* ovvero *Unique Resource Identifier*) a ogni risorsa dell'applicazione.
L'URI deve essere univoco e descrittivo riguardo al tipo di risorsa che si sta richiedendo.

La comunicazione si basa sul protocollo HTTP e coinvolge due tipi di attori chiamati client e server.
Il server si occupa di esporre l'API e i client di effettuare una serie di richieste per manipolare le risorse a seconda delle loro necessità.
Ogni messaggio scambiato tra i client e il server si occupa di effettuare una singola operazione su un oggetto.
La richiesta è composta da un metodo che descrive quale azione compiere, e un URI che indica il soggetto.

Ogni URI é composto da diverse sezioni, tra esse il *path* che definisce una collocazione logica della risorsa all'interno dell'applicazione.
Gli indirizzi delle risorse sono creati ragruppando quelli che hanno informazioni associate. Per esempio se si vuole accedere a un test viene identificata prima la suite di appartenenza e poi l'id del test: `/easywood/doh2hw527jm`.
Per riferirsi alle reference invece ci si riferisce al test di appartenenza e successivamente viene specificata la distribuzione, in questo modo: `/easywood/doh2hw527jm/64bit`.

I metodi principali sono:

- **GET** Per ottenere le informazioni relative a una risorsa
- **POST** Per la creazione di nuove risorse
- **PUT** Si riferisce alla modifica di elementi già presenti nel sistema
- **DELETE** Per eliminare dati

Ogni API definisce una semantica specifica per le sue operazioni.
Nel nostro caso non ogni metodo è stato definito dal servizio, perché altre operazioni come la creazione di un test o una suite esulano dalle responsabilità dell'applicazione.

Per i test è disponibile il metodo GET per ottenerne il nome, il path relativo e le sue reference.
Ognuna di esse è composta da stato di errore e di stato di registrazione nel sistema (registrato, modificato o non registrato).
Per le reference sono disponibili anche i metodi PUT e DELETE. PUT è per definizione da specifica di REST idempotente e si occupa della modifica di una risorsa, percio nel nostro caso registra nel sistema le eventuali modifiche/differenze ottenute durante la sessione di test, se la reference si trova in stato di errore essa verrà sostituita dal suo file current, altrimenti se non è in errore o è già stata modificata non verrà effettuata alcuna operazione.

I soggetti dell'API sono le operazioni di registrazione dei test, perciò l'operazione di DELETE assume la semantica di "undo", ovvero ripristino dello stato del sistema all'ultima configurazione registrata nel controllo di versione.

Il server risponderà con un pacchetto HTTP composto da uno status code e un corpo contente il risultato dell'operazione.
Lo status code è un codice di 3 cifre che descrive se l'operazione è andata a buon fine o il tipo di errore (per esempio: risorsa non trovata, utente non autorizzato ecc...).

Capitolo 6

Frontend

Questa parte dell'applicazione consiste in un'interfaccia grafica che consente all'utente di visualizzare, sostituire i test e di committarne le modifiche.
Puó anche essere integrata con un tool esterno per consultare le differenze tra il risultato atteso e quello ottenuto nei test in errore.
L'interfaccia é una pagina web dinamica perciò disponibile online a tutti gli sviluppatori.

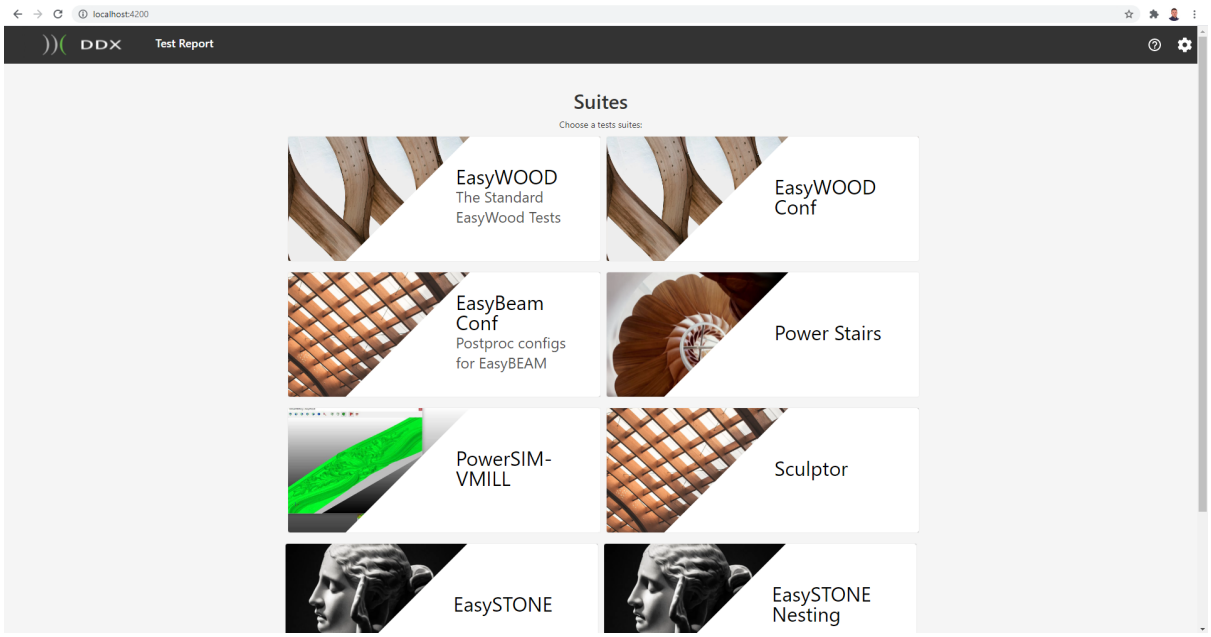

Figura 6.1: La pagina di home dell'interfaccia web attraverso la quale é possibile scegliere quale test suite consultare.

6.1 Angular

Angular [5] é un framework web che consente di creare applicazioni dinamiche e reattive. Questo framework é specifico per lo sviluppo di *SPA* (Single Page application).
Una Single Page Application differisce da una applicazione web normale disponendo di una sola pagina e renderizzando dinamicamente i singoli componenti HTML in base allo stato dell'applicazione.
Il rendering dinamico consente una maggiore fluidità nel cambiamento di pagina, non avendo l'overload di richiedere la risorsa sulla rete.
Nonostante le SPA consistano in un solo file HTML hanno comunque la possibililtà di visualizzare documenti specifici in base al loro indirizzo corrente, grazie a un sistema chiamato *Routing*.

6.1.1 Scelta dell'utilizzo

É stato scelto di usare Angular, perchè è un framework completo supportato da Google ed é già stato usato precedentemente per lo sviluppo di altri applicativi.

Angular ha una struttura a componenti ovvero basici elementi grafici con una singola responsabilità che composti tra loro formano elementi più avanzati.
Questo stile strutturale consente il riuso del codice, per esempio, un componente sviluppato precedentemente consente di renderizzare un file CAD proprietario all'interno della pagina web.

6.1.2 Typescript

Un altro motivo per la scelta di Angular è Typescript [10].
Typescript è un *superset di javascript* cioè un'estensione che ne ingloba tutte le funzionalità e ne aggiunge altre.
La sua peculiarità principale è l'aggiunta di uno step di compilazione che favorisce il controllo della correttezza del codice attraverso una tipizzazione statica.

6.2 Funzionalità

La prima responsabilità dell'interfaccia web é rendere i test fruibili graficamente.
Ogni test é rappresentato da una scheda (in gergo *card*) che contiene una lista di reference con un indicatore grafico del loro stato.
É possibile visualizzare tutti i test o solo quelli con almeno una reference in errore.

6.2.1 visualizzazione dei test

Uno scopo del report é rendere piú facile individuare i test in errore.
É perció possibile filtrare i test in base al loro stato rendendo visualizzabili solo quelli con almeno una reference differente.
Per rendere piú intuibile l'utilizzo del programma viene visualizzata una pagina speciale nel caso non ci siano test con errori in modo che l'utente sappia che é andato tutto a buon fine.

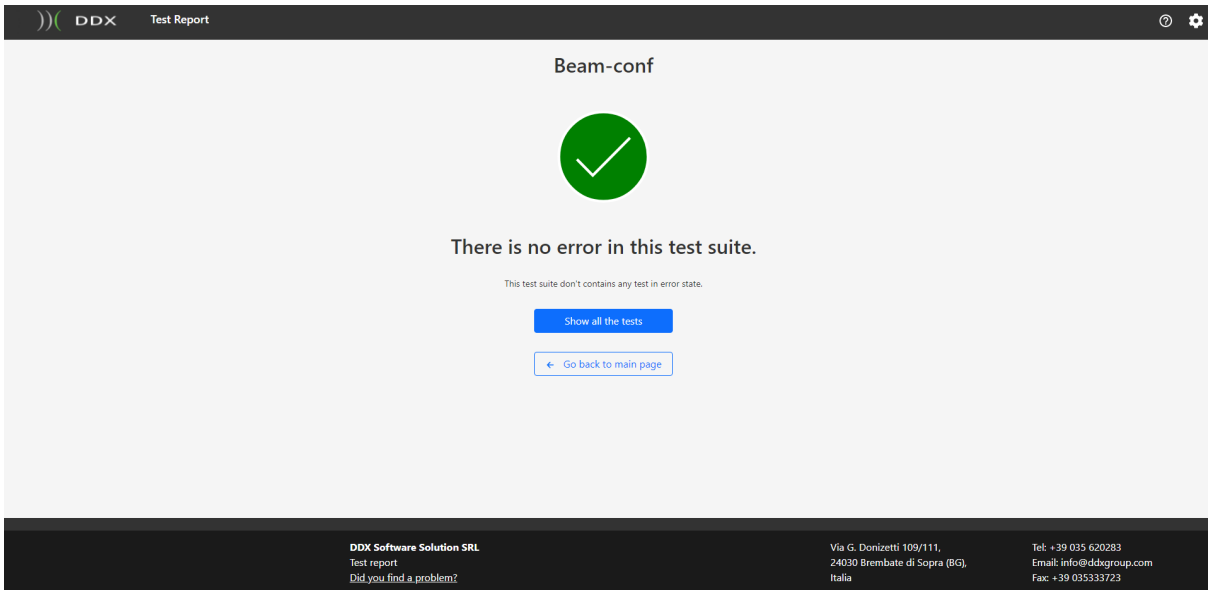

Figura 6.2: Pagina per suite senza errori

Lo sviluppatore puó anche nascondere le reference corrette all'interno di una card.

Un ulteriore modo per togliere un test dalla vista é l'operazione di *collapse*: essa consente di nascondere tutte le reference lasciando solamente il nome del test.
Con una specifica pagina di configurazione si puó personalizzare il comportamento del programma in modo che tutte le card vengano collassate all'avvio dell'applicazione.

6.2.2 Gestione interattiva dei file sovvrascritti

L'iter di modifica di un test consiste nel:

- Confronto di un oracolo con il suo risultato in errore
- Aggiunta delle azioni in un area predisposta

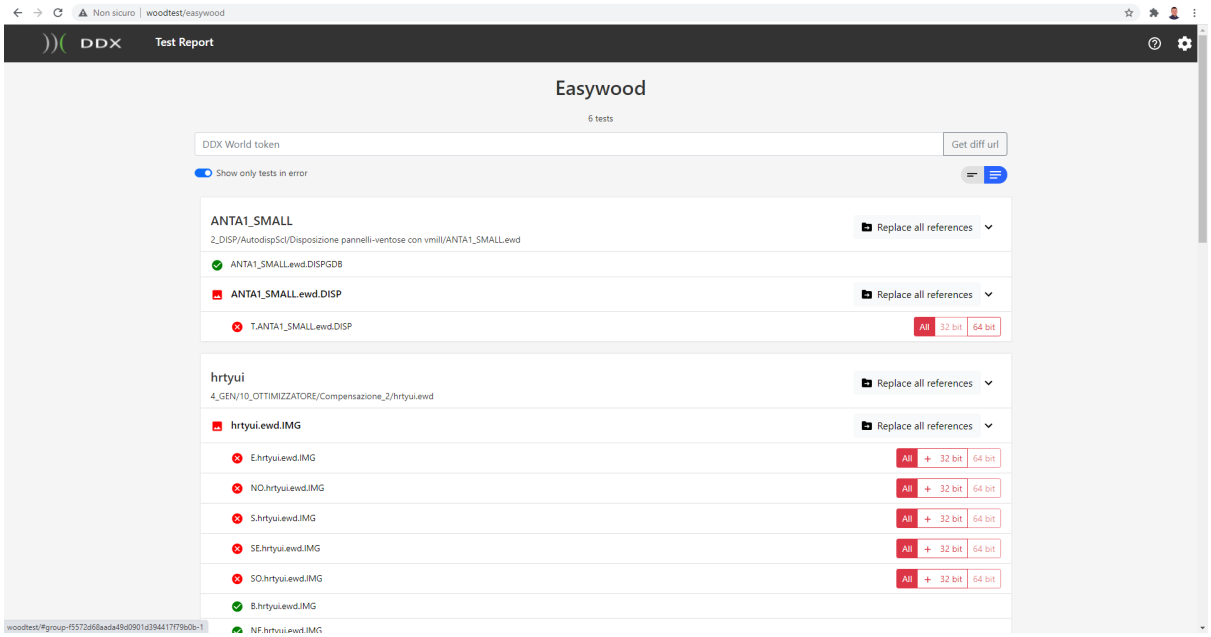

Figura 6.3: Un esempio di visualizzazione di test

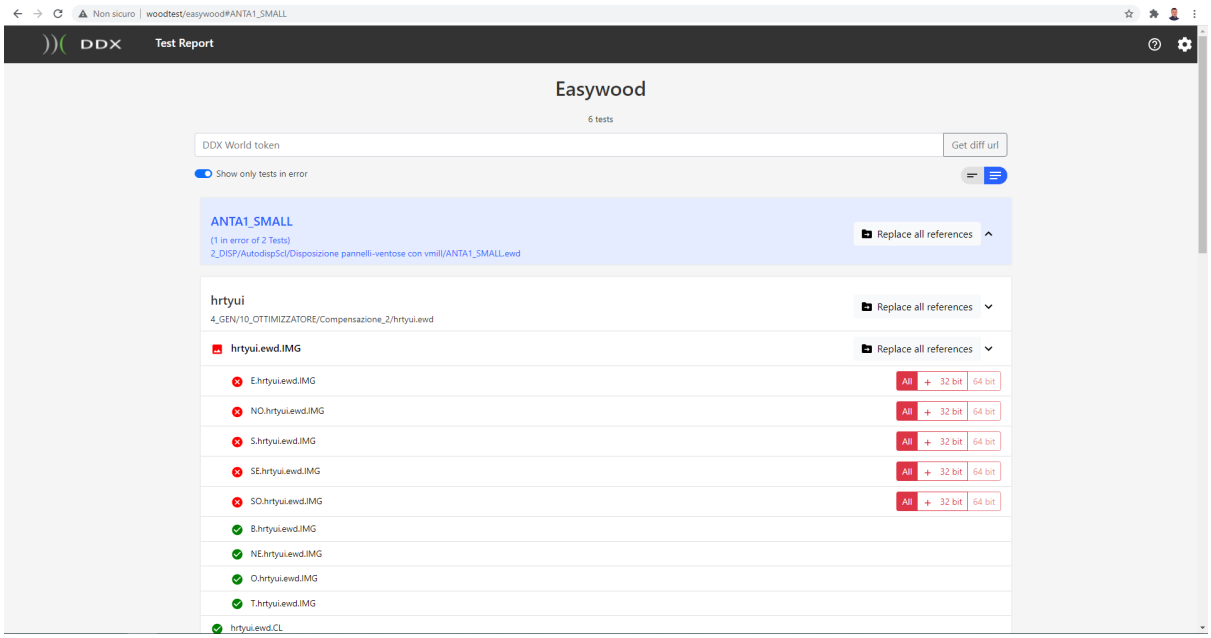

Figura 6.4: Un test in stato *collapsed*

- Creazione di un commit

Il sistema tiene traccia delle azioni effettuate perchè esse non sono effettive e condivise in rete fino a quando non viene creato un commit. L'interfaccia web dispone una sezione per salvare le azioni e committarle. In un area specifica vengono visualizzate tutte le azioni, che possono essere annullate o definitivamente registrate.

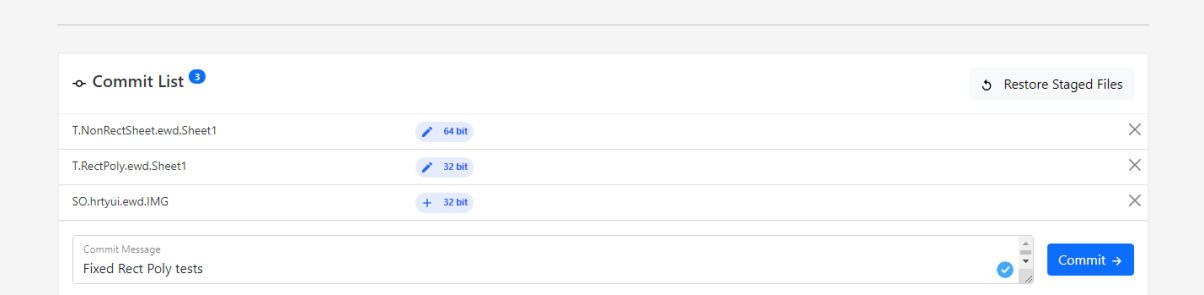

Figura 6.5: L'area di stage premilinare

6.2.3 Navigazione

É disponibile un metodo di navigazione per aumentare l'usabilità dell'applicazione.
Durante la navigazione è possibile rendere un test *focussed* ed effettuare operazioni su esso usando delle scorciatoie da tastiera.
Con il tasto **C** E' possibile rendere il test *collapsed* o usando **S** mettere tutte le sue reference in area di stage.
Nello stesso modo si puo spostare il focus della navigazione sul prossimo elemento o su quello precedente, rispettivamente con **N**(next) o **B**(back).
Quando il focus si sposta su un nuovo test viene effettuata un operazione di scroll per includerlo nella parte visibile della pagina.

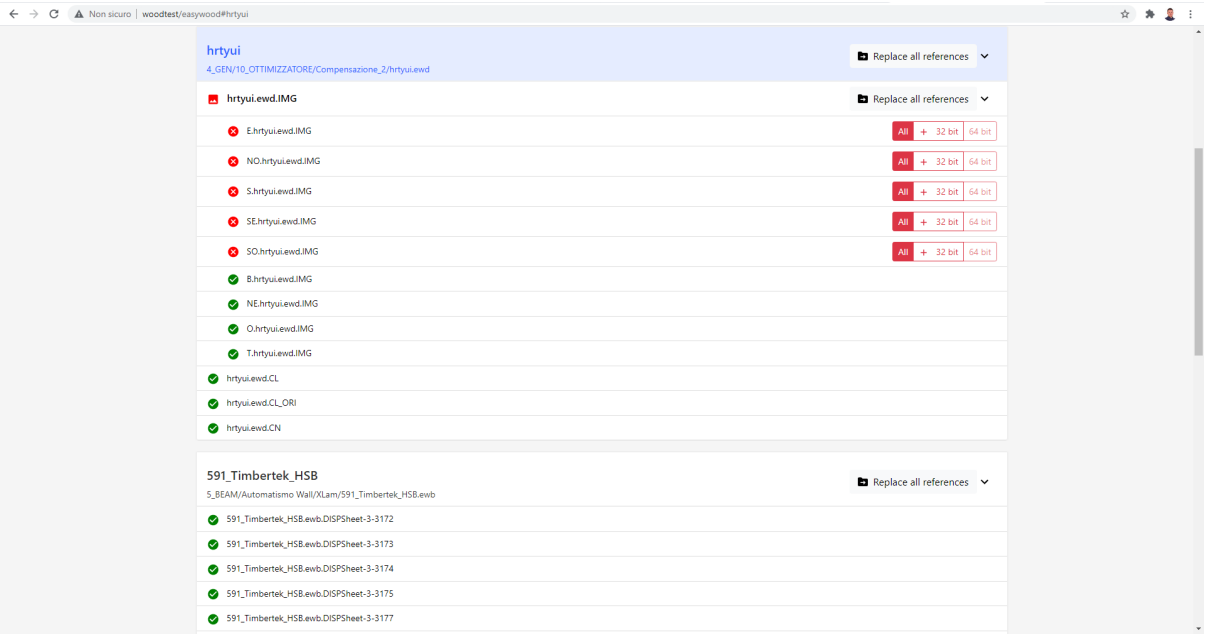

Figura 6.6: Un test in stato di focus. La sua scheda relativa viene evidenziata e l'url della pagina viene modificato includendo il nome della risorsa.

Lo stato della navigazione viene descritto nell'URL, scrivendo il nome dell'elemento in focus nella sezione denominata *fragment*.
Se la pagina viene richiesta con un fragment già impostato, in fase di rendering dei tests viene posto il focus automaticamente sul test con quel nome (in questo modo sará

visualizzabile giá al caricamento della pagina).
La gestione dei fragment rende condivisibile una risorsa primaria dell'applicazione quale il test indirizzandolo tramite un URL, come voluto dalla filosofia web.

6.2.4 Impostazioni grafiche

É possibile personalizzare il comportamento dell'applicazione usando una pagina di impostazioni.
Questi settaggi sono specifici all'utente e salvati in locale in modo persistente.
Le configurazioni disponibili consistono in:

- Visualizzare solo i test in errore quando si visualizza una suite.
- Scegliere se le schede sono in stato *collapsed* all'avvio.
- Nei test in errori visualizzare anche le reference corrette o solo quelle in stato di errore.

Le impostazioni consentono di modificare il layout grafico dell'applicazione e favorire la ricerca dei test errati a seconda delle esigenze specifiche dello sviluppatore e della struttura della suite.

6.3 Interazione con gli altri tool aziendali

La consultazione delle differenze viene delegata a un tool aziendale già sviluppato in precedenza, denominato *Differ*.
Anch'esso é un tool web quindi questa integrazione avviene costruendo un URI per navigare nella pagina successiva.
Il processo di esecuzione dei test di occupa di salvare i risultati di ogni sessione di test e rendere i risultati disponibili tramite Differ mantenendo dunque una cronologia.
Il report in oggetto, invece produce un interfaccia in tempo reale basata lo stato del server.
Per sincronizzare lo stato tra i due sistemi é necessario recuperare l'ultima sessione di test eseguita e, per ogni reference, creare un url che associ il test con le sue differenze.

Capitolo 7

Distribuzione e uso nel processo di sviluppo

7.1 Bundling e deploy

Ogni team che si occupa dello sviluppo di un'applicazione di CAD/CAM ha adibito un server interno per la compilazione e il testing del proprio programma.
L'applicativo di report dei test è stato installato su ogni server.

Essendo l'applicazione web-based è stata adibita una porta dove servire la pagina. Sulla stessa porta è stata esposta anche l'API.

In fase di sviluppo del report vengono usati due processi, uno per il frontend e uno per il backend, che comunicano tra loro tramite messaggi HTTP.
Questa gestione è diversa in fase di produzione siccome il processo che serve la parte frontend si occupa di ricaricare la pagina quando occorre una modifica sui file del progetto angular, in modo da mostrare le modifiche in modo interattivo allo sviluppatore.
Per l'interfaccia web viene dunque effettuata una fase detta di "bundling", dove i diversi file vengono compilati in file statici (JavaScript [8], HTML [6] e CSS [2]).
Vengono esposte entrambe le parti dell'applicazione sulla stessa porta grazie al sistema di routing.
Tutti gli indirizzi relativi all'API sono stati spostati nella sottoroute `/v1`.
Per le altre richieste viene controllato se esiste un file statico con lo stesso percorso altrimenti viene servito il file HTML principale in modo da delegare la segnalazione di errori o visualizzazione di pagine specifiche al sistema di routing usata dalla Single Page Application.

7.2 Processo di esecuzione dei test

I test vengono lanciati usando uno script.
Esso si occupa di lanciare il programma ed eseguire tutti i test in ogni suite.
Successivamente carica i risultati su un server di storage apposito e notifica lo sviluppatore con una mail al termine della sessione.

La mail di notifica contiene informazioni preliminari quali il numero di test in errore e un link alla pagine del test report corrispondente.

L'integrazione tra il differ e il report viene gestita da questo programma.
Durante la fase di upload viene creato un token per la sessione appena terminata.
Per ogni test in errore caricato sul server è creata una pagina HTML con le differenze tra le reference e i current.

L'indirizzo di ogni pagina è composto dal token di sessione e il nome del test.
Nel link relativo al test report viene passato come query param il token della sessione e il report creerá l'indirizzo relativo alla pagina delle differenze per ogni elemento in stato di errore.

Capitolo 8

Sviluppi Futuri

8.1 Migrazione test suite da svn a git

Le test suite sono attualmente revisionate dal sistema SVN che dispone di un controllo di versione molto piu elementare rispetto a GIT.

Subversion é stato precedentemente utilizzato per la revisione di tutti i progetti all'interno dell'azienda, test inclusi.
Attualmente si sta lentamente migrando verso l'uso estensivo di GIT, che offre molte piú funzionalitá.

Il test report favorisce questa migrazione utilizzando in modo trasparente all'utente entrambi i sistemi di versione.

8.2 Performance dei test

Il software ha la possibililtà di tenere traccia del tempo di esecuzione di un singolo test. Attualmente le performance di ogni test viene scritta in un database flat-file [1].

Se un tempo supera una soglia che viene considerata il tempo di esecuzione standard di un test questo viene segnalato nella mail di notifica.
L'intenzione futura sarebbe includere nel report i tempi di esecuzione dei singoli test e segnalare quelli con un tempo di esecuzione troppo alto.

[1] Database con una struttura uniforme che vengono salvati in un unico file binario

8.3 Differ integrato

Una proposta per migliorare il report di test é quella di reimplementare il sistema di differenze senza riusare quello già esistente.
Questo rimuoverebbe il problema di allineare i due sistemi passando il token di sessione a entrambi.
Inolte in questo modo il report di test non puo garantire di essere sincronizzato con il differ esterno: se vengono effettuate modifiche (anche da controllo di versione) sui file reference il sistema non puo visualizzarne le differenze in tempo reale.

Un differ integrato si assumerebbe l'onere di calcolare le differenze tra i due file binari ed interpretarli correttamente in modo da esporre nell'API un confronto tra i file leggibile.
Questo consentirebbe di poter confrontare i file in qualsiasi momento senza caricarne prima le differenze su un server apposito.

8.4 Raggruppamento per cartelle

Alcune test suite si occupano di testare multipli moduli all'interno dell'applicativo.
I test adibiti a un singolo modulo vengono raggruppati nella stessa cartella.
Un membro di un team di sviluppo si occupa del mantenimento di un insieme ristretto di moduli.
Dunque una necessitá molto diffusa é visualizzare solo i test di un modulo oppure escludere dalla visualizzazione i test di moduli non interessati. Per soddisfare questa specifica si potrebbe disporre un layout di raggruppamento dei test per cartella.
Questo consentirebbe di poter visualizzare e gestire i test in base al loro modulo di appartenenza.

La complicanza tecnica di questa suddivisione consiste nel separare le modifiche attualmente registrate nel sistema di versioning per il precommit.
Il commit delle modifiche di un modulo non deve registrare anche operazioni effettuate su test in altre cartelle ma tutta la suite é revisionata nella stessa repository.
Questo implica la creazione di multiple aree di stage che devono essere implementate ad-hoc, essendo questa una funzionalità non presente in nessun sistema di controllo di versione.

Capitolo 9

Conclusioni

9.1 Risultato finale e obiettivi iniziali

In questa tesi é stato descritto il software "Test Report" per l'analisi e accettazione dei test.
Nel corso di questo trattato é stata posta l'attenzione sui concetti quali l'architettura REST, l'importanza del processo di testing e l'uso degli strumenti di sviluppo come l'integrazione continua e i sistemi di revisione del software.

Il progetto é riuscito a soddisfare tutti i punti necessari definiti dalle specifiche iniziali. Inoltre nel periodo di stage é stato possibile anche distribuire il progetto in produzione su tutti i server di test.

9.2 Ringraziamenti

Un ringraziamento speciale al mio relatore Gianluca Della Vedova, per la sua pazienza e disponibilitá.

Ringrazio il mio correlatore Antonio Riva per l'aiuto datomi e l'entusiasmo trasmesso durante il periodo di stage.

Un ulteriore ringraziamento va alla mia famiglia che mi ha sempre supportato, in particolar modo a mia nonna Angiolina e a mia nonna Anna che sono state un esempio di cura e di amore.
A loro voglio dedicare questa tesi.

Bibliografia

[1] Apache. *SubVersion*. URL: https://subversion.apache.org/.

[2] *CSS*. URL: https://en.wikipedia.org/wiki/Cascading_Style_Sheets.

[3] *Flask*. URL: https://flask.palletsprojects.com/en/2.0.x/.

[4] *Git*. URL: https://git-scm.com/about.

[5] Google. *Angular*. URL: https://angular.io/.

[6] *HTML*. URL: https://en.wikipedia.org/wiki/HTML.

[7] *HTTP*. URL: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

[8] *JavaScript*. URL: https://en.wikipedia.org/wiki/JavaScript.

[9] *JSON*. URL: https://en.wikipedia.org/wiki/JSON.

[10] Microsoft. *Typescript*. URL: https://www.typescriptlang.org//.

[11] *OpenAPI*. URL: https://spec.openapis.org/oas/v3.1.0.

[12] *Python*. URL: https://www.python.org.

[13] Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.