

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

CORSO DI LAUREA IN INFORMATICA



Advertising management system:  
Creazione e gestione di contenuti pubblicitari

*Tesi di laurea*

*Relatore*

Prof. Claudio Enrico Palazzi

*Laureando*

Alessandro Discalzi

---

ANNO ACCADEMICO 2019-2020



# Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage dal laureando Alessandro Discalzi presso l'azienda SCAI ITEC. Lo stage è stato svolto al termine del percorso di studi della laurea triennale in informatica e la sua durata è stata di 312 ore. L'obiettivo dello stage è stato di implementare un applicativo per la creazione e per la gestione di contenuti pubblicitari. Il presente documento vuole illustrare il contesto aziendale dove si è svolto lo stage, le attività svolte e una valutazione sul lavoro effettuato e su quanto appreso.



“Nessuno ha mai ottenuto nulla con le lacrime.”

— Brucaliffo

# Ringraziamenti

*Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Claudio Enrico Palazzi, relatore della mia tesi, per l'aiuto e il sostegno che mi ha fornito durante la stesura del lavoro.*

*Desidero ringraziare con affetto la mia famiglia, e in particolare i miei genitori per il sostegno e per il grande aiuto che mi hanno dato durante gli anni di studio.*

*Desidero inoltre ringraziare il mio tutor aziendale, Dott. Bledar Gogaj, e il suo collega, Dott. Marco Lionello per l'enorme aiuto che mi hanno dato durante il periodo di stage.*

*Un ringraziamento infine, ai miei amici per tutti i bei momenti passati insieme e per avermi sopportato tutti questi anni.*

*Padova, Settembre 2020*

Alessandro Discalzi



# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	L'azienda . . . . .	1
1.2	Scopo dello stage . . . . .	2
1.3	Tecnologie utilizzate . . . . .	2
1.3.1	JHipster . . . . .	2
1.3.2	Java Enterprise . . . . .	3
1.3.3	Spring . . . . .	3
1.3.4	Hibernate . . . . .	4
1.3.5	REST <sup>1</sup> . . . . .	4
1.3.6	Oracle . . . . .	5
1.3.7	Liquibase . . . . .	6
1.3.8	Angular . . . . .	6
1.4	Strumenti di sviluppo . . . . .	7
1.4.1	Eclipse . . . . .	7
1.4.2	Maven . . . . .	7
1.4.3	Git . . . . .	7
1.4.4	SQL Developer . . . . .	8
1.5	Organizzazione del testo . . . . .	8
<b>2</b>	<b>Obbiettivi e pianificazione</b>	<b>9</b>
2.1	Obbiettivi . . . . .	9
2.1.1	Obbiettivi obbligatori . . . . .	9
2.1.2	Obbiettivi desiderabili . . . . .	9
2.1.3	Obbiettivi opzionali . . . . .	9
2.2	Pianificazione . . . . .	10
2.3	Aspettative personali . . . . .	11
<b>3</b>	<b>Metodologia di sviluppo</b>	<b>13</b>
3.1	Scrum . . . . .	13
3.1.1	Ruoli . . . . .	13
3.1.2	Artefatti . . . . .	14
3.1.3	Fasi . . . . .	17
<b>4</b>	<b>Analisi e progettazione</b>	<b>19</b>
4.1	Analisi e progettazione iniziale . . . . .	19
4.2	Progettazione del database . . . . .	19
4.2.1	Entità . . . . .	20

---

<sup>1</sup>REST: acronimo di Representational State Transfer.

4.2.2	Enumerazioni . . . . .	22
4.2.3	Relazioni . . . . .	22
4.2.4	Implementazione . . . . .	23
4.3	Definizione dei template per i contenuti . . . . .	25
4.3.1	Template di base . . . . .	25
4.3.2	Template di pagina . . . . .	26
4.4	Definizione dei servizi di backend . . . . .	26
<b>5</b>	<b>Prodotto</b>	<b>29</b>
5.1	Prodotto software . . . . .	29
5.1.1	Login . . . . .	29
5.1.2	Scelta funzionalità . . . . .	30
5.1.3	Gestione contenuti . . . . .	30
5.1.4	Sezione editore . . . . .	31
5.1.5	Creazione di un contenuto . . . . .	31
5.1.6	Anteprima di un contenuto . . . . .	32
5.1.7	Clonazione di un contenuto . . . . .	33
5.1.8	Modifica di un contenuto . . . . .	33
5.1.9	Associazione di un contenuto . . . . .	34
5.1.10	Eliminazione di un contenuto . . . . .	34
5.1.11	Audit . . . . .	34
5.2	Documentazione . . . . .	35
5.2.1	Manuale Utente . . . . .	35
5.2.2	Manuale Sviluppatore . . . . .	35
5.3	Test . . . . .	36
5.3.1	Test di unità . . . . .	36
5.3.2	Test di integrazione . . . . .	38
5.3.3	Test di sistema . . . . .	38
5.3.4	Test di performance . . . . .	38
5.4	Accessibilità . . . . .	38
<b>6</b>	<b>Considerazioni finali</b>	<b>39</b>
6.1	Soddisfacimento degli obiettivi . . . . .	39
6.1.1	Obbiettivi obbligatori . . . . .	39
6.1.2	Obbiettivi desiderabili . . . . .	39
6.1.3	Obbiettivi opzionali . . . . .	39
6.2	Conoscenze acquisite . . . . .	40
6.2.1	JHipster . . . . .	40
6.2.2	Java Enterprise . . . . .	40
6.2.3	Spring . . . . .	40
6.2.4	Hibernate . . . . .	41
6.2.5	Rest . . . . .	41
6.2.6	Oracle . . . . .	41
6.2.7	Liquibase . . . . .	41
6.2.8	Angular . . . . .	42
6.2.9	Eclipse . . . . .	42
6.2.10	Maven . . . . .	42
6.2.11	Git . . . . .	42
6.2.12	SQLDeveloper . . . . .	42
6.3	Valutazione finale . . . . .	43



<i>INDICE</i>	ix
<b>Glossario</b>	<b>45</b>
<b>Acronimi</b>	<b>47</b>
<b>Bibliografia</b>	<b>49</b>

# Elenco delle figure

1.1	Logo SCAI ITEC . . . . .	1
1.2	Logo JHipster . . . . .	3
1.3	Logo Java EE . . . . .	3
1.4	Logo Spring . . . . .	4
1.5	Logo Hibernate . . . . .	4
1.6	Logo REST . . . . .	5
1.7	Logo Oracle . . . . .	5
1.8	Logo Liquibase . . . . .	6
1.9	Logo Angular . . . . .	6
1.10	Logo Eclipse . . . . .	7
1.11	Logo Maven . . . . .	7
1.12	Logo Git . . . . .	7
1.13	Logo SQL Developer . . . . .	8
3.1	Screenshot del backlog presente su Taiga rispetto uno degli sprint terminati	15
4.1	Immagine di preview di un template contenente un testo e un'immagine	21
4.2	Modello ER generato da JDLStudio . . . . .	25
5.1	Schermata di login . . . . .	29
5.2	Schermata di scelta funzionalità . . . . .	30
5.3	Schermata di gestione contenuti . . . . .	30
5.4	Schermata di gestione contenuti . . . . .	31
5.5	Creazione di un contenuto . . . . .	32
5.6	Anteprima di un contenuto di esempio . . . . .	32
5.7	Clonazione di un contenuto . . . . .	33
5.8	Modifica di un contenuto . . . . .	33
5.9	Richiesta di associazione di un contenuto . . . . .	34
5.10	Eliminazione di un contenuto . . . . .	34
5.11	Visualizzazione audit . . . . .	35
5.12	Metriche dell'applicazione . . . . .	38

# Elenco delle tabelle



# Capitolo 1

## Introduzione

### 1.1 L'azienda

**SCAI ITEC**<sup>1</sup> è un'azienda italiana appartenente al **gruppo SCAI**. ITEC si occupa di consulenza, System Integration ed Application management in ambito ICT. L'azienda opera principalmente in settore bancario, assicurativo, industriale e di pubblica amministrazione e servizi.

Gli elementi chiave del successo e della crescita di SCAI ITEC sono:

- vasta e profonda conoscenza delle tecnologie;
- grande attenzione per la soddisfazione del cliente;
- molta esperienza, maturata nel corso del tempo.

ITEC è oggi una delle maggiori realtà nel nord-est del paese in ambito ICT e si propone come partner per qualsiasi tipo di applicazione, progetto e servizio modellato sulle specifiche necessità del cliente.

Grazie alla consolidata esperienza nel ruolo di **system Integrator**<sup>[g]</sup> ed alle soluzioni leader di mercato proposte, ITEC è in grado di garantire ai propri clienti risposte rapide, concrete e qualificate in base alle specifiche esigenze di tipo gestionale e applicativo.

Ultimo, ma non meno importante, dei motivi per cui l'azienda è all'avanguardia rispetto le nuove tecnologie è il grande investimento di ITEC in ricerca, sviluppo e formazione del personale.



**Figura 1.1:** Logo SCAI ITEC

---

<sup>1</sup>SCAI ITEC abbrev. ITEC.

## 1.2 Scopo dello stage

L'obiettivo principale dello stage è stato quello di inserire lo studente all'interno di una nuova progettualità, con un particolare focus sulle tematiche legate alle tecnologie multimediali e alla loro distribuzione. Lo studente, affiancato da un IT Architect, ha avuto la possibilità di partecipare al disegno, alla progettazione e realizzazione di una nuova progettualità. L'obiettivo è stato apprendere le tecnologie e le best practice utilizzate in azienda nel ciclo di vita di un applicativo. La progettualità vista è volta a creare un software per la gestione di [contenuti informativi](#)<sup>[g]</sup>, per la loro creazione, modifica e distribuzione nei vari canali di vendita. Grazie a questa nuova applicazione si potrà:

- creare, modificare eliminare e clonare dei [contenuti informativi](#);
- raggruppare i [contenuti informativi](#) su segmenti di mercato e distribuirli;
- pianificare l'esecuzione e l'aggiornamento dei [contenuti informativi](#) sui vari canali di distribuzione;
- seguire un processo di Authoring (paradigma Editore, Redattore, Supervisore) nella fase di creazione e distribuzione.

In quanto l'intero progetto è di dimensione non indifferente l'obiettivo dello stage è stato di sviluppare le funzionalità relative al ruolo di Editore, più nello specifico:

- creazione di un contenuto;
- modifica di un contenuto;
- eliminazione di un contenuto;
- preparazione di un contenuto per la distribuzione;
- auditing delle azioni effettuate dagli utenti;
- documentazione delle funzionalità implementate.

## 1.3 Tecnologie utilizzate

### 1.3.1 JHipster

**JHipster** è una piattaforma di sviluppo, con uno stack tecnologico ben definito, utilizzata per generare, sviluppare e rilasciare, applicazioni e web services all'avanguardia. Supporta molteplici tecnologie per il frontend, tra le quali Angular, React e Vue. Fornisce inoltre supporto per le applicazioni per dispositivi mobili utilizzando Ionic e React Native. Per quanto riguarda il backend, JHipster supporta spring Boot (con l'ausilio di Java o Kotlin), Micronaut, Quarkus, NodeJS e .NET. Per il rilascio sono adottati i principi di [cloud nativo](#)<sup>[g]</sup>. Il rilascio è inoltre supportato su AWS, Azure, Cloud Foundry, Google Cloud Platform, Heroku ed OpenShift.

L'obiettivo di JHipster è generare applicazioni web o microservizi all'avanguardia, unendo:

- uno stack lato server robusto, ad alte prestazioni e coperto da test;

- un interfaccia utente accattivante, moderna e mobile-first usando Angular, React o Vue e Bootstrap per il CSS;
- un workflow ben definito per fare la build dell'applicazione con Maven o Gradle;
- un architettura a microservizi resiliente, utilizzando i principi di [cloud nativo](#);
- infrastruttura definita come codice, in modo da rendere la distribuzione su cloud veloce.

Nel corso dello stage JHipster è stato utilizzato per generare l'applicazione di base, utilizzata come punto di partenza per lo sviluppo.



**Figura 1.2:** Logo JHipster

### 1.3.2 Java Enterprise

**Java Enterprise**, conosciuto anche come Java EE è un insieme di specifiche che mirano ad estendere Java 8, aggiungendo funzionalità enterprise come elaborazione distribuita e servizi web. Le applicazioni Java EE possono essere eseguite sia come [microservizi](#)<sup>[g]</sup> che su [application server](#)<sup>[g]</sup>. In entrambi i casi vengono gestite: transazionalità, scalabilità, sicurezza e concorrenza. Nel corso dello stage Java EE è stato utilizzato per la programmazione lato backend.



**Figura 1.3:** Logo Java EE

### 1.3.3 Spring

**Spring** è un framework applicativo open source e un container per l'[inversione di controllo](#)<sup>[g]</sup> utilizzato dalla piattaforma Java. Le funzionalità di base possono essere

usate da una qualsiasi applicazione Java, mentre quelle più avanzate sono disponibili solamente per Java Enterprise. Il framework Spring ha a se associati vari moduli, nel corso del progetto sono stati utilizzati principalmente:

- Spring Boot: utilizzato per creare applicazioni basate su spring eseguibili senza la necessità di configurare un web server;
- Spring Data: il cui obiettivo è di facilitare la gestione e l'interazione di applicazioni Java con un database.



**Figura 1.4:** Logo Spring

### 1.3.4 Hibernate

**Hibernate** è un framework per lo sviluppo di applicazioni in Java, utilizzato per gestire e mantenere su un database relazionale un insieme di oggetti Java. Le sue funzionalità principali sono:

- mappare oggetti Java come tabelle su database;
- convertire i campi dati Java a quelli del **DBMS**<sup>[g]</sup> utilizzato;
- generare chiamate SQL e convertire la risposta ottenuta in un oggetto Java.

Tali funzionalità sono state largamente utilizzate nel corso dello stage.



**Figura 1.5:** Logo Hibernate

### 1.3.5 REST<sup>2</sup>

**REST** è un modello architetturale per i sistemi distribuiti. I sistemi rest si basano su HTTP e prevedono una struttura degli URL ben definita, che identifichi univocamente le risorse secondo la convenzione del modello stesso<sup>3</sup>. In REST per il trasferimento di dati vengono utilizzati i metodi HTTP, più nello specifico:

- GET: per il recupero di informazioni;
- POST, PUT, PATCH: per l'inserimento di informazioni;
- DELETE: per l'eliminazione di informazioni.

---

<sup>2</sup>**REST**: acronimo di **R**epresentational **S**tate **T**ransfer.

<sup>3</sup>**Resource Naming**: <https://restfulapi.net/resource-naming/>.



I principi guida di REST sono:

- client-server: separazione dei problemi della UI da quelli di storage dei dati;
- statelessness: ogni richiesta deve avere tutte le informazioni necessarie per il suo processamento;
- cacheable: i client possono memorizzare in cache le risposte, queste devono essere definite esplicitamente o implicitamente cacheable, in modo da evitare il riutilizzo di dati errati;
- uniform interface: utilizzo di un'interfaccia di comunicazione omogenea tra client e server in modo da disaccoppiare e semplificare l'architettura per poterla modificare a blocchi;
- layered system: la struttura del sistema può essere composta da strati gerarchici. In questo caso ogni componente non può "vedere" oltre lo strato con cui sta interagendo;
- code on demand (opzionale): il codice lato client può essere esteso scaricando ed eseguendo applet o script.

Nella definizione di API se queste rispettano tutti i vincoli imposti dall'architettura REST allora possono essere definite RESTful.



**Figura 1.6:** Logo REST

### 1.3.6 Oracle

Oracle database è un DBMS di tipo relazionale prodotto da Oracle corporation.

I database oracle sono noti per offrire performance, scalabilità, affidabilità e sicurezza oltre a poter essere utilizzati sia on premise che nel cloud.



**Figura 1.7:** Logo Oracle

### 1.3.7 Liquibase

**Liquibase** è una libreria open source indipendente dal **DBMS** utilizzato. Durante il periodo di stage è stata utilizzata per tracciare, gestire e applicare le modifiche allo schema del database.



**Figura 1.8:** Logo Liquibase

### 1.3.8 Angular

**Angular** è un framework open source per la progettazione e lo sviluppo di applicazioni web. Le applicazioni angular vengono eseguite interamente a lato client ma grazie alla moltitudine di moduli presenti è possibile integrare un sistema di backend più complesso eseguito lato server. Nel corso dello stage, il frontend, è stato scritto interamente in Angular.



**Figura 1.9:** Logo Angular

## 1.4 Strumenti di sviluppo

### 1.4.1 Eclipse

L'IDE utilizzato per lo sviluppo, previo consiglio del tutor aziendale, è stato **Eclipse**. Eclipse è un ambiente di sviluppo integrato multiplatforma e rientra nella categoria di software libero, distribuito secondo i termini della **Eclipse Public License**.



**Figura 1.10:** Logo Eclipse

### 1.4.2 Maven

**Maven** è uno strumento di gestione di progetti software, è basato su un Project Object Model (POM) e può gestire la build, il reporting e la documentazione di un progetto. Nel corso dello stage Maven è stato utilizzato principalmente per automatizzare la build del progetto, sia in sviluppo che in produzione.



**Figura 1.11:** Logo Maven

### 1.4.3 Git

**Git** è un sistema di controllo di versione distribuito, gratuito ed open source, progettato per gestire progetti di qualsiasi tipo. Nel corso dello stage l'utilizzo di Git è stato affiancato a quello di GitLab, una piattaforma web per la gestione di repository Git.



**Figura 1.12:** Logo Git

#### 1.4.4 SQL Developer

**SQL Developer** è un ambiente di sviluppo integrato per lavorare con SQL nei database Oracle. Nel corso dello stage è stato utilizzato per gestire e testare il database Oracle usato in produzione.



**Figura 1.13:** Logo SQL Developer

### 1.5 Organizzazione del testo

**Il secondo capitolo** descrive gli obiettivi dello stage, la pianificazione del lavoro effettuata a monte e le aspettative personali riguardanti lo stage;

**Il terzo capitolo** approfondisce la metodologia di lavoro e i ruoli adottati dal team di sviluppo;

**Il quarto capitolo** descrive l'analisi, la progettazione iniziale, e le soluzioni adottate durante la codifica;

**Il quinto capitolo** descrive le funzionalità del prodotto finale, la documentazione e i test eseguiti;

**Il sesto capitolo** contiene le considerazioni finali riguardanti lo stage e una valutazione personale sul lavoro svolto;

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

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*<sup>[g]</sup>;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

## Capitolo 2

# Obbiettivi e pianificazione

*In questo capitolo sono descritti gli obiettivi dello stage, la pianificazione del lavoro e le aspettative personali.*

### 2.1 Obbiettivi

L'obiettivo di questo stage è la realizzazione di una Proof of Concept dell'applicazione, in cui vengono rese disponibili le funzionalità descritte in §1.2. Per aumentare la produttività e permettere allo studente di conoscere nuove tecnologie viene utilizzato JHipster. Lo studente verrà inserito in un gruppo di lavoro composto da 4 persone in modo da favorire, oltre alla comprensione delle nuove tecnologie, la capacità di lavorare in un team. Al termine del periodo di stage verrà effettuata una presentazione del prodotto alla direzione dell'azienda.

Nel piano di lavoro, documento la cui stesura è avvenuta prima dell'inizio dello stage, sono stati individuati i seguenti obiettivi suddivisi in obbligatori, desiderabili e opzionali.

#### 2.1.1 Obbiettivi obbligatori

- Ob1: Conoscenza del framework Spring e in particolare Spring MVC REST;
- Ob2: Interazione e gestione database Oracle;
- Ob3: Realizzazione delle funzionalità backend del progetto.

#### 2.1.2 Obbiettivi desiderabili

- D1: Realizzazione delle funzionalità frontend del progetto;
- D2: Conoscenza base sviluppo applicazioni frontend Angular;
- D3: Grado di autonomia nel processo di analisi/sviluppo.

#### 2.1.3 Obbiettivi opzionali

- Op1: Conoscenza base dei strumenti per CI/CD.

## 2.2 Pianificazione

Lo stage prevede una durata di 312 ore complessive corrispondenti a 8 ore di lavoro giornaliero per un periodo di circa 8 settimane. L'orario di lavoro è dal Lunedì al Venerdì, dalle ore 9:00 alle 13:00 e dalle ore 14:00 alle 18:00. L'ora tra le 13:00 e le 14:00 è dedicata alla pausa pranzo.

La pianificazione redatta per il periodo di stage, divisa per obiettivi settimanali, è la seguente:

- **prima settimana:**
  - studio strumenti di sviluppo (Eclipse, Maven, Git);
  - analisi dei requisiti.
- **seconda settimana:**
  - creazione struttura del database;
  - gestione database Oracle.
- **terza settimana:**
  - studio e utilizzo del framework Spring;
  - studio e utilizzo di Hibernate;
  - realizzazione dell'object-relational mapping.
- **quarta settimana:**
  - utilizzo spring MVC e Jackson;
  - realizzazione dei servizi REST.
- **quinta settimana:**
  - studio e utilizzo di elementi avanzati di Oracle;
  - gestione changeset Liquibase;
  - sviluppo di altri servizi REST.
- **sesta settimana:**
  - studio di Angular;
  - sviluppo frontend;
- **settima settimana:**
  - continuous integration e continuous delivery;
  - utilizzo di Sonarqube;
  - controllo qualità del codice.
- **ottava settimana:**
  - gestione cache dell'applicazione;
  - ottimizzazione;

## 2.3 Aspettative personali

Le mie aspettative per quanto riguarda lo stage erano molteplici.

Prima tra tutte la possibilità di lavorare in un'azienda che produce software in modo da poter capire, almeno in parte, come funziona la vita in azienda. In secondo luogo il mio obiettivo era quello di imparare nuove tecnologie e le best practice adottate dall'azienda ospitante, anche grazie alla stretta collaborazione con il mio tutor. Ultima cosa ma non meno importante è la possibilità di farsi conoscere da un'azienda leader nel settore, in modo da poter pensare ad eventuali collaborazioni future.





## Capitolo 3

# Metodologia di sviluppo

*In questo capitolo viene descritta la metodologia di lavoro e i ruoli adottati dal team di sviluppo.*

### 3.1 Scrum

**Scrum** è un framework agile per lo sviluppo, consegna e manutenzione di prodotti software e non. Scrum è progettato per l'utilizzo in team di dimensione ridotta. Di seguito vengono descritti ruoli, fasi e artefatti del framework.

#### 3.1.1 Ruoli

##### **Scrum master**

Lo Scrum Master aiuta il team di sviluppo ad apprendere e applicare Scrum per conseguire valore di business. Lo Scrum Master fa tutto ciò che è in suo potere per aiutare il Team, il Product Owner e l'organizzazione ad avere successo. Lo ScrumMaster non è il manager dei membri del Team, né è un project manager, team leader, o rappresentante del team. Lo scopo dello Scrum Master è:

- aiutare a rimuovere gli ostacoli durante lo sviluppo;
- evitare interferenze esterne;
- aiutare il Team ad adottare al meglio le pratiche di sviluppo agile;
- fare in modo che tutti applichino Scrum nel miglior modo possibile.

##### **Product Owner**

Il Product Owner ha la responsabilità di massimizzare il ritorno sugli investimenti (ROI), di identificare le caratteristiche del prodotto, traducendole in una lista di priorità, di decidere cosa dovrebbe andare in cima alla lista per il prossimo Sprint, e di riassegnare le priorità, aggiornandole con continuità. Il Product owner detiene la responsabilità di profitto del prodotto, se questo è commerciale. In Agile il Product owner rappresenta il cliente e nell'applicazione di Scrum può e deve:

- definire il Product Backlog, le user stories e gli acceptance criteria;

- definire le priorità nel Product Backlog e la data di rilascio del prodotto;
- accettare o rifiutare quanto sviluppato;
- cancellare lo sprint se risulta fallimentare o poco utile.

### Team di sviluppo

Il team di sviluppo è composto da un insieme di persone, in genere meno di 10, e si occupa di sviluppare quanto definito dal product owner. Il team Scrum deve essere "cross-funzionale", ovvero includere tutte le competenze necessarie allo sviluppo del prodotto. I membri del team devono essere proattivi e aperti allo studio di tecnologie che vanno oltre le loro competenze. Il team di sviluppo:

- costruisce il prodotto definito dal Product owner;
- possiede tutte le conoscenze per ottenere un prodotto potenzialmente rilasciabile alla fine di ogni sprint;
- è auto organizzato, con un alto grado di autonomia e responsabilità;
- decide quanti e quali elementi del Product backlog sviluppare;
- ha la responsabilità di sviluppo, test e rilascio del prodotto;
- non possiede un team leader, in quanto in Scrum nel team di sviluppo sono considerati tutti di pari livello.

Nel corso dello stage i ruoli erano così suddivisi:

- Scrum master: Bledar Gogaj
- Team di sviluppo: Alessandro Discalzi, Tania Parolin
- Product owner: Marco Lionello

### 3.1.2 Artefatti

#### Product backlog

Il Product backlog è un elenco di funzionalità, centrate sul cliente e ordinato per priorità; esiste e si evolve per tutta la durata del prodotto. Il Product backlog definisce quindi tutto ciò che deve essere fatto ed include una moltitudine di voci, più nello specifico:

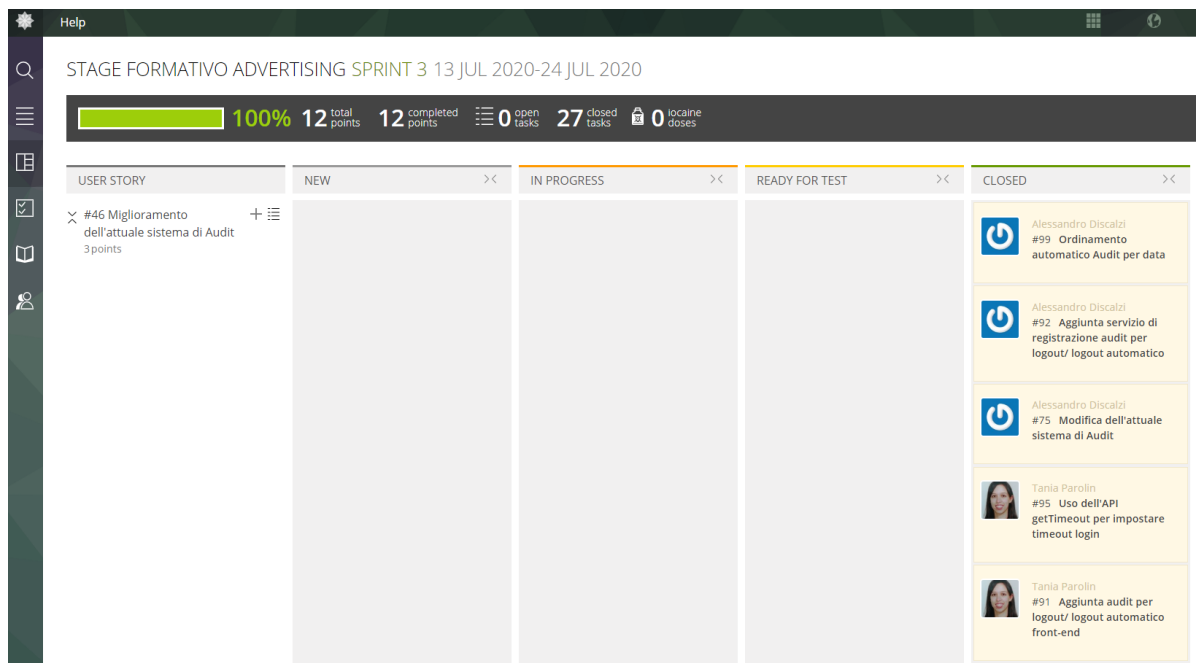
- nuove funzionalità da implementare;
- obiettivi di miglioramento;
- lavori di ricerca;
- difetti da risolvere, se in numero contenuto;

Un buon Product backlog è **DEEP**:

- **dettagliato**, con maggior attenzione alle voci di priorità più alta;

- **stimato:** per ogni voce deve esserci una stima per il completamento, la stima viene definita dal team di sviluppo;
- **emergente:** il Product backlog viene aggiornato in base alla variabilità del progetto;
- **prioritizzato:** le voci sono ordinate per priorità, quelle con priorità più alta forniscono maggior valore al prodotto.

Nel corso del progetto di stage per gestire il Product Backlog è stato utilizzato **Taiga**, un tool per la gestione di progetti agile.



**Figura 3.1:** Screenshot del backlog presente su Taiga rispetto uno degli sprint terminati

Nel Product backlog di AMS la lista delle attività eseguite, in ordine di priorità, è la seguente:

- design e realizzazione della base di dati;
- design e realizzazione del template di base per i contenuti pubblicitari;
- realizzazione della funzione di preview di un contenuto;
- gestione autenticazione e autorizzazione in base ai ruoli;
- realizzazione dei servizi di back-end;
- design e implementazione della **User Interface (UI)**<sup>[8]</sup>;
- integrazione front-end e back-end;
- implementazione della creazione di nuovi contenuti;

- implementazione della modifica di un contenuto;
- miglioramento [UX](#)<sup>[g]</sup> e [UI](#);
- inizio stesura documentazione;
- implementazione sistema di auditing;
- miglioramento dei contenuti di esempio;
- predisposizione di oracle;
- implementazione breadcrumbs;
- aggiunta controlli di validazione;
- completamento della documentazione;
- controlli sull'accessibilità del software;
- miglioramento pipeline di [CI](#)<sup>[g]</sup>;
- predisposizione demo.

### Definition of done

Durante ogni sprint ciò che viene fatto costituisce un prodotto potenzialmente rilasciabile, questo deve essere approvato dal Product owner, dallo Scrum master e dal Team di sviluppo prima dell'inizio dello sprint successivo. La definition of done è un insieme di regole che definiscono quando ciò che viene fatto può essere definito rilasciabile. La definition of done adottata durante lo stage è la seguente:

- AMS (Advertising Management System) deve essere stato compilato senza warning;
- il software deve essere stato deployato nell'ambiente locale senza l'introduzione di nuovi errori e/o warning;
- devono essere stati eseguiti i test sulle nuove funzionalità;
- il codice deve essere stato revisionato dallo Scrum Master/Product Owner;
- la feature implementata deve essere accettata dal Product Owner.

Nei casi in cui questa definizione non fosse applicabile (ad esempio nel caso di modifiche che richiedono più voci/user stories/sprint) avrebbe dovuto essere evidenziata la violazione del Definition of Done allo Scrum Master ed al Product Owner ritardando la chiusura del Done a quando fosse stato effettivamente possibile.

### 3.1.3 Fasi

#### Sprint

Uno sprint è un periodo di tempo ben definito, di solito due settimane o un mese, durante il quale il team di sviluppo completa una parte di lavoro in base a quanto definito nel Product backlog. Gli sprint hanno durata fissa che non può essere estesa, tuttavia se lo sprint risulta fallimentare o obsoleto può essere cancellato prima del termine dal Product owner.

Nel corso del progetto didattico la durata degli sprint è stata fissata a due settimane, mentre la durata e gli argomenti discussi durante le riunioni descritte successivamente hanno seguito le regole di Scrum.

#### Sprint planning

Lo sprint planning è un incontro che viene effettuato prima di ogni sprint, la cui durata è limitata a due ore per ogni settimana di sprint. L'obiettivo dello sprint planning è definire cosa rilasciare al termine del prossimo sprint e come farlo. Il lavoro da fare viene selezionato dal product backlog e inserito nello sprint anche in base alle stime di completamento definite dal team di sviluppo.

#### Daily scrum

Il Daily Scrum è una delle pratiche chiave di Scrum. Si tratta di un meeting giornaliero, della durata massima di 15 minuti, a cui partecipa il team di sviluppo, lo Scrum master e, se richiesto, il Product owner. Il Daily scrum serve a sincronizzare il team e durante questo ogni membro deve rispondere a tre domande:

- cosa è stato fatto dall'ultima riunione?
- cosa sarà fatto prima della prossima riunione?
- quali difficoltà si sono incontrate?

Nel caso alcuni ostacoli necessitino di discussioni approfondite queste possono essere fatte al termine del Daily scrum.

#### Sprint review

La sprint review si tiene alla fine di uno sprint, in modo da ispezionare gli incrementi e aggiornare, se necessario, il Product backlog di conseguenza. Durante la sprint review il team di lavoro e gli stakeholders collaborano per vedere, in base a quanto fatto durante lo sprint, quali sono le prossime cose che si potrebbero fare per aumentare il valore del prodotto. La sprint review dura massimo un'ora per ogni settimana di sprint, durante la riunione si svolgono le seguenti attività:

- il Product owner spiega quali attività del backlog sono state fatte e quali no;
- il team di sviluppo discute di cosa è andato bene, di cosa è andato storto e di come si sono affrontati i problemi durante lo sprint;
- il Team di sviluppo mostra il lavoro fatto e risponde ad eventuali domande riguardo l'incremento;
- se necessario il Product owner decide le date di rilascio in base a quanto fatto;

- il gruppo di lavoro collabora per decidere cosa fare prossimamente, questo serve anche come input allo sprint planning;
- review del potenziale di mercato del prodotto, se il prodotto è commerciale.

**Sprint retrospective**

La sprint retrospective ha luogo dopo la Sprint review e prima dello Sprint planning e la sua durata è limitata a 45 minuti per ogni settimana di sprint. Durante la retrospettiva, che viene vista come una possibilità di miglioramento per il team, si discute di cosa è andato bene, di cosa è andato storto e di come si può migliorare per il prossimo sprint. Questo favorisce il team in quanto si possono migliorare le metodologie adottate nello sprint precedente in base a ciò che è andato storto.

## Capitolo 4

# Analisi e progettazione

*Il seguente capitolo descrive l'analisi, la progettazione iniziale, e le soluzioni adottate durante la codifica*

### 4.1 Analisi e progettazione iniziale

Le funzionalità da implementare e gli obiettivi da raggiungere erano già ben definiti all'inizio del progetto, in quanto l'analisi funzionale è stata fornita dal Product owner insieme ad una prima definizione del product backlog. Per questo motivo la fase di analisi dei requisiti da parte del team di sviluppo è stata molto breve e si è concentrata principalmente su come le funzionalità del prodotto dovessero essere fruibili all'utente finale. La progettazione del software non è avvenuta solamente all'inizio, ma si è protratta durante tutto il periodo di stage. Questo ha garantito maggior flessibilità nell'implementazione delle funzionalità e ha favorito l'utilizzo di Scrum. Tale scelta tuttavia ha anche introdotto il rischio che una debole progettazione iniziale potesse causare problemi nelle settimane a venire. Questo rischio è stato subito mitigato grazie ad un'attenta progettazione della base di dati pensando, nel miglior modo possibile, alle entità, a come esse fossero in relazione tra di loro e alla logica necessaria per implementare le funzionalità richieste. Inoltre, in quanto JHipster utilizza uno stack tecnologico ben definito, e impone dei vincoli architetturali da seguire, tale rischio è ulteriormente ridotto poiché non è necessario progettare l'architettura del prodotto. Nel corso di questo capitolo vengono descritte le principali attività di progettazione svolte durante il corso dello stage.

### 4.2 Progettazione del database

La progettazione del database è stata una delle attività più importanti e deteneva un alto grado di priorità anche nel Product backlog. La progettazione del database è stata divisa in due parti: la prima mirata a individuare le entità partendo dall'analisi dei requisiti fatta dal Product owner, mentre la seconda mettendo in relazione tali entità aggiungendo tabelle se necessario. Per quanto riguarda la vera e propria implementazione del database è stato sfruttato un tool fornito da JHipster ovvero **JDLStudio**, che verrà descritto in modo dettagliato successivamente nel corso di questa sezione.

### 4.2.1 Entità

Durante la progettazione della base di dati sono state definite le entità descritte di seguito. Per ciascuna di esse viene riportata una breve descrizione e la lista dei suoi campi dati corredata da tipo di dato, nella lista seguente non vengono riportate chiavi primarie né chiavi esterne poiché l'inserimento di tali campi dati è automatizzato e verrà descritto dettagliatamente nella sezione relativa alle [relazioni tra entità](#). Mentre si stava effettuando la progettazione del database è sorto il problema di come si volessero salvare i [contenuti informativi](#) e i template. Si è optato di suddividere i contenuti in pagine, ognuna delle quali può contenere uno o più elementi multimediali e/o testuali. Inoltre in uno stesso contenuto si può selezionare un template diverso per ogni pagina. Per quanto riguarda i template, i quali sono definiti a monte dal team di sviluppo, si è scelto di strutturarli definendo un template al cui interno sono contenuti vari item che possono essere sia testuali che multimediali. Questo rende tutto il più modulare possibile, in quanto non pone limiti né in fase di creazione di un contenuto, né nella definizione dei template e nel loro utilizzo.

#### Content

Definisce un contenuto informativo, i suoi campi dati sono:

- name (String): Nome del contenuto;
- state (State): Enumerazione che definisce i possibili stati di un contenuto;
- description (String): descrizione del contenuto informativo;
- slideTime (Integer): durata delle slide di un contenuto;

#### ContentPage

Definisce una pagina inserita in un contenuto. Ogni contenuto può avere più pagine che scorreranno come slide durante la visualizzazione dello stesso; ogni pagina contiene degli item multimediali o testuali. I suoi campi dati sono:

- pageNumber (Integer): numero della pagina all'interno di un contenuto.

#### MultimediaItem

Definisce un item multimediale (immagine o video) contenuto in una pagina. I suoi campi dati sono:

- path (String): percorso del file multimediale caricato sul server, nel caso il caricamento delle immagini avvenga su NAS;
- resolution (Resolution): enumerazione che definisce la risoluzione dell'immagine caricata;
- multimediaFile (Blob): file multimediale caricato su database, nel caso il caricamento non avvenga su NAS;

#### TextItem

Definisce un testo contenuto in una pagina. I suoi campi dati sono:

- text (Blob): contenuto testuale inserito in una pagina.



### Template

Definisce un template che verrà utilizzato dalle pagine di un contenuto. Ogni pagina può utilizzare template definiti a monte che determinano come immagini, video e contenuti testuali vengono disposti nella pagina al momento della presentazione. I suoi campi dati sono:

- `previewImage` (Blob): immagine di preview del template, utilizzata per aiutare l'utente a capire come questo è definito;



**Figura 4.1:** Immagine di preview di un template contenente un testo e un'immagine

- `templateHTML` (Blob): contiene l'html del template;
- `name` (String): Nome del template.

### TemplateItem

Definisce un item all'interno di un template. Per item si intende un singolo testo, immagine o video. Questa entità viene utilizzata in fase di creazione e di visualizzazione di un contenuto ed è utilizzata per sapere cosa va inserito in una pagina che utilizza un determinato template. I suoi campi dati sono:

- `description` (String): descrizione dell'item del template (e.g. video a schermo intero);
- `type` (itemType): enumerazione che definisce il contenuto che viene inserito in quell'area del template;
- `TemplateArea` (Integer): area del template in cui un certo oggetto va inserito, utilizzato in fase di preview per inserire immagini, video e testi al posto giusto all'interno del template.

### CSS

Definisce il CSS utilizzato da uno o più template. I suoi campi dati sono:

- `templateCSS` (Blob): contiene il CSS utilizzato da uno o più template.

### CustomAudit

Entità definita per implementare il sistema di Audit. I suoi campi dati sono:

- `action` (Action): enumerazione che definisce il tipo di azione registrata negli audit;
- `description` (String): descrizione dettagliata dell'azione effettuata;
- `timestamp` (Instant): data e ora in cui l'azione registrata è stata eseguita.

**User**

Entità predefinita in JHipster contenente tutte le informazioni necessarie per registrare un utente. Per utilizzarla è sufficiente definire relazioni ad essa.

**4.2.2 Enumerazioni**

Durante la progettazione della base di dati, in alcuni casi, si è ritenuto necessario definire alcuni campi dati come enumerazione al posto di utilizzare una semplice stringa di testo. Questo garantisce maggior controllo e impone vincoli che evitano il verificarsi di errori relativi all'utilizzo di nomi diversi per esprimere lo stesso concetto. Le enumerazioni definite durante la progettazione del database sono le seguenti.

**State**

Lista dei possibili stati di un contenuto informativo, ovvero: *CREATED*, *TO\_ASSOCIATE*, *ASSOCIATED*, *TO\_AUTHORIZE*, *AUTHORIZED*, *PUBLISHED*.

**Resolution**

Lista delle possibili risoluzioni di un elemento multimediale, ovvero: *HIGH*, *MEDIUM*, *LOW*.

**ItemType**

Lista delle diverse tipologie di template item, ovvero: *IMAGE*, *VIDEO*, *TEXT*.

**Action**

Lista delle azioni da registrare nel sistema di auditing che possono essere effettuate da un utente, ovvero: *CREATE*, *DELETE*, *UPDATE*, *AUTHENTICATION\_SUCCESS*, *AUTHENTICATION\_FAILURE*, *LOGOUT*, *TIMEOUT*.

**4.2.3 Relazioni**

Di seguito vengono evidenziate le relazioni tra le entità descritte in precedenza e ne viene data una breve descrizione.

**Content-ContentPage**

Ogni Content può avere una o più ContentPage, ogni ContentPage è inserita in un solo contenuto.

**Template-ContentPage**

Ogni Template può essere utilizzato da una o più ContentPage, ogni ContentPage utilizza un solo Template.

**Content-page-MultimediaItem**

Ogni ContentPage può contenere uno o più MultimediaItem, ogni MultimediaItem è contenuto in una sola pagina.

**ContentPage-TextItem**

Ogni ContentPage può contenere uno o più TextItem, ogni TextItem è contenuto in una sola pagina.

**Template-TemplateItem**

Ogni Template ha al suo interno uno o più TemplateItem, ogni TemplateItem fa riferimento a un solo Template.

**MultimediaItem-TemplateItem**

Ogni MultimediaItem può riferirsi ad un solo TemplateItem, ad ogni TemplateItem possono essere riferiti uno o più MultimediaItem.

**TextItem-TemplateItem**

Ogni TextItem può riferirsi ad un solo TemplateItem, ad ogni TemplateItem possono essere riferiti uno o più TextItem.

**Content-User**

Ogni Content è creato da un solo User, ogni User può creare uno o più content.

**CSS-Template**

Ogni Template può avere uno o più CSS, ogni CSS può essere utilizzato da uno o più Template.

**CustomAudit-User**

Ogni Audit si riferisce ad un solo User, ad ogni User possono essere registrati uno o più Audit.

#### 4.2.4 Implementazione

**JDL Studio**

JDL Studio è un tool fornito da JHipster che permette di definire lo schema entità-relazioni di un database tramite un apposito linguaggio di scripting chiamato, appunto, JDL. Una delle funzionalità più utili di JDLStudio è la possibilità di visualizzare il modello ER, che viene automaticamente aggiornato ad ogni modifica.

### Definizione di un'entità in JDLStudio

Un'entità viene definita in JDL specificando il suo nome e i suoi campi dati. Un esempio di definizione di un'entità è il seguente:

**Listing 4.1:** Definizione entità Content

```
entity Content {  
    name String unique,  
    state State,  
    description String,  
    slideTime Integer  
}
```

### Definizione di un'enumerazione in JDLStudio

Un'enumerazione viene definita in JDL specificandone il nome e i suoi possibili valori dati. Un esempio di definizione di un'entità è il seguente:

**Listing 4.2:** Definizione enumerazione State

```
enum State {  
    CREATED,  
    TO_ASSOCIATE,  
    ASSOCIATED,  
    TO_AUTHORIZE,  
    AUTHORIZED,  
    PUBLISHED  
}
```

### Definizione di una relazione in JDLStudio

Per definire una relazione in JDL bisogna innanzitutto specificare il tipo di relazione (uno a uno, uno a molti, molti a molti). Fatto questo è sufficiente specificare l'entità di partenza della relazione e quella di arrivo. Un esempio di definizione di relazioni in JDL è il seguente:

**Listing 4.3:** Definizione relazioni uno a molti

```
relationship OneToMany {  
    Content{page} to ContentPage{content},  
    Template{page} to ContentPage{template},  
    ContentPage{multimediaItem} to MultimediaItem{page},  
    TemplateItem{multimediaItem} to MultimediaItem{templateItem},  
    Template{templateItem} to TemplateItem{template},  
    ContentPage{textItem} to TextItem{page},  
    TemplateItem{textItem} to TextItem{templateItem}  
}
```

### Modello ER

Il modello ER autogenerato presenta imperfezioni nella visualizzazione delle relazioni, nonostante ciò è stato ritenuto sufficiente per gli scopi del progetto. Una volta creato il

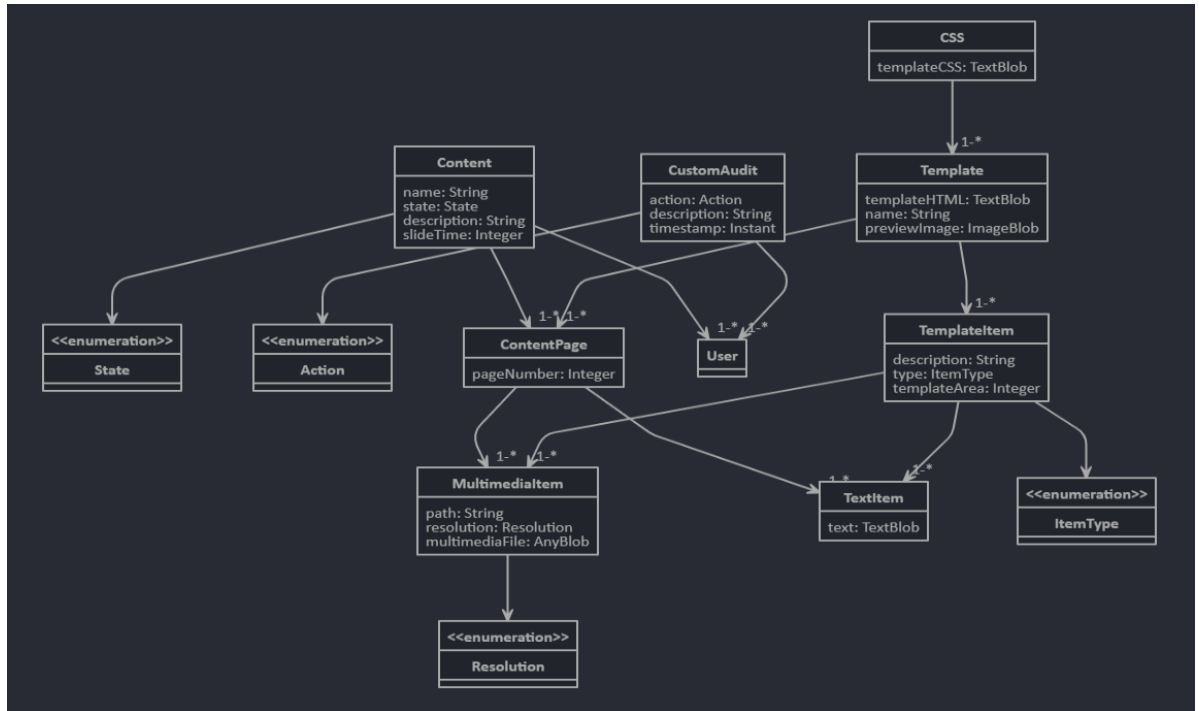


Figura 4.2: Modello ER generato da JDLStudio

modello del database tramite JDL Studio è possibile importarlo direttamente all'interno dell'applicativo. Tale procedura oltre a generare il database creerà anche una struttura di base dell'applicazione completa di back-end e front-end. Seppur questa non risulta utilizzabile evita la scrittura di numerose linee di codice al programmatore riducendo l'introduzione di errori umani.

## 4.3 Definizione dei template per i contenuti

In quanto nella progettazione del database è stato scelto di modellare i template in modo che fossero più modulari ed estendibili possibile, è stata necessaria un'accurata definizione degli stessi. Si è optato per definire un template di base utilizzato da ogni contenuto informativo, e una serie di template personalizzati utilizzati dalle pagine inserite in un contenuto.

### 4.3.1 Template di base

Il template di base definito contiene poche informazioni. Definisce infatti la struttura base della pagina html e la funzione per visualizzare le slide. Contiene inoltre vari

placeholder utilizzati per inserire dinamicamente le informazioni di un determinato contenuto informativo. I placeholder definiti sono:

- `{CSS}`: Utilizzato per inserire il css dei template utilizzati;
- `{CONTENTID}`: Utilizzato per inserire l'id del contenuto come id nel div più esterno;
- `{PAGES}`: Utilizzato per inserire le pagine presenti in un contenuto;
- `{SLIDETIME}`: Utilizzato per impostare lo slidetime scelto in un contenuto;

### 4.3.2 Template di pagina

I template di pagina sono quelli scelti dall'utente durante la creazione di un contenuto. Anche in questo caso i template, scritti in html, avranno al loro interno opportuni placeholder per inserire i dati al momento dell'utilizzo. Poiché non si può sapere a priori come saranno definiti i template futuri sono state definite delle regole da seguire durante la creazione di un nuovo template. La lista delle regole definite è la seguente:

- definire un nome del template univoco;
- anteporre il nome del template ad ogni classe o id utilizzati per il CSS;
- non definire css per gli elementi html, utilizzare solamente classi e id;
- definire un tag `<div>...</div>` differente per ogni item del template;
- nel caso si volessero inserire video o immagini all'interno di un nuovo template seguire gli esempi forniti insieme alla documentazione;
- definire i placeholder per l'inserimento di video, immagini o elementi testuali nel formato `{ItemTypeTemplateArea}`;

## 4.4 Definizione dei servizi di backend

Durante l'importazione del modello del database JHipster genera automaticamente alcuni servizi RESTful per ciascuna entità. Più nello specifico:

- creazione di un record;
- eliminazione di un record;
- modifica di un record;
- ricerca di un record per ID;
- ricerca di tutti i record presenti;

I servizi autogenerati tuttavia non erano sufficienti, alcuni di essi hanno avuto la necessità di essere modificati mentre per altre funzionalità ne sono stati implementati di nuovi. I servizi con la necessità di essere adattati erano, in genere, quelli di modifica. Questi sono stati adattati in modo che gestissero le relazioni correttamente. Sono stati inoltre modificati i servizi di eliminazione, in modo che rispettassero i vincoli di integrità referenziale. I servizi definiti ad hoc sono invece i seguenti:

- servizio di update dello stato di un contenuto;
- servizio di creazione della preview di un contenuto;
- servizio di upload di un file multimediale;
- servizio di ricerca di un file multimediale per ID e che ritorna il file in Base64;





# Capitolo 5

## Prodotto

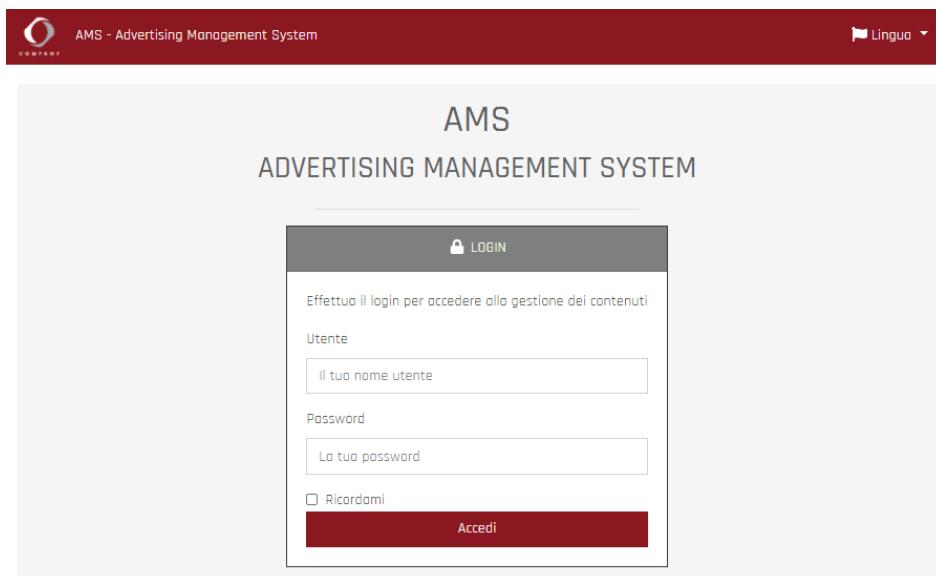
*In questo capitolo viene descritto il prodotto software, la documentazione e i test eseguiti.*

### 5.1 Prodotto software

Nel corso di questa sezione vengono descritte tutte le funzionalità implementate al momento del termine dello stage.

#### 5.1.1 Login

Il login del prodotto finale dovrà essere gestito tramite un sistema di [SSO](#) esterno. Nonostante ciò l'implementazione di tale sistema non era tra gli obiettivi di stage, per la demo finale è stato ritenuto sufficiente utilizzare un [mock](#) dello stesso.



AMS - Advertising Management System

Lingua ▾

### AMS

#### ADVERTISING MANAGEMENT SYSTEM

LOGIN

Effettua il login per accedere alla gestione dei contenuti

Utente

Password

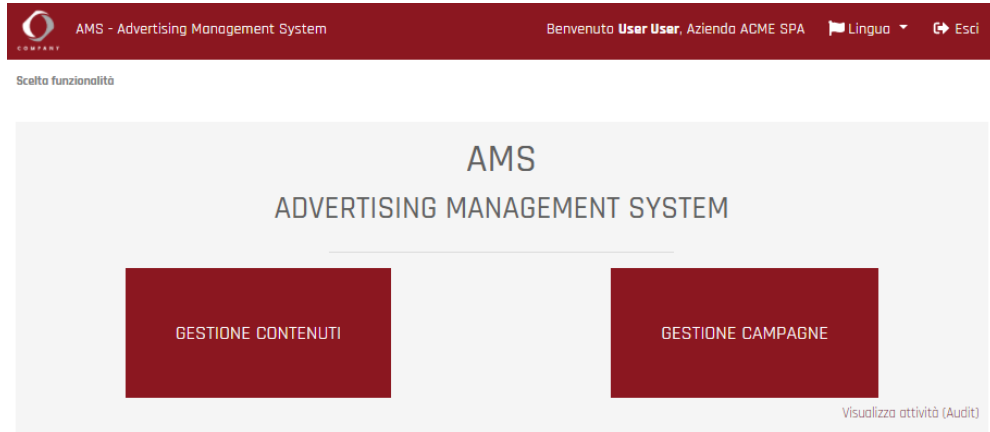
☐ Ricordami

Accedi

**Figura 5.1:** Schermata di login

### 5.1.2 Scelta funzionalità

La schermata principale del prodotto è la schermata di scelta funzionalità. Tramite questa schermata si possono raggiungere le varie funzioni offerte dal prodotto. Più



**Figura 5.2:** Schermata di scelta funzionalità

nello specifico, le funzionalità raggiungibili tramite questa schermata sono:

- gestione dei contenuti;
- gestione delle campagne (Non abilitata);
- visualizzazione audit.

### 5.1.3 Gestione contenuti

Tramite la schermata di gestione contenuti è possibile raggiungere le sezioni di lavoro divise per ruolo (Editore, Redattore, Supervisore).



**Figura 5.3:** Schermata di gestione contenuti

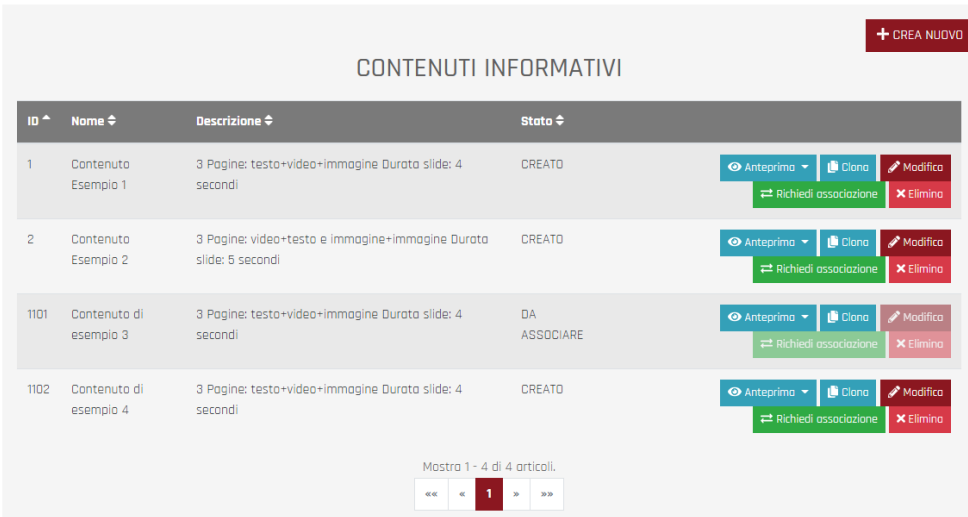
Ovviamente ogni utente può visualizzare solamente le sezioni relative ai ruoli a lui assegnati.

### 5.1.4 Sezione editore

Questa sezione è quella in cui sono utilizzabili la maggior parte delle funzionalità obbiettivo dello stage. In questa schermata è visualizzata una lista di tutti i [contenuti informativi](#). Per ciascuno di questi è possibile:

- visualizzarne l'anteprima;
- clonarlo;
- modificarlo;
- richiederne l'associazione;
- eliminarlo.

Nel caso sia stata richiesta l'associazione di un contenuto, questo non potrà più essere eliminato né modificato. Tramite questa pagina è inoltre possibile creare un nuovo contenuto.



CONTENUTI INFORMATIVI				+ CREA NUOVO	
ID	Nome	Descrizione	Stato		
1	Contenuto Esempio 1	3 Pagine: testo+video+immagine Durata slide: 4 secondi	CREATO	Anteprima	Clona
				Richiedi associazione	Modifica
				Elimina	
2	Contenuto Esempio 2	3 Pagine: video+testo e immagine+immagine Durata slide: 5 secondi	CREATO	Anteprima	Clona
				Richiedi associazione	Modifica
				Elimina	
1101	Contenuto di esempio 3	3 Pagine: testo+video+immagine Durata slide: 4 secondi	DA ASSOCIARE	Anteprima	Clona
				Richiedi associazione	Modifica
				Elimina	
1102	Contenuto di esempio 4	3 Pagine: testo+video+immagine Durata slide: 4 secondi	CREATO	Anteprima	Clona
				Richiedi associazione	Modifica
				Elimina	

Mostra 1 - 4 di 4 articoli.

Figura 5.4: Schermata di gestione contenuti

### 5.1.5 Creazione di un contenuto

La schermata di creazione di un contenuto è divisa in due aree. In quella di sinistra si inseriscono le informazioni generali relative al contenuto. In quella di destra è possibile creare le pagine del contenuto, selezionando per ciascuna uno tra i vari template disponibili. Nel caso si scelgano template con contenuti multimediali è sufficiente che questi vengano caricati attraverso l'apposito pulsante. Nel caso ci siano contenuti testuali da inserire è possibile invece utilizzare l'apposito editor di testo.

**Figura 5.5:** Creazione di un contenuto

### 5.1.6 Anteprima di un contenuto

La possibilità di visualizzare l'anteprima di un contenuto una volta creato è una delle funzionalità più utili e importanti. L'anteprima ermette infatti di visualizzare come sarebbe un contenuto una volta pubblicato e permette di decidere se richiederne l'associazione o modificarlo.

**NUOVA FUNZIONE ONLINE**

**Figura 5.6:** Anteprima di un contenuto di esempio

### 5.1.7 Clonazione di un contenuto

La possibilità di clonare un contenuto è utile nel caso si voglia creare un contenuto simile ad uno già pubblicato senza la necessità di dover partire da zero. Per clonare un contenuto è sufficiente cliccare su *"Clona"* nella sezione dedicata al ruolo di editore. Fatto ciò si aprirà automaticamente un pannello modale in cui è necessario inserire il nome che si vuole assegnare al nuovo contenuto.

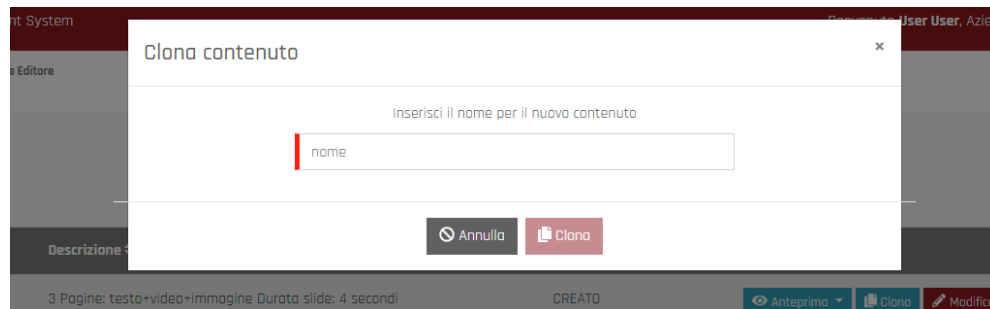


Figura 5.7: Clonazione di un contenuto

### 5.1.8 Modifica di un contenuto

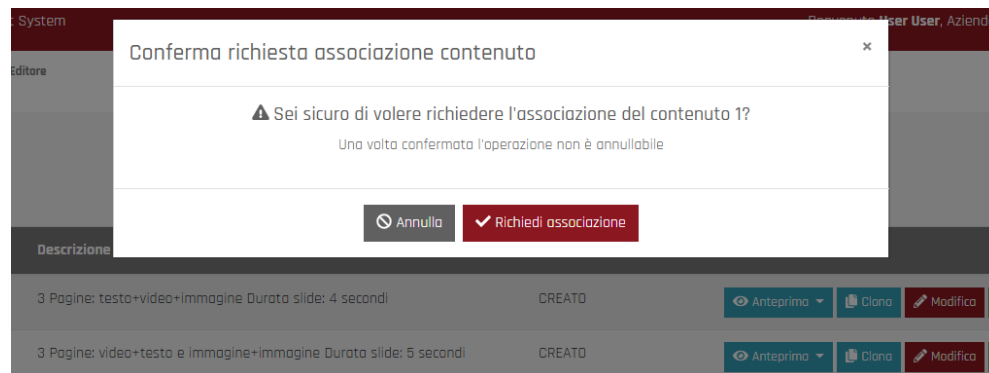
Modificare un contenuto può essere indispensabile nel caso, ad esempio, siano state inserite immagini o testi errati. La schermata di modifica di un contenuto si presenta in modo analogo a quella di creazione con la differenza che i campi dati hanno già le informazioni del contenuto al loro interno.



Figura 5.8: Modifica di un contenuto

### 5.1.9 Associazione di un contenuto

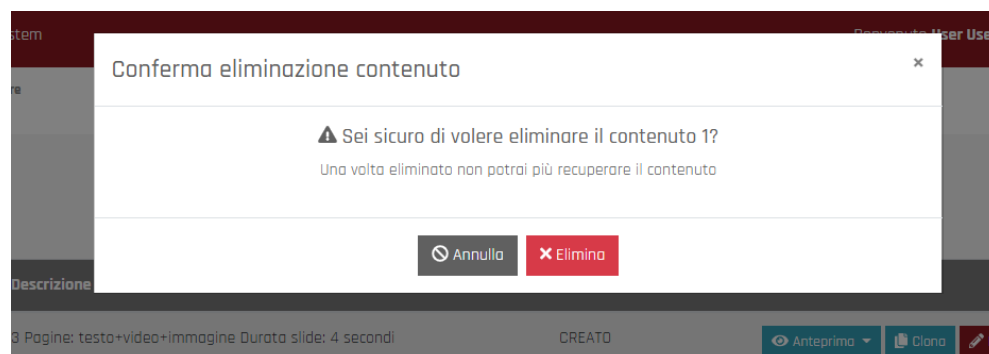
La richiesta di associazione di un contenuto è il primo passo da eseguire perchè questo possa essere pubblicato. Una volta richiesta l'associazione di un contenuto questo non potrà più essere modificato ne eliminato. L'associazione può essere approvata o rifiutata da un redattore; nel primo caso il contenuto potrà proseguire nel suo ciclo di vita, nel secondo tornerà allo stato iniziale e sarà di nuovo modificabile.



**Figura 5.9:** Richiesta di associazione di un contenuto

### 5.1.10 Eliminazione di un contenuto

Un contenuto potrebbe essere creato per errore o non più necessario, in questo caso può tornare utile l'eliminazione. L'eliminazione è irreversibile.



**Figura 5.10:** Eliminazione di un contenuto

### 5.1.11 Audit

La schermata di visualizzazione degli audit, raggiungibile dalla schermata principale, è visualizzabile da chiunque abbia effettuato il *Login* e raccoglie varie informazioni sulle attività effettuate dagli utenti.

VISUALIZZAZIONE ATTIVITÀ (AUDIT)			
Azione ↕	Descrizione ↕	Data e Ora ↕	Utente ↕
AUTHENTICATION_SUCCESS	Authentication success for user: user	17 ago 2020, 18:59:44	user
TIMEOUT	User: user has been automatically logged out because session expired.	17 ago 2020, 18:59:36	user
AUTHENTICATION_SUCCESS	Authentication success for user: user	17 ago 2020, 18:38:40	user
TIMEOUT	User: user has been automatically logged out because session expired.	17 ago 2020, 18:37:48	user
AUTHENTICATION_SUCCESS	Authentication success for user: user	17 ago 2020, 18:18:02	user
TIMEOUT	User: user has been automatically logged out because session expired.	17 ago 2020, 18:15:32	user
AUTHENTICATION_SUCCESS	Authentication success for user: user	17 ago 2020, 18:00:13	user
TIMEOUT	User: user has been automatically logged out because session expired.	17 ago 2020, 18:00:06	user
TIMEOUT	User: user has been automatically logged out because session expired.	17 ago 2020, 18:00:06	user
UPDATE	State of content with ID: 1101 has been changed from: CREATED To: TO_ASSOCIATE.	17 ago 2020, 15:40:23	user
CREATE	A new content with ID: 1102 has been created by cloning content with ID: 1101	17 ago 2020, 15:40:20	user
CREATE	A new content with ID: 1101 has been created by cloning content with ID: 1	17 ago 2020, 15:40:06	user
AUTHENTICATION_SUCCESS	Authentication success for user: user	17 ago 2020, 15:39:45	user
TIMEOUT	User: user has been automatically logged out because session expired.	17 ago 2020, 15:39:38	user
AUTHENTICATION_SUCCESS	Authentication success for user: user	17 ago 2020, 15:14:29	user
AUTHENTICATION_SUCCESS	Authentication success for user: user	17 ago 2020, 15:01:15	user
LOGOUT	User: admin logged out.	17 ago 2020, 15:01:08	admin

Figura 5.11: Visualizzazione audit

## 5.2 Documentazione

Oltre allo sviluppo dell'applicazione vera e propria una parte delle ore a disposizione è stata dedicata alla documentazione. In accordo con il Product Owner è stato deciso di redarre solamente il manuale utente e il manuale sviluppatore.

### 5.2.1 Manuale Utente

Lo scopo di questo documento è illustrare tutte le funzionalità del prodotto. In tal modo l'utente finale ha a disposizione tutte le informazioni per utilizzare il software correttamente. Il manuale utente descrive:

- requisiti di sistema;
- utilizzo dell'applicazione.

### 5.2.2 Manuale Sviluppatore

Lo scopo di questo documento è illustrare le scelte implementative effettuate durante lo sviluppo e dare informazioni utili ad uno sviluppatore che comincia a lavorare a questo progetto. Il manuale sviluppatore tuttavia non ha lo scopo di sostituirsi alla documentazione ufficiale delle tecnologie utilizzate. Per tale motivo all'interno di esso sono contenuti numerosi riferimenti a documenti esterni. Il manuale sviluppatore descrive:

- setup dell'ambiente di lavoro;
- gestione delle dipendenze Angular;
- gestione della dipendenza ad Oracle;
- utilizzo di Angular CLI;
- build e packaging dell'applicazione;
- testing;

- utilizzo di Docker;
- continuous integration;
- scelte implementative effettuate.

## 5.3 Test

Nello sviluppo di un'applicazione il testing è una delle fasi più importanti. Poiché ogni due settimane veniva presentata una demo dell'applicativo i test sono stati eseguiti di continuo durante lo sviluppo, sia in modo automatico che non. Nel corso di questa sezione vengono descritti i test svolti durante lo sviluppo.

### 5.3.1 Test di unità

I test di unità sono eseguiti per verificare se sono presenti errori nei singoli metodi. Nel corso dello stage i test di unità sono stati eseguiti tramite JUnit e Jest. Tali framework sono sviluppati rispettivamente per Java e per Javascript. Il primo è stato utilizzato per testare il back-end mentre il secondo per il front end.

**Listing 5.1:** Test di unità in Java

```
@Test
@Transactional
public void createContent() throws Exception {
    int databaseSizeBeforeCreate = contentRepository.findAll()
        ().size();
    // Create the Content
    restContentMockMvc.perform(post("/api/contents")
        .contentType(MediaType.APPLICATION_JSON)
        .content(TestUtil.convertObjectToJsonBytes(content)))
        .andExpect(status().isCreated());

    // Validate the Content in the database
    List<Content> contentList = contentRepository.findAll();
    assertThat(contentList).hasSize(databaseSizeBeforeCreate
        + 1);
    Content testContent = contentList.get(contentList.size()
        - 1);
    assertThat(testContent.getName()).isEqualTo(DEFAULT_NAME)
        ;
    assertThat(testContent.getState()).isEqualTo(DEFAULT_
        STATE);
    assertThat(testContent.getDescription()).isEqualTo(
        DEFAULT_DESCRIPTION);
    assertThat(testContent.getSlideTime()).isEqualTo(DEFAULT_
        SLIDE_TIME);
}
```



Listing 5.2: Test di unità in JavaScript

```
describe('Component Tests', () => {
describe('Home Component', () => {
  let comp: HomeComponent;
  let fixture: ComponentFixture<HomeComponent>;
  let accountService: AccountService;
  let loginModalService: LoginModalService;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      imports: [AmsTestModule],
      declarations: [HomeComponent],
    })
    .overrideTemplate(HomeComponent, '')
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(HomeComponent);
    comp = fixture.componentInstance;
    accountService = TestBed.get(AccountService);
    loginModalService = TestBed.get(LoginModalService);
  });

  it('Should call accountService.getAuthenticationState on init', () => {
    // WHEN
    comp.ngOnInit();

    // THEN
    expect(accountService.getAuthenticationState).
      toHaveBeenCalled();
  });

  it('Should call accountService.isAuthenticated when it checks authentication', () => {
    // WHEN
    comp.isAuthenticated();

    // THEN
    expect(accountService.isAuthenticated).toHaveBeenCalled();
  });

  it('Should call loginModalService.open on login', () => {
    // WHEN
    comp.login();

    // THEN
    expect(loginModalService.open).toHaveBeenCalled();
  });
});
});
```

### 5.3.2 Test di integrazione

I test di integrazione vengono eseguiti per verificare che non ci siano errori di integrazione tra le varie componenti del sistema. Nel corso del progetto questa fase di test è stata ritenuta di grande importanza, in particolare nell'integrazione tra il back-end e il front-end. In questo caso i suddetti test sono stati eseguiti manualmente.

### 5.3.3 Test di sistema

I test di sistema vengono eseguiti per assicurare che il sistema soddisfi i requisiti richiesti. Nel corso dello stage dopo l'implementazione di una nuova funzionalità l'intero sistema veniva testato. Al termine di ogni sprint tali test venivano eseguiti dal product owner, in tal caso questi possono essere considerati test di accettazione.

### 5.3.4 Test di performance

I test di performance vengono eseguiti per controllare che il sistema non risulti lento durante il suo utilizzo. Tali test sono stati eseguiti con l'ausilio del pannello amministratore generato da JHipster. Questo permette di visualizzare una moltitudine di informazioni come ad esempio le risorse occupate o il tempo di esecuzione delle API.

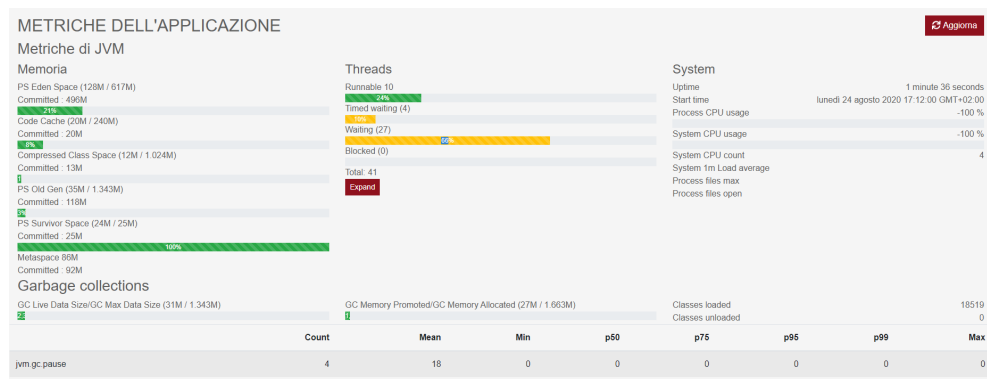


Figura 5.12: Metriche dell'applicazione

## 5.4 Accessibilità

I test di accessibilità sono necessari per verificare che l'applicativo risulti utilizzabile anche da utenti con disabilità. Nel corso del progetto si è cercato di attenersi alle regole definite nello standard **WCAG 2.1**. I test veri e propri sono stati poi eseguiti tramite *Lighthouse*, un tool per l'analisi dinamica di pagine web.

## Capitolo 6

# Considerazioni finali

*In questo capitolo sono esposte le considerazioni finali sul lavoro svolto corredate da una valutazione personale sullo stage.*

### 6.1 Soddisfacimento degli obiettivi

#### 6.1.1 Obiettivi obbligatori

- Ob1: soddisfatto. Ho imparato informazioni di base e non sull'utilizzo e il funzionamento di Spring. Inoltre ho capito il funzionamento di Spring MVC REST anche grazie al confronto diretto col tutor aziendale;
- Ob2: soddisfatto. Il database oracle è stato implementato e ho compreso come interagirci. Ho anche avuto la possibilità di evidenziare le differenze di Oracle rispetto ad altri [DBMS](#) di tipo relazionale;
- Ob3: Tutte le funzionalità di back-end richieste sono state implementate e testate. Il prodotto finale consente infatti di eseguire tutte le operazioni evidenziate nell'analisi funzionale.

#### 6.1.2 Obiettivi desiderabili

- D1: soddisfatto. Il front-end del prodotto è stato implementato e testato. Il front-end è correttamente integrato con il back-end;
- D2: soddisfatto. Angular è stato utilizzato per tutto lo sviluppo del front-end. Ho avuto modo di vedere come utilizzare le librerie di Angular e come modificare la presentazione di una pagina;
- D3: soddisfatto. Il team di sviluppo è stato ritenuto autonomo e proattivo da parte dei rispettivi tutor aziendali.

#### 6.1.3 Obiettivi opzionali

- Op1: parzialmente soddisfatto. Ho avuto modo di comprendere il funzionamento e l'utilità di un processo di CI/CD. Ho capito il funzionamento della pipeline

di CI/CD presente nella repository su GitLab tuttavia non ho avuto modo di implementarne una da zero.

## 6.2 Conoscenze acquisite

Di seguito verranno elencate le conoscenze acquisite per ciascuna delle tecnologie utilizzate nel progetto e descritte nelle sezioni [1.3](#) e [1.4](#).

### 6.2.1 JHipster

JHipster è stato utilizzato come base di partenza per la creazione dell'applicazione. Nello specifico ho imparato a:

- generare applicazioni con JHipster;
- importare un modello di database in un applicazione autogenerata;
- gestire i problemi legati all'aggiunta di nuove entità nel database;
- conoscere l'architettura definita dai programmatori di JHipster.

### 6.2.2 Java Enterprise

Il back-end dell'applicazione è interamente scritto in Java. Nel corso dello stage ho potuto apprendere:

- costrutti avanzati di Java;
- utilizzo di nuove librerie per lo sviluppo di applicazioni web;
- utilizzo di richieste multipart per gestire il caricamento di elementi multimediali;
- scrittura di codice Java più leggibile.

### 6.2.3 Spring

L'utilizzo di spring è stato fondamentale per lo sviluppo dell'applicazione. Durante lo sviluppo gli argomenti trattati sono stati i seguenti:

- utilizzo di spring MVC per separare la logica dalla presentazione;
- utilizzo di Spring data per facilitare l'interazione con il database e garantire la transazionalità delle operazioni;
- utilizzo di spring boot in modo da avviare l'applicazione all'interno del container Spring, senza necessità di configurare un web server;
- funzionalità core di Spring;

### 6.2.4 Hibernate

Hibernate ha facilitato di molto l'interazione col database, è stato infatti utilizzato per:

- mappare le entità definite in java come tabelle su Oracle;
- gestire query effettuate ottenendo come risposta un oggetto Java;
- gestire la conversione dei campi dati da quelli utilizzati da Java a quelli utilizzati da Oracle.

### 6.2.5 Rest

Le [Application Program Interface](#) sono state definite seguendo il modello definito da REST. Durante la codifica delle [Application Program Interface](#) ho avuto modo di:

- comprendere meglio le differenze tra i metodi HTTP (GET, PUT, POST, PATCH, DELETE);
- definire [Application Program Interface](#) RESTful;
- evitare e prestare più attenzione ai problemi di sicurezza legati alla logica del codice scritto;
- scrivere codice che termina *"gracefully"*, ovvero che anche in caso di errori gravi non causa interruzioni al sistema.

### 6.2.6 Oracle

Durante l'implementazione di Oracle sono stati trattati diversi argomenti, alcuni non direttamente correlati col database stesso ma comunque degni di nota. Più nello specifico:

- aggiunta e utilizzo dei driver Oracle a un progetto Java;
- studio delle differenze tra Oracle e altri [DBMS](#);
- configurazione e utilizzo di una VPN per l'accesso all'intranet aziendale.

### 6.2.7 Liquibase

Liquibase ha permesso di gestire in modo semplice ed efficace le modifiche allo schema del database. Durante lo stage ho imparato a:

- modificare manualmente i changelog di Liquibase;
- inserire nuove entità e relazioni nello schema;
- comprendere e risolvere conflitti nei changelog.

### 6.2.8 Angular

Il front-end di AMS è interamente scritto in Angular. Gli argomenti trattati sono stati:

- importazione e utilizzo di moduli Angular;
- modifiche all'interfaccia grafica;
- inserimento di traduzioni automatiche per gestire l'internazionalizzazione;
- integrazione di [Application Program Interface](#) RESTful con il front-end.

### 6.2.9 Eclipse

Nonostante avessi già utilizzato Eclipse in precedenza ho comunque acquisito nuove conoscenze di tale strumento, più nello specifico:

- gestione di un progetto Maven con Eclipse;
- debug dell'applicazione.

### 6.2.10 Maven

La fase di build del progetto è stata gestita interamente con Maven. Ho avuto modo di apprendere:

- il funzionamento del file POM per la gestione delle dipendenze;
- la build dell'applicazione tramite Maven;
- l'esecuzione dei test automatici tramite Maven;

### 6.2.11 Git

Per gestire il versionamento del software è stata utilizzata una repository su GitLab. Durante lo sviluppo è stato necessario apprendere al meglio l'utilizzo di GIT per gestire il versionamento e i conflitti, più nello specifico:

- utilizzo di git da linea di comando;
- utilizzo del [Feature branch workflow](#)<sup>[g]</sup>;
- gestione dei conflitti durante il Merge;
- rilascio di una nuova versione del software.

### 6.2.12 SQLDeveloper

Durante il corso del progetto questo strumento è stato utilizzato per:

- verificare che i dati fossero inseriti correttamente nel database;
- verificare che le query sul database restituissero il risultato aspettato.

### 6.3 Valutazione finale

Sono molto contento dell'esperienza fatta durante il periodo di stage. Ho avuto la possibilità di lavorare all'interno di un'azienda assieme ad un team di lavoro composto da un altro stagista e dai due tutor. Questo ha favorito l'apprendimento di competenze tecniche e, soprattutto, ha migliorato la mia capacità di lavorare in gruppo in un contesto lavorativo. Per quanto riguarda il prodotto tutti gli obiettivi prefissati sono stati completati e alla fine dello stage è stata fatta una presentazione del prodotto al direttore. La presentazione è andata bene e tutte le funzionalità richieste sono state mostrate, al termine di essa ci sono state fatte alcune domande con una modalità simile a un colloquio. Qualche giorno dopo sono stato nuovamente contattato dall'azienda che mi ha fatto una proposta di assunzione, spronandomi prima a terminare gli studi. Questo mi ha reso molto fiero del lavoro svolto, reputo quindi questa esperienza del tutto positiva.





# Glossario

**API** il termine *Application Programming Interface API* indica un insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. 41, 42, 45

**Application Server** un *Application Server* è un server che offre le funzionalità necessarie per l'*hosting* di un'applicazione web. 3, 45

**CI** in ingegneria del software la *CI* (o Continuous integration) è una pratica che si applica in contesti in cui lo sviluppo del software avviene attraverso un sistema di controllo versione. Consiste nell'allineamento frequente dagli ambienti di lavoro degli sviluppatori verso l'ambiente condiviso. 16, 45

**Cloud nativo** le tecnologie sviluppate nativamente nel cloud permettono di creare ed eseguire applicazioni in ambienti scalabili, dinamici e moderni. Questo approccio è esemplificato da contenitori, mesh dei servizi, i microservizi, l'infrastruttura non modificabile e le API dichiarative. Tali tecniche consentono di avere sistemi resilienti, gestibili e osservabili, consentendo di apportare modifiche a un'elevata frequenza. 2, 3, 45

**Contenuto informativo** contenuto pubblicitario che può essere visualizzato su sistemi di advertising multimediale situati in luoghi pubblici come, ad esempio, cartelloni o totem pubblicitari. 2, 20, 31, 45

**DBMS** in informatica un *DBMS* (o Database Management System) è un sistema progettato per consentire la creazione, la manipolazione e l'interrogazione a un database. 4-6, 39, 41, 45

**Feature branch workflow** il *Feature branch workflow* è un flusso di lavoro utilizzato su Git, tale metodo prevede che per ogni feature che si vuole aggiungere al prodotto è necessario creare un nuovo branch. Una volta che il lavoro su una feature è concluso si può procedere con una merge request sul ramo di sviluppo. 42, 45

**inversione di controllo** in ingegneria del software l'*inversione di controllo* (o IoC), è un pattern secondo il quale un componente di livello applicativo riceve il controllo da parte di un componente appartenente ad una libreria.. 3, 45

**Microservizi** un architettura a *microservizi* organizza un'applicazione come una raccolta di servizi. In tale architettura ciascuna funzione è implementabile in modo indipendente, in questo modo un servizio non funzionante non rischia di compromettere gli altri. [3](#), [46](#)

**Mock** un *mock* è un oggetto simulato che riproduce, in modo controllato, lo stesso comportamento di un oggetto reale. [29](#), [46](#)

**SSO** il *Single Sign On* è una modalità di controllo degli accessi che prevede l'utilizzo di un'unica autenticazione valida su diversi sistemi informatici. [29](#), [46](#)

**System integrator** un *system integrator* si occupa di far dialogare diversi impianti e tecnologie, con lo scopo di creare una nuova struttura che implementi e migliori le funzionalità di quelle di origine. [1](#), [46](#)

**UX** *UX* (o User Experience) è un termine utilizzato per definire la relazione tra un utente e il prodotto. Un prodotto per aver una buona user experience deve cercare di comprendere i bisogni dell'utente ed aiutarlo nell'utilizzo del prodotto stesso. [16](#), [46](#)

# Acronimi

UI [UI. 15](#), [16](#), [47](#)



# Bibliografia

## Riferimenti bibliografici

James P. Womack, Daniel T. Jones. *Lean Thinking, Second Editon*. Simon & Schuster, Inc., 2010.

## Siti web consultati

*Manifesto Agile*. URL: <http://agilemanifesto.org/iso/it/>.