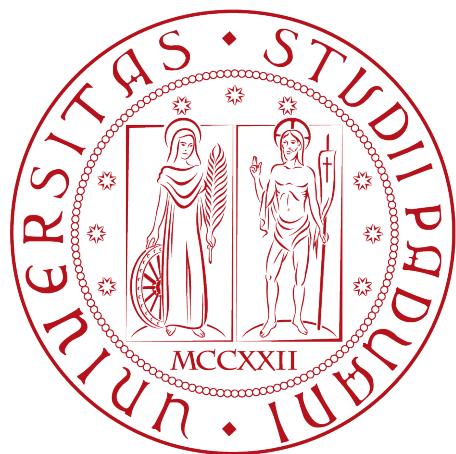


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA

CORSO DI LAUREA IN INFORMATICA



## Recommendation system

*Tesi di laurea triennale*

*Relatore*

Prof. Tullio Vardanega

*Laureando*

Alberto Andeliero

---

ANNO ACCADEMICO 2015-2016



Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live

— John Woods

Dedicato a tutta la mia famiglia che mi ha dato questa possibilità e sostenuto nel momento del bisogno. Inoltre questa dedica va anche a tutte le persone che mi stanno accanto, specialmente ai miei amici.



# Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, dal laureando Alberto Andeliero presso l'azienda Nextep S.r.l. a Carmignano Di Brenta(PD).



# Indice

<b>1 L’azienda</b>	<b>1</b>
1.1 Presentazione azienda . . . . .	1
1.1.1 L’azienda . . . . .	1
1.1.2 Prodotti e servizi . . . . .	2
1.1.3 Tecnologie di riferimento . . . . .	3
1.2 Processi aziendali . . . . .	5
1.2.1 Metodologia agile . . . . .	5
1.3 Strumenti a supporto dei processi . . . . .	7
1.3.1 Gestione di progetto . . . . .	7
1.3.2 Documentazione . . . . .	8
1.3.3 Sistema di versionamento . . . . .	8
1.3.4 Ambiente di sviluppo . . . . .	8
1.3.5 Sistemi operativi . . . . .	9
1.4 Clientela tipo e propensione all’innovazione . . . . .	9
<b>2 Il progetto nella strategia aziendale</b>	<b>11</b>
2.1 L’azienda e gli stage . . . . .	11
2.2 Il progetto . . . . .	12
2.2.1 Il progetto . . . . .	12
2.2.2 Obiettivi . . . . .	12
2.3 Vincoli . . . . .	13
2.3.1 Vincoli temporali . . . . .	13
2.3.2 Vincoli metodologici . . . . .	13
2.3.3 Vincoli tecnologici . . . . .	15
2.4 Obiettivi personali . . . . .	17
<b>3 Il progetto di stage</b>	<b>19</b>
3.1 Organizzazione . . . . .	19
3.1.1 Pianificazione . . . . .	20
3.2 Analisi dei requisiti . . . . .	20
3.2.1 Requisiti . . . . .	20
3.2.2 Casi d’uso . . . . .	22
3.3 Progettazione architetturale . . . . .	25
3.3.1 Visione ad alto livello . . . . .	25
3.3.2 Architettura . . . . .	26
3.3.3 Database . . . . .	29
3.3.4 Design pattern . . . . .	31
3.4 Progettazione di dettaglio . . . . .	34

3.4.1	Controllo stato . . . . .	34
3.4.2	Inserimento behavior . . . . .	35
3.4.3	Raccomandazione . . . . .	35
3.5	Verifica e Validazione . . . . .	36
3.5.1	Resoconto risultati . . . . .	37
3.6	Sommario funzionamento . . . . .	39
<b>4</b>	<b>Valutazione Retrospettiva</b>	<b>43</b>
4.1	Bilancio sui risultati . . . . .	43
4.1.1	Obiettivi conseguiti . . . . .	43
4.1.2	Obiettivi personali . . . . .	44
4.2	Bilancio formativo . . . . .	44
4.3	Distanza tra formazione universitaria e lavoro . . . . .	46
	<b>Glossary</b>	<b>47</b>
	<b>Acronyms</b>	<b>49</b>
	<b>Bibliografia</b>	<b>51</b>

# Elenco delle figure

1.1	Logo di Nextep srl . . . . .	1
1.2	Sito Associazione Maestri Sci Italiani: realizzato da Nextep. Immagine tratta da <a href="#">Nextep</a> . . . . .	2
1.3	Classificazione delle tecnologie utilizzate . . . . .	3
1.4	Diagramma della metodologia di sviluppo <i>agile</i> . Immagine tratta da <a href="#">GlobalTeckz</a> . . . . .	5
1.5	Diagramma del modello <i>Scrum</i> . Immagine tratta da <a href="#">Wikipedia</a> . . . . .	7
1.6	Piattaforma di gestione di progetto Jira . . . . .	8
1.7	Classificazione degli strumenti utilizzati . . . . .	9
2.1	Esempio di un albero decisionale . . . . .	12
2.2	Pannello di controllo travis . . . . .	14
2.3	Tipica architettura di un'applicazione che utilizza Play Framework. Immagine tratta da <a href="#">Omer Haderi Blog</a> . . . . .	15
2.4	Esempio di modellazione in OrientDb. Immagine tratta da <a href="#">Scalac Blog</a>	17
3.1	Suddivisione delle attività . . . . .	19
3.2	Distribuzione requisiti in percentuale per tipologia . . . . .	21
3.3	Distribuzione requisiti in percentuale per importanza . . . . .	22
3.4	UC0: Tres . . . . .	23
3.5	UC1: Tres . . . . .	24
3.6	Visione generale dell'architettura . . . . .	25
3.7	Diagramma del <i>package tres</i> . . . . .	26
3.8	Diagramma del <i>package controllers</i> . . . . .	27
3.9	Diagramma del <i>package models</i> . . . . .	27
3.10	Diagramma del <i>package commons</i> . . . . .	28
3.11	Diagramma del <i>package storage</i> . . . . .	28
3.12	Diagramma del <i>package algorithm</i> . . . . .	29
3.13	Modello a grafo del database . . . . .	30
3.14	Contesto di utilizzo del <i>design pattern Model View Controller (MVC)</i> .	31
3.15	Contesto di utilizzo del <i>design pattern Data Access Object (DAO)</i> .	31
3.16	Contesto di utilizzo del <i>design pattern strategy</i> . . . . .	32
3.17	Contesto di utilizzo del <i>design pattern singleton</i> . . . . .	33
3.18	Diagramma di sequenza: controllo stato. . . . .	34
3.19	Diagramma di sequenza: inserimento comportamento. . . . .	35
3.20	Diagramma di sequenza: raccomandazione. . . . .	36
3.21	Diagramma di sequenza: raccomandazione. . . . .	39

4.1	Riassunto requisiti . . . . .	43
-----	-------------------------------	----

## Elenco delle tabelle

# Capitolo 1

## L’azienda

*Questo capitolo tratta dettagliatamente della azienda ospitante, il suo business, l’organizzazione interna e l’innovazione.*

### 1.1 Presentazione azienda

Questa sezione descrive l’azienda che ha ospitato lo *stage*, elencando generalità e la struttura generale dell’azienda. Inoltre si illustrano alcuni prodotti e i servizi offerti da Nextep srl, alla propria clientela.

#### 1.1.1 L’azienda

L’azienda con cui ho svolto lo stage formativo è Nextep srl<sup>1</sup>, la cui sede operativa si trova a Carmignano Di Brenta(PD). Nextep è una società fondata nel 2000 che opera nel settore dell’informatica e della comunicazione. Il *core business* dell’azienda, è riconosciuto nelle attività di progettazione, realizzazione e gestione di infrastrutture di *Information and Digital Communication Technology*. L’organico di Nextep srl è formato da una ventina di persone, suddivise tra *web marketing* e tecnici (sviluppatori, grafici e sistemisti). L’ambiente di lavoro è condiviso con altre due aziende: Allos e Zero12. Allos si occupa della formazione del capitale umano di organizzazioni medio grandi, mentre Zero12 si occupa di servizi *cloud* e applicazioni *mobile*. Il risultato è un ambiente di *co-working*, dove le competenze del singolo vengono messe a disposizione di tutti, in modo da facilitare la crescita professionale dei dipendenti delle varie aziende. Non è raro che le tre aziende collaborino tra di loro per la realizzazione di alcuni progetti.



**figura 1.1:** Logo di Nextep srl

---

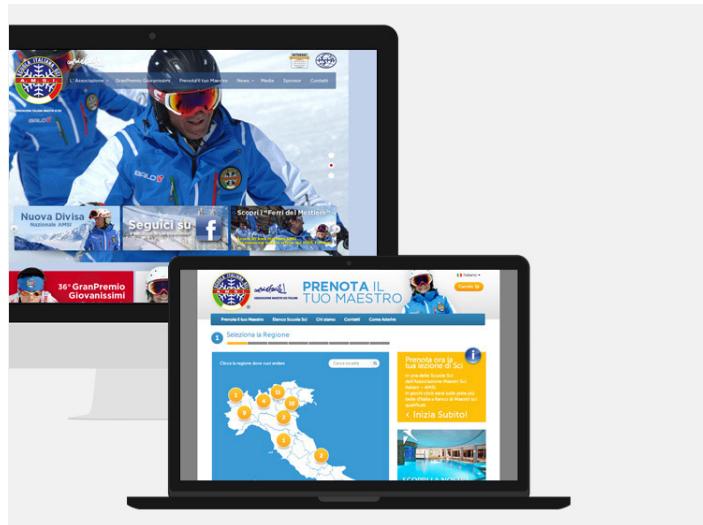
<sup>1</sup><http://www.nextep.it/>.

### 1.1.2 Prodotti e servizi

Il principale settore di attività dell’azienda è il *Web*. Nextep è specialista nella progettazione e sviluppo di siti *web*, portali, *social intranet* e soluzioni di *e-commerce*. L’azienda fornisce sistemi per il *knowledge management*, che sono dei sistemi per raccogliere, sviluppare, conservare e rendere accessibile la conoscenza delle persone che fanno parte di una organizzazione. Si occupa inoltre dello sviluppo di applicazioni mobile. L’azienda pone il suo focus, nella realizzazione di sistemi con interfacce innovative e soprattutto accessibili all’utente, grazie alle competenze acquisite nel ramo della *User Interface* e *User Experience*. Realizza sistemi per l’integrazione con tecnologie e servizi di *Cloud Computing*.

I principali servizi offerti da Nextep srl ai propri clienti sono:

- Fornitura di servizi a canone (*server virtuali*, applicazioni *web*, servizi di *monitoring* di infrastrutture *Information Communication Technology (ICT)*, *backup online*, canoni di assistenza e manutenzione dei sistemi, canoni di gestione dei sistemi);
- Elaborazione dati, analisi e supporto decisionale in ambito *web marketing*, *social marketing* e *social intranet*;
- Servizio Clienti, attività di supporto tecnico e di gestione di infrastrutture *ICT* e siti *web*.



**figura 1.2:** Sito Associazione Maestri Sci Italiani: realizzato da Nextep. Immagine tratta da [Nextep](#)

In particolare l’azienda, fornisce servizi per migliorare l’efficacia delle strategie di comunicazione *web*, dedicando particolare attenzione alla reputazione e all’identità digitale. Collabora, inoltre, assieme alle aziende nella gestione delle informazioni digitali, della sicurezza e della disponibilità dei dati e delle applicazioni. Infine nel ramo del *marketing*, Nextep fornisce consulenza *web marketing*, *social marketing* e *web analytics*.

### 1.1.3 Tecnologie di riferimento

Essendo il *web* un settore soggetto a continue evoluzioni, le tecnologie utilizzate cambiano col tempo. Nextep cerca di aggiornare le tecnologie utilizzate, in modo da fornire ai propri clienti prodotti sempre migliori e innovativi.

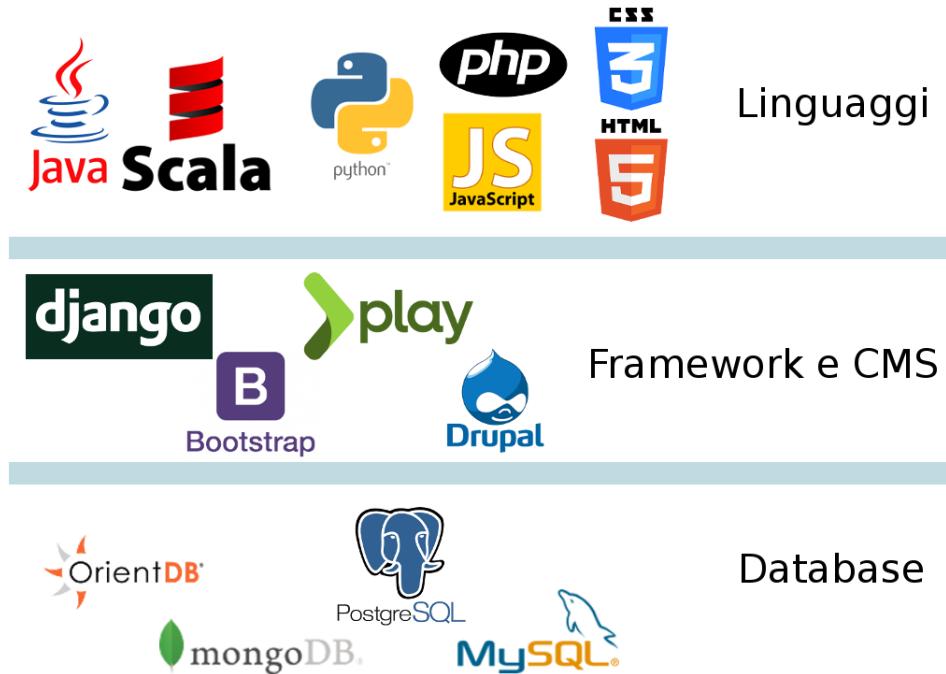


figura 1.3: Classificazione delle tecnologie utilizzate

#### Linguaggi di programmazione

L’azienda lavora principalmente in ambito *web*, quindi utilizza i linguaggi di programmazione più moderni per lo sviluppo dei prodotti. I linguaggi di programmazione più utilizzati in ordine di importanza sono: Php, Python, Java e Scala. Per quanto riguarda le interfacce grafiche naturalmente utilizza tecnologie quali Html5, Css3 e Javascript.

#### Framework e CMS

**Django** è un *web framework open source* per lo sviluppo di applicazioni *web*, scritto in linguaggio Python, seguendo il *pattern MVC*. Fornisce nativamente delle applicazioni, per la gestione dei contenuti e degli accessi.

**Play Framework** è un *web framework open source* scritto in Java e Scala, che implementa il *pattern MVC*. Il suo scopo è di migliorare la produttività degli sviluppatori, usando il paradigma *Convention Over Configuration*.

**Bootstrap** è [framework](#) per la creazione di siti e applicazioni per il *web*. Essa contiene modelli di progettazione basati su Html e Css. Necessarie per la tipografia, che per le varie componenti dell'interfaccia come moduli, bottoni e navigazione, e altri componenti dell'interfaccia.

**Drupal** è una piattaforma *software* di [Content Management System \(CMS\)](#), modulare, scritta in linguaggio Php e distribuita sotto licenza GNU GPL. Grazie alla sua architettura modulare, permette il riutilizzo sistematico del codice scritto e quindi un vantaggio, in termini di tempo e manutenzione.

### Database

Il *team* di sviluppo utilizza per la persistenza sia *database* relazionali, sia non-relazionali. Fra i *database* relazionali impiega per la maggiore le tecnologie quali MySql, Sql Server e PostgreSql. Per quanto riguarda i *database* non-relazionali utilizza soluzioni come MongoDb oppure OrientDb. A seconda del contesto vengono studiati i pro e contro, in modo da poter trarre tutti i vantaggi possibili.

## 1.2 Processi aziendali

Questa sezione illustra l'organizzazione interna dell'azienda, descritta per processi.



figura 1.4: Diagramma della metodologia di sviluppo *agile*. Immagine tratta da [GlobalTeckz](#).

### 1.2.1 Metodologia agile

L'azienda per la gestione dei progetti adotta la metodologia *agile*<sup>2</sup>.

Questa metodologia si basa su 4 principi:

- Gli individui e le interazioni più che i processi e gli strumenti;
- Il *software* funzionante più che la documentazione esaustiva;
- La collaborazione col cliente più che la negoziazione dei contratti;
- Rispondere al cambiamento più che seguire un piano.

In particolare tra le metodologie *agile* l'azienda segue nello specifico il modello *Scrum*, dove pianifica ciclicamente degli *sprint*(un periodo di 2-4 settimane).

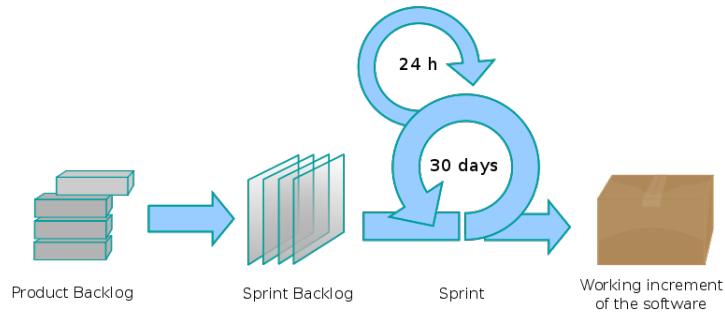
<sup>2</sup><http://www.agilemanifesto.org/>.

Gli elementi costruttivi principali in *Scrum* sono:

- **Product Backlog**, è un documento ad alto livello per l'intero progetto. Contiene una lista delle funzionalità desiderate, con priorità assegnate in base al valore di *business* e i *Backlog Item* che sono descrizioni di tutte le caratteristiche richieste;
- **Sprint Backlog**, è un documento che contiene informazioni su come il *team* sta procedendo nell'implementare le funzionalità, per lo *sprint* successivo. Le funzionalità vengono prelevate dalla pila del documento di *Product Backlog*, in una quantità fattibile per essere realizzata nello *sprint*;
- **Burn Down Chart**, è un grafico che rappresenta il lavoro nello *Sprint Backlog* che deve essere ancora completato. Aggiornato ogni giorno, dà una semplice visione di come procede lo *sprint*;
- **Incremento**, è la somma di tutti gli elementi del *Product Backlog*, completati durante uno *sprint* e tutti gli *sprint* precedenti.

Le principali pratiche che Nextep svolge sono:

- **Daily scrum**, detto anche *stand up*, è una riunione che tiene quotidianamente e serve per verificare lo stato di avanzamento durante uno *sprint*;
- **Scrum of Scrums**, è una riunione che tiene meno frequentemente del *Daily Scrum*. Consente ai *team* di discutere del loro lavoro, concentrandosi su aree di sovrapposizione e d'integrazione;
- **Sprint Planning Meeting**, è una riunione che l'azienda svolge all'inizio di ogni *sprint*. Vengono pianificati gli obiettivi con relative priorità e le tempistiche. In questa riunione, partecipano *Product Owner*, *Scrum Master*, intero *Scrum Team* e tutti i *manager* del caso interessati o dai rappresentanti della clientela;
- **Sprint Review**, al termine dello *sprint* l'azienda svolge questa riunione per ispezionare l'incremento e adattare, se necessario, il *Product Backlog*. In conformità a questo e dei cambiamenti al *Product Backlog*, fatti durante lo *sprint*, i partecipanti pianificano il prossimo incremento;
- **Sprint Retrospective Meeting**, si svolge dopo la *Sprint Review*, in questa occasione lo *Scrum Team* ispeziona se stesso e crea un piano di miglioramento, da attuare durante il successivo *sprint*.



**figura 1.5:** Diagramma del modello *Scrum*. Immagine tratta da [Wikipedia](#).

Questa metodologia consente e promuove un maggiore coinvolgimento del cliente rispetto a quelle tradizionali. Consente inoltre una maggiore flessibilità e reattività al cambio dei requisiti che possono avvenire in corso d'opera.

## 1.3 Strumenti a supporto dei processi

Questa sezione, illustra gli strumenti utilizzati a supporto dei processi e le tecnologie utilizzate per lo sviluppo.

### 1.3.1 Gestione di progetto

Per fornire assistenza ai clienti l'azienda utilizza Zendesk<sup>3</sup>, che è uno strumento che permette di centralizzare tutti i canali di comunicazione tra i quali *social network*, *email* e telefono. Inoltre fornisce una panoramica generale, della prestazione dell'assistenza e della soddisfazione del cliente.

Per la gestione di progetto invece utilizza lo strumento Jira<sup>4</sup>, che fornisce degli strumenti per la gestione di sviluppo agile con il modello *Scrum*, utilizzato nell'azienda. Nello specifico, Jira permette:

- Assegnazione dei *ticket* tra i membri del *team* di sviluppo;
- Segnalazione di *issue*;
- Integrazione con il sistema di *repository*;
- Pianificazione dello *sprint*.

Nextep non utilizza la *Kanban Board* integrata in Jira, perché questo strumento di processo non è compatibile con la metodologia *agile* in uso. Pertanto adopera una *Scrum Board* fisica presente in ufficio.

---

<sup>3</sup><https://www.zendesk.it/>.

<sup>4</sup><https://www.atlassian.com/software/jira>.

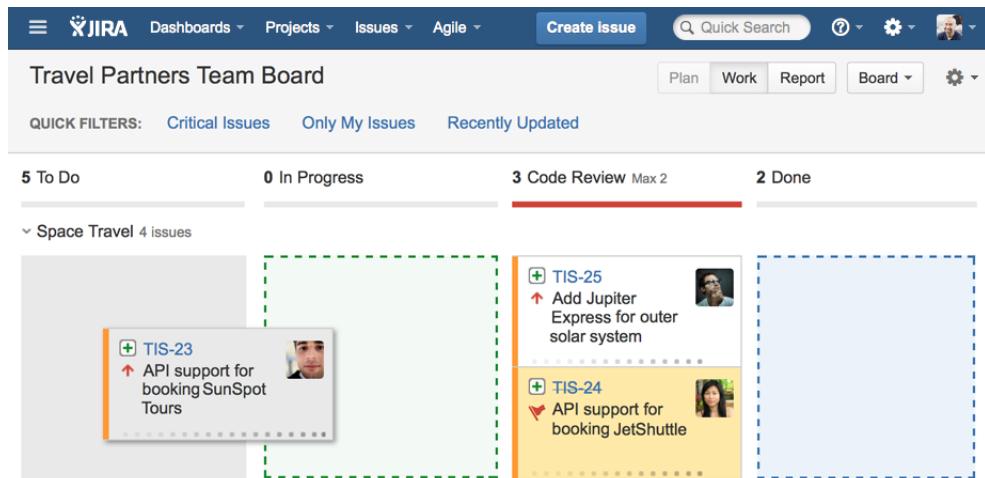


figura 1.6: Piattaforma di gestione di progetto Jira

### 1.3.2 Documentazione

Nextep per la produzione documentale utilizza una serie di strumenti di Google Docs, per una serie di fattori tra i quali:

- Compatibilità con tutti i sistemi operativi utilizzati;
- Permette di lavorare collaborativamente ai documenti;
- Disponibilità;
- Affidabilità.

### 1.3.3 Sistema di versionamento

Come sistema di versionamento, il *team* utilizza lo strumento di Git<sup>5</sup>.

I *repository* sono gestiti in un *server* locale, ed occasionalmente in server esterni.

### 1.3.4 Ambiente di sviluppo

A seconda del contesto, il *team* utilizza l'ambiente di sviluppo che meglio si adatta alle tecnologie in uso nel progetto. In particolar modo adopera i seguenti **Integrated Development Environment (IDE)**:

- Eclipse o IntelliJ: per lo sviluppo di applicazioni in Java e/o Scala;
- Smultron/PhpStorm: per lo sviluppo di applicazioni in Php;
- Sublime Text: per lo sviluppo in Html/Css, Javascript e Php.

---

<sup>5</sup><https://git-scm.com/>.

### 1.3.5 Sistemi operativi

L'azienda non impone nessun obbligo di utilizzo di un sistema operativo. I dipendenti utilizzano, a seconda delle preferenze, sistemi operativi quali Windows, Linux e MacOs. In particolar modo quest'ultimo, visto il recente investimento aziendale in nuovi terminali. Il fine è di incentivare i dipendenti, alla migrazione a questo sistema operativo.



**figura 1.7:** Classificazione degli strumenti utilizzati

## 1.4 Clientela tipo e propensione all'innovazione

La tipologia di clientela per cui l'azienda lavora, copre una gamma che va dal singolo privato alle organizzazioni di grandi dimensioni.

Essendo gli *e-commerce* il *core business* dell'azienda, Nextep lavora principalmente per aziende dell'ambito manifatturiero e rivendite.

L'azienda lavora per clienti/aziende italiane e molto spesso per clienti della zona, visto il riconoscimento guadagnato nel padovano.

Nextep riconosce questi valori guida da seguire:

- Attitudine all'innovazione per mantenere competitività;
- Condivisione del *know how* con le altre aziende in sede.

Questi principi rendono possibile un ambiente di *co-working*, dove le 3 aziende della sede sperimentano e innovano, grazie alle conoscenze messe in condivisione. Per alimentare questa dinamica, l'azienda promuove periodicamente la formazione dei dipendenti, in modo che essi possano essere motivati e partecipi in questo ambiente. Visto l'importanza delle dinamiche personali, l'azienda organizza, al bisogno, degli incontri di *team building* per avere sempre dei dipendenti in grado di lavorare in gruppo.



## Capitolo 2

# Il progetto nella strategia aziendale

*Questo capitolo fornisce una descrizione dettagliata del progetto di stage, e dei motivi che hanno spinto l'azienda a proporlo, oltre che i benefici derivati da tale progetto. Vengono inoltre discussi i vincoli tecnologici e metodologici imposti.*

### 2.1 L'azienda e gli stage

Questa sezione descrive il rapporto dell'azienda con gli stage e spiega quali sono i motivi, che portano l'azienda a farli. In particolare per il progetto svolto.

Data la forte evoluzione del settore *web*, l'azienda persegue l'obiettivo di esplorare i nuovi segmenti del campo, per mantenere un ruolo di rilievo nel mercato. Per questo motivo c'è l'esigenza di sperimentare continuamente nuove soluzioni, per valutare possibili impieghi e vantaggi. Si vengono quindi a creare le condizioni per un'opportunità di *stage*, dove l'azienda richiede allo stagista una soluzione, per le funzionalità richieste. Inoltre l'azienda sfrutta l'occasione degli *stage*, per valutare una possibile collaborazione e successivamente una assunzione in azienda, nel caso che il soggetto sia valutato positivamente. Nel precedente *stage*, visto l'interesse nel campo delle raccomandazioni, sono stati sviluppati dei prototipi che potessero soddisfare una raccomandazione utente, in ambito *e-commerce*. I sistemi di raccomandazione in ambito commerciale sono molto utili. Permettono un guadagno dal punto di vista economico, e dal punto di vista della *user experience*, perché consentono di inserire, pubblicità più pertinenti ai gusti dell'utente. In precedenza durante un recente *stage*, l'azienda aveva sviluppato il modulo di raccomandazione denominato DRE. Un sistema basato sul *Collaborative filtering*, che permette di fornire una raccomandazione in base alla somiglianza degli utenti.

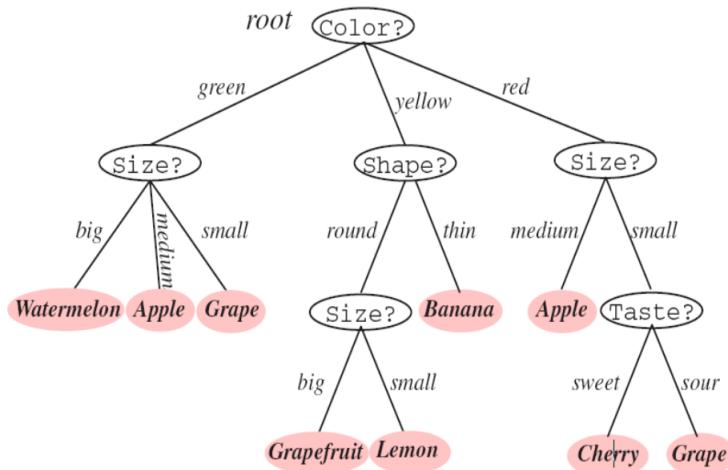
## 2.2 Il progetto

Questa sezione, descrive il progetto di *stage* e gli obiettivi che l'azienda ha fissato, in relazione alle sue specifiche esigenze di mercato.

### 2.2.1 Il progetto

Il progetto di stage consiste nella realizzazione di un sistema di raccomandazione, basato sulla classificazione del comportamento degli utenti all'interno del sito. Il sistema deve poter permettere l'inserimento delle azioni dell'utente, ogni volta che il suo comportamento ha prodotto un risultato. Una volta collezionata una considerevole mole di dati relativi ai comportamenti, il sistema deve poter fornire una raccomandazione all'utente sulla base del suo comportamento attuale.

Il comportamento di un utente è descritto tramite una serie di iterazioni e il risultato raggiunto. A sua volta una iterazione è descritta tramite l'azione e l'oggetto, su cui viene svolta. Questo comportamento viene poi elaborato dall'albero di decisione, generato dai dati collezionati precedentemente. La generazione dell'albero si basa sull'algoritmo di classificazione *ID3*. Esso permette di selezionare l'attributo, per discriminare il comportamento con il miglior guadagno informativo. Il sistema deve poter fornire una raccomandazione, anche probabilistica, nel caso di dati parziali sul suo comportamento.



**figura 2.1:** Esempio di un albero decisionale

### 2.2.2 Obiettivi

Prima dell'inizio dello stage, Nextep ha redatto un *Piano di Lavoro* contenente gli obiettivi da realizzare, nelle settimane di *stage*. Essi erano suddivisi in minimi e massimi. Successivamente in seguito ad una variazione delle priorità aziendali, sono stati effettuati dei cambiamenti. Inizialmente, ho effettuato il *porting* del modulo preesistente DRE, in modo che la persistenza fosse compatibile con OrientDb e con l'ultima versione di Play Framework. Visto il protrarsi delle attività di *porting*, l'azienda ha deciso di sospendere il lavoro, per poter sviluppare il nuovo modulo di raccomandazione.

Qui di seguito presento gli obiettivi dello stage:

- Obiettivi formativi
  - Formazione sui sistemi di raccomandazione e degli algoritmi di apprendimento;
  - Studio delle tecnologie utilizzate, Scala e OrientDb;
  - Studio degli strumenti utilizzati, in particolar modo dei **framework** e degli **IDE**.
- Obiettivi minimi
  - Utilizzo del database OrientDb.
- Obiettivi massimi
  - Implementazione di nuovi modelli di algoritmi di apprendimento, per migliorare l'individuazione dei gusti dell'utente;
  - Migliorare la fase di apprendimento del modello stesso;
  - Implementazioni di algoritmi di *cluster*, per il raggruppamento di elementi omogenei in un insieme di dati.

## 2.3 Vincoli

Questa sezione elenca i vincoli relativi al progetto di *stage*.

### 2.3.1 Vincoli temporali

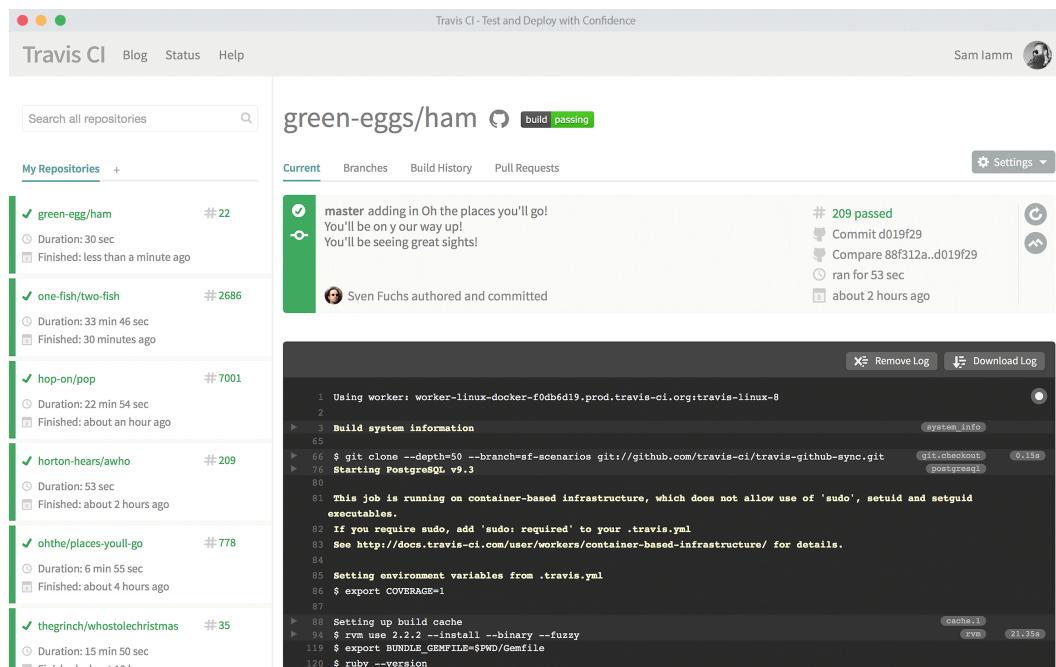
Per lo *stage* i vincoli temporali sono imposti direttamente dall'università. Essa stabilisce una durata minima di 300 ore e una durata massima di 320 ore. Per questo motivo, ho modulato il *Piano di Lavoro* in modo da distribuire le ore lavorative in 8 settimane. Con l'azienda ho concordato un impegno di 300 ore, divise in 37.5 ore settimanali. Le variazioni del piano sono state concordate con il *tutor* aziendale, in maniera da rispettare i vincoli temporali, imposti in seguito a ritardi dovuti al *porting* del modulo DRE.

### 2.3.2 Vincoli metodologici

Come strumento di versionamento, ho scelto Git, già utilizzato in precedenza e per il supporto che riceve da molti strumenti di sviluppo. Per la documentazione del codice ho scelto Scaladoc, fornito insieme al compilatore. Similmente a Javadoc permette di documentare direttamente nel codice sorgente. Per il *testing* nel progetto ho utilizzato TravisCI, esso fornisce un servizio di *continuous integration*, facilmente integrabile con i *repository* di Github<sup>1</sup>.

---

<sup>1</sup><https://github.com/>.



**figura 2.2:** Pannello di controllo travis

Per lo sviluppo del codice ho scelto come **IDE** IntelliJ, in quanto:

- Già precedentemente utilizzato;
- Ambiente di sviluppo compatibile con i maggiori linguaggi *JVM-based*;
- Integrazione con Git;
- Integrazione per lo sviluppo in Scala con Sbt.

Per la gestione progetti ho utilizzato il *build tool* Sbt. Nell'ambiente di sviluppo di Scala è la scelta predominante ed è compatibile con il **framework** utilizzato *Play Framework*. Le caratteristiche distintive di questo **framework** sono elencate di seguito:

- Definizione del file di configurazione in Scala;
- Compatibilità con Scaladoc per documentazione;
- Supporto per progetti misti Java/Scala;
- Supporto per i maggiori *tool* di *testing*;
- Supporto per la gestione delle librerie.

Per la realizzazione dei diagrammi *UML* ho utilizzato Astah Professional<sup>2</sup>, uno strumento già utilizzato in precedenza e compatibile con gli ultimi standard *UML 2.0*.

<sup>2</sup><http://astah.net/>.

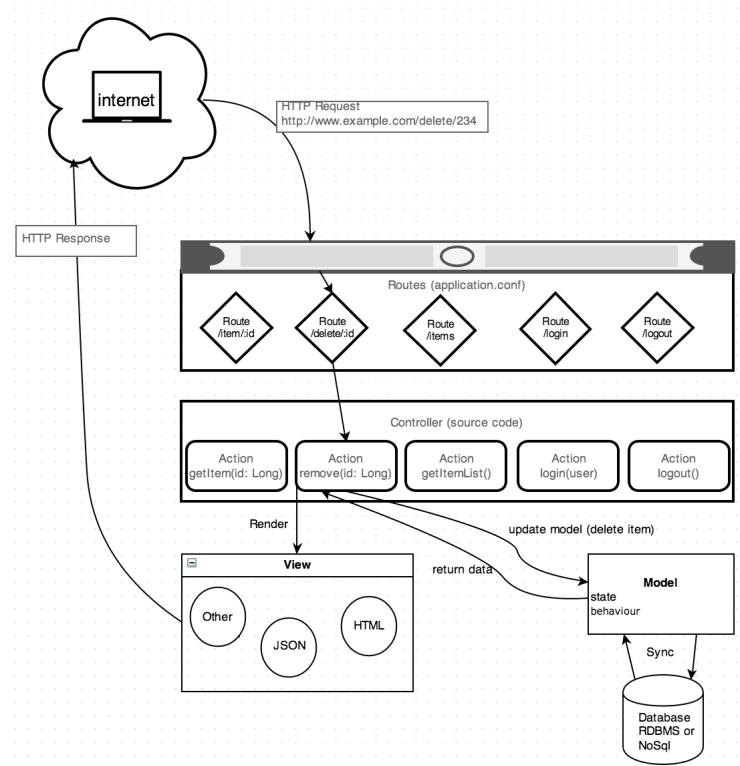
### 2.3.3 Vincoli tecnologici

#### Scala

Il principale linguaggio di programmazione adottato è Scala<sup>3</sup> (acronimo di "*Scalable Language*"). E' stato studiato per poter interoperare con *JRE*. E' possibile quindi scrivere applicazioni sia in Java e Scala, che eseguono nella stessa *JVM*. E' un linguaggio multi paradigma, che integra concetti provenienti dai linguaggi della programmazione ad oggetti e della programmazione funzionale. Scala è un linguaggio completamente *object oriented*, infatti ogni elemento del linguaggio è un oggetto, compresi i tipi primitivi e le funzioni. Inoltre è un linguaggio funzionale, include funzioni di prima classe e una libreria con strutture dati immutabili efficienti.

#### Play Framework

Play è un **framework open source**, scritto in Java e Scala. Permette l'implementazione di *web application* con il *pattern MVC*. Questo **framework** trae ispirazione ed è simile a Ruby on Rails e Django.



**figura 2.3:** Tipica architettura di un'applicazione che utilizza Play Framework. Immagine tratta da [Omer Haderi Blog](#).

<sup>3</sup><http://www.scala-lang.org/>.

Le peculiarità che lo contraddistinguono dagli altri [framework](#), sono le seguenti:

- Privo di stato: non viene mantenuta alcuna sessione sul server, relativa ai dati dell'utente corrente;
- Metodi statici: tutti i metodi dei *controller* che vengono invocati dal [framework](#) sono statici, o nel caso in cui si usi la versione Scala, sono funzioni di oggetti di Scala;
- Gestione asincrona dell'*input* e dell'*output*: grazie all'uso di Netty Server, Play può gestire le richieste lunghe in modo asincrono;
- Architettura modulare: come per Rails e Django, ci sono i moduli;
- Supporto nativo per Scala: non solo Play è fatto internamente in Scala, ma espone anche delle interfacce Scala. Le interfacce Java sono state messe appositamente in *package* diversi, affinché possano seguire le convenzioni di Java.

Al suo interno permette una implementazione delle *views*, specificando l'interfaccia con un linguaggio di *markup* (e.g. Html, Xml, [Javascript Object Notation \(JSON\)](#)) in combinazione con del codice Scala. Vengono fornite molte classi di *utility*, come ad esempio *Play Json Library* che permette di operare con dei dati in formato [JSON](#). Fornisce inoltre, un adattamento della libreria Specs2, che permette un supporto per il *testing* di una applicazione in Play. Una tipica applicazione in Play, funziona convogliando tutte le *HTTP Request* nella componente *Router* che invocherà a sua volta, il *Controller* pre-selezionato. Il *Controller* interagisce con il *Model* che a sua volta restituisce un risultato. In genere, questo risultato, verrà poi *renderizzato* con la *View* e fatto ritornare tramite un *HTTP Response*.

## OrientDB

Per la persistenza dei dati ho utilizzato OrientDb<sup>4</sup>. Un *DBMS* multi-modello con una licenza *open source*. Supporta i seguenti modelli di persistenza:

- *Graph*: modello a grafo;
- *Document*: modello a documenti come MongoDB;
- *Object*: modello ad oggetti, che consente di memorizzare direttamente i *POJO*.

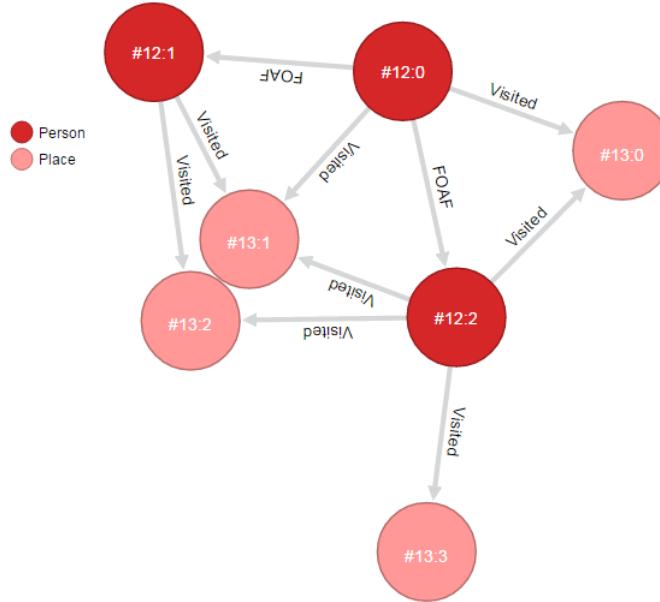
Inoltre supporta modelli di definizione dello schema quali *schema-less*, *schema-full* e *schema-mixed*. Per motivi prestazionali non permette le *join*. Le relazioni sono gestite come in un database a grafo, ovvero delle connessioni dirette tra i *record*. Questa caratteristica lo rende adatto nel contesto dei *Big Data*. La complessità computazionale per gestire una creazione di un *link* è pari a O(1). Tra le caratteristiche più importanti di OrientDb:

- Scalabile: ovvero è possibile aggiungere al sistema un altro *server*, per aumentarne le prestazioni;
- Transazionale: supporta transazioni completamente *ACID*, garantendo che tutte vengano processate in modo affidabile. Nel caso di *crash* viene ripristinato lo stato precedente (rollback);

---

<sup>4</sup><http://orientdb.com/>.

- Poliglotta: supporta *query* in un dialetto *SQL-like*, oppure delle *Application Program Interface (API)* *Representational State Transfer (REST)* che forniscono l'accesso ai dati.



**figura 2.4:** Esempio di modellazione in OrientDb. Immagine tratta da [Scalac Blog](#)

## 2.4 Obiettivi personali

Questa sezione elenca gli obiettivi personali, che hanno portato allo svolgimento dello *stage*.

Ho conosciuto l'azienda durante l'evento *Stage-IT*. Mi ha dato la possibilità di approfondire molte proposte di *stage*, da parte aziende provenienti da tutta la provincia di Padova e non. In seguito a tutte le proposte disponibili, ho accettato la proposta di Nextep per i seguenti motivi:

- Il progetto permetteva di lavorare con tecnologie innovative, che mi avrebbero fornito un valore professionale aggiunto;
- Il progetto proposto era interessante, perché toccava rami dell'intelligenza artificiale e dei sistemi di raccomandazione;
- L'utilizzo di Scala, in modo da poter padroneggiare un linguaggio con un paradigma funzionale, in forte ascesa di popolarità.



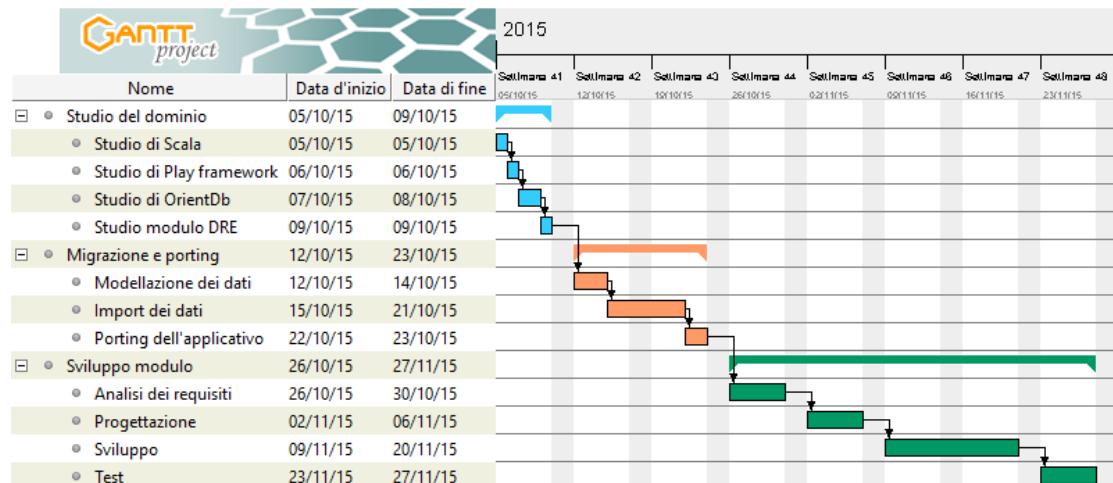
# Capitolo 3

## Il progetto di stage

*Questo capitolo tratta dettagliatamente delle attività di pianificazione, studio, analisi, progettazione e sviluppo, svolte durante lo stage.*

### 3.1 Organizzazione

Per conseguire tutti gli obiettivi fissati, ho previsto un ammontare di 300 ore. Insieme al *tutor* aziendale ho concordato la data di inizio stage il 5/10/2015 e la data di fine stage il 27/11/2015. Quindi un carico lavorativo di 37.5 ore settimanali, ovvero 7.5 ore giornaliere.



**figura 3.1:** Suddivisione delle attività

### 3.1.1 Pianificazione

In sede di pianificazione, prima dell'inizio dello *stage*, ho programmato 3 macro attività sequenzialmente. Ho assegnato, nel primo periodo, una settimana per l'attività di studio del dominio, in quanto questa attività è propedeutica alle successive attività. Nel periodo successivo di 2 settimane, ho pianificato l'attività di *porting* dell'applicativo DRE, su richiesta del *tutor* per motivi di strategia aziendale. Nell'ultimo periodo, ho programmato l'attività di sviluppo del modulo denominato Tres. Ho privilegiato quest'ultima attività, assegnando un totale di 5 settimane lavorative. Qui di seguito illustro le sotto-attività programmate per ogni settimana.

- **Prima settimana:**
  - Studio del *database* OrientDB;
  - Studio del linguaggio di programmazione Scala;
  - Studio del *framework* Play;
  - Studio del progetto preesistente;
- **Seconda e terza settimana:** migrazione dalla tecnologia MongoDB a OrientDB e *porting* all'ultima versione di Play;
- **Quarta settimana:** Analisi dei requisiti del nuovo modulo;
- **Quinta settimana:** progettazione architetturale;
- **Sesta e settima settimana:** implementazione modulo;
- **Ottava settimana:** test.

La figura 3.1 presenta il diagramma di *gantt* utilizzato per la pianificazione.

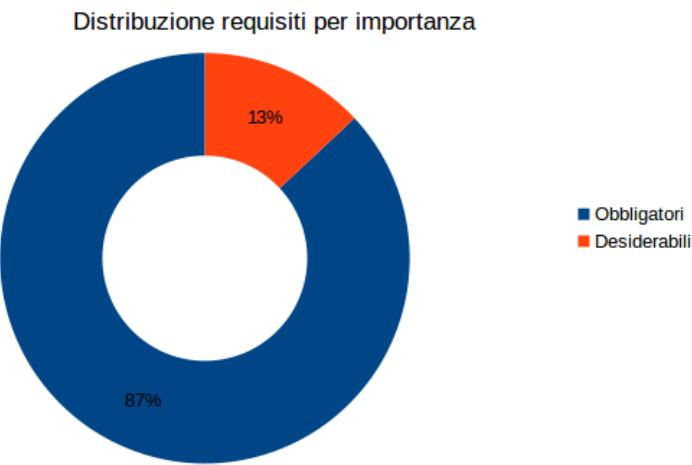
## 3.2 Analisi dei requisiti

### 3.2.1 Requisiti

Per l'attività di analisi dei requisiti, ho svolto molteplici incontri insieme al *tutor* aziendale, per delineare quali funzionalità dovesse offrire il sistema. Il *tutor* mi ha fornito molti esempi ed illustrazioni, per capire al meglio quali requisiti il sistema dovesse soddisfare. Tuttavia questa attività è stata molto ostica, perché sia il *tutor* e sia il *CEO* esprimevano richieste diverse e spesso molto discordanti. Visto che il proponente non aveva una visione chiara sul sistema da implementare, ho cercato di assumere un atteggiamento più propositivo, per aiutarlo a comprendere i suoi bisogni e portare a termine l'analisi dei requisiti nei tempi prefissati. Una volta definito le funzionalità, ho trascritto i requisiti in un file testuale, in un formato organizzato tabellare. Visto le dimensioni ridotte del progetto ho prediletto il tracciamento dei requisiti con un file di testo anziché utilizzare un sistema automatizzato. Il file è stato poi sottoposto a versionamento all'interno del *repository*, per permettere operazioni di consultazione e modifica all'evolversi dei requisiti. La notazione scelta per suddividere i requisiti è la seguente:

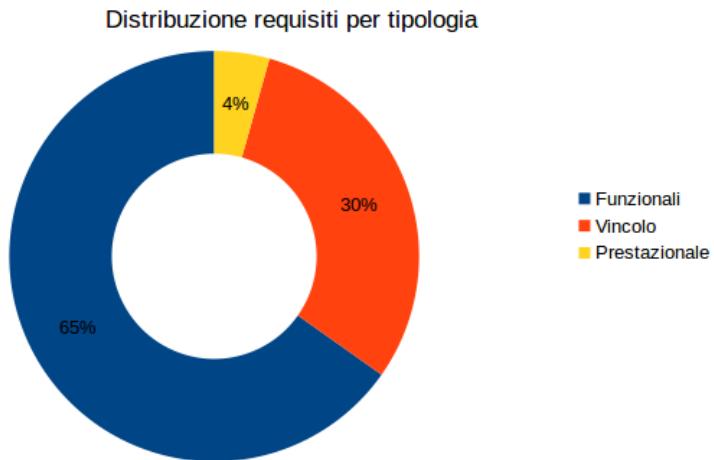
R[Importanza][Tipologia][Codice]

- Importanza può assumere i seguenti valori:
  - **OBB:** requisito obbligatorio. Il soddisfacimento è necessario per il raggiungimento degli obiettivi dello *stage*;
  - **DES:** requisito desiderabile. L'implementazione non è fondamentale, ma dà valore aggiunto al prodotto.
- Tipologia può assumere i seguenti valori:
  - **F:** requisiti funzionali. Specifica una funzionalità che il *software* deve avere;
  - **V:** requisiti di vincolo. Specifica il vincolo che il *software* deve avere;
  - **P:** requisiti di prestazione. Specifica un vincolo di *performance* che il *software* deve fornire.
- Codice è un *id* numerico univoco.



**figura 3.2:** Distribuzione requisiti in percentuale per tipologia

Durante l'analisi dei requisiti, prima di trasporre i requisiti secondo la notazione precedentemente menzionata, ho rielaborato i requisiti. In modo che essi non contenessero ambiguità verbali, se presenti ho corredato al documento di analisi un glossario, dove specificare i termini ed evitare così interpretazioni errate del requisito. Inoltre ho suddiviso ogni requisito in più requisiti, fino a che ogni singolo requisito risultasse indivisibile. Invece i requisiti che risultavano ridondanti tra loro, li ho uniti affinché ogni requisito risultasse indipendente. Ho verificato che ogni requisito fosse verificabile secondo una metrica in modo da certificarne il soddisfacimento. Al termine dell'analisi, i requisiti individuati ammontano a 23. Come raffigurato nell'immagine 3.2, i requisiti sono 15 di natura funzionale, 1 prestazionale e i restanti 7 requisiti di vincolo. I requisiti sono stati classificati per priorità, determinando così 20 requisiti obbligatori e 3 desiderabili. I requisiti desiderabili riguardano nella maggior parte gli obiettivi di massima fissati dal piano di lavoro.



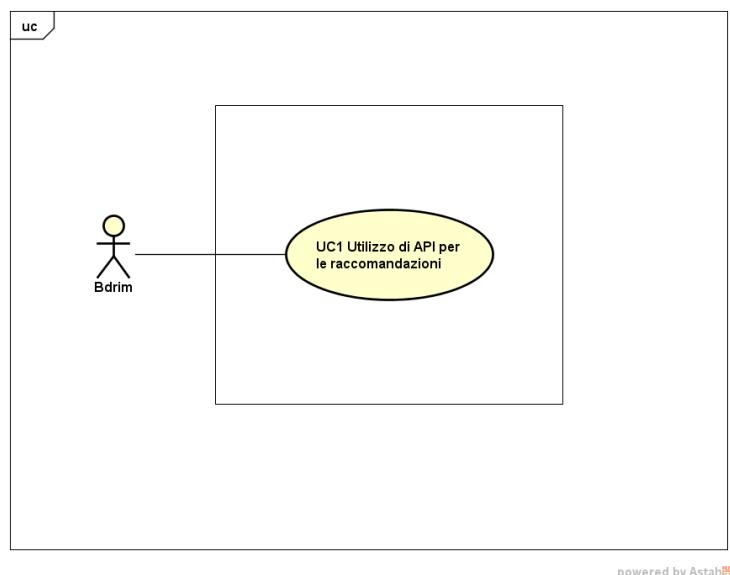
**figura 3.3:** Distribuzione requisiti in percentuale per importanza

### 3.2.2 Casi d'uso

Durante l'analisi dei requisiti ho realizzato dei diagrammi [Unified Modeling Language \(UML\)](#), utilizzando il *software* Astah. Tres è concepito per operare con Bdrim, una piattaforma di raccolta dati sviluppato in collaborazione con Allos. Quindi l'unico attore coinvolto nell'utilizzo di Tres è Bdrim, che espone un caso d'uso principale. In questa sezione riporto i casi d'uso principali, che descrivono maggiormente il sistema nel suo complesso. Per ogni caso d'uso ho specificato:

- Gli attori coinvolti;
- Pre e post condizione;
- Scopo e descrizione;
- Flusso principale degli eventi.

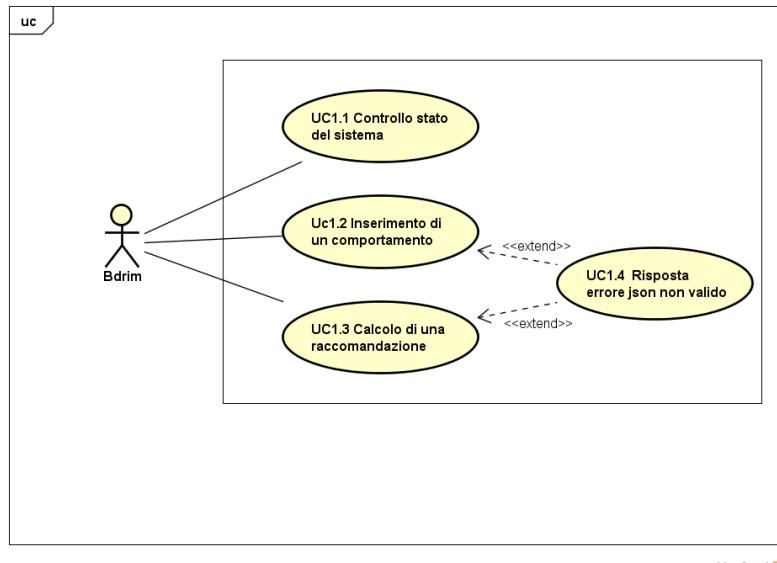
Inoltre, se previsto, ho specificato estensione e lo scenario alternativo.

**UC0**

powered by Astah

**figura 3.4:** UC0: Tres

- **Attori:** Bdrim
- **Scopo e descrizione:** Questo caso d'uso descrive le funzionalità messe a disposizione dal sistema. Principalmente Bdrim può usufruire delle **API** per le raccomandazioni fornite da Tres.
- **Pre-condizione:** Il *server* è stato avviato.
- **Flusso principale degli eventi:** Bdrim utilizza le **API** per ottenere una raccomandazione.
- **Post-condizione:** Il sistema risponde alle richieste HTTP effettuate da Bdrim.

**UC1**

powered by Astah

**figura 3.5:** UC1: Tres

- **Attori:** Bdrim
- **Scopo e descrizione:** Questo caso d'uso illustra le funzionalità offerte dal sistema a Bdrim. Bdrim ha la possibilità di effettuare delle richieste REST al sistema;
- **Pre-condizione:** Il *server* è stato avviato e Tres è stato attivato.
- **Flusso principale degli eventi:**
  - 1 Bdrim richiede lo stato del sistema [UC1.1];
  - 2 Nel caso il sistema sia pronto Bdrim richiede la raccomandazione[UC1.3];
  - 3 Bdrim può inserire un comportamento[UC1.2].
- **Scenario alternativo:** Bdrim effettua delle richieste *HTTP*, inviando un oggetto **JSON** malformato, oppure invalido. Il sistema ritorna una risposta *HTTP* che segnala l'errore in formato **JSON**.
- **Estensione:** Bdrim riceve una risposta *HTTP* contenente un **JSON**, che segnala l'errore di dati non validi [UC1.4];
- **Post-condizione:** Il sistema risponde alle richieste *HTTP*, ritornando una risposta *HTTP*. I dati ritornando in formato **JSON**.

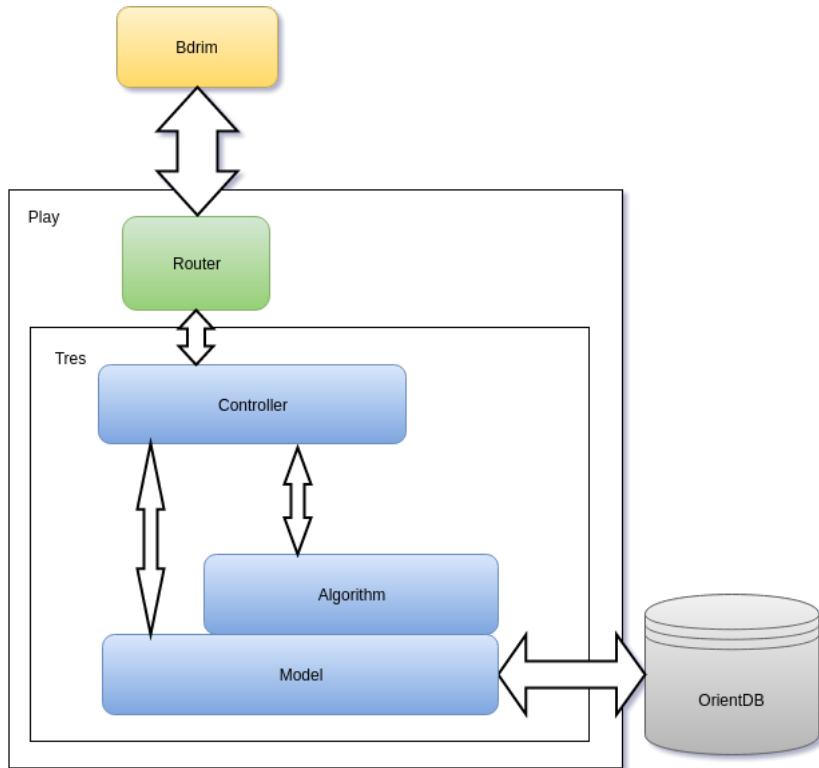
### 3.3 Progettazione architetturale

Questa sezione descrive l'attività di progettazione. Illustra l'architettura generale del sistema, le relazioni interne, la struttura del *database* e infine i *design pattern* utilizzati.

#### 3.3.1 Visione ad alto livello

Il contesto di utilizzo dell'applicativo è la piattaforma di Bdrim. Questa piattaforma permette di gestire nuovi moduli, per introdurre nuove funzionalità. Ogni modulo deve essere assolutamente isolato, per poter essere facilmente modificabile e interscambiabile con altri moduli, che migliorino le sue funzionalità. Quindi la visione generale del sistema, si riconduce ad un modello *client-server* dove i ruoli sono:

- **Server:** rappresentato da Tres, elabora le richieste *HTTP* ricevute da Bdrim.
- **Client:** rappresentato unicamente da Bdrim.

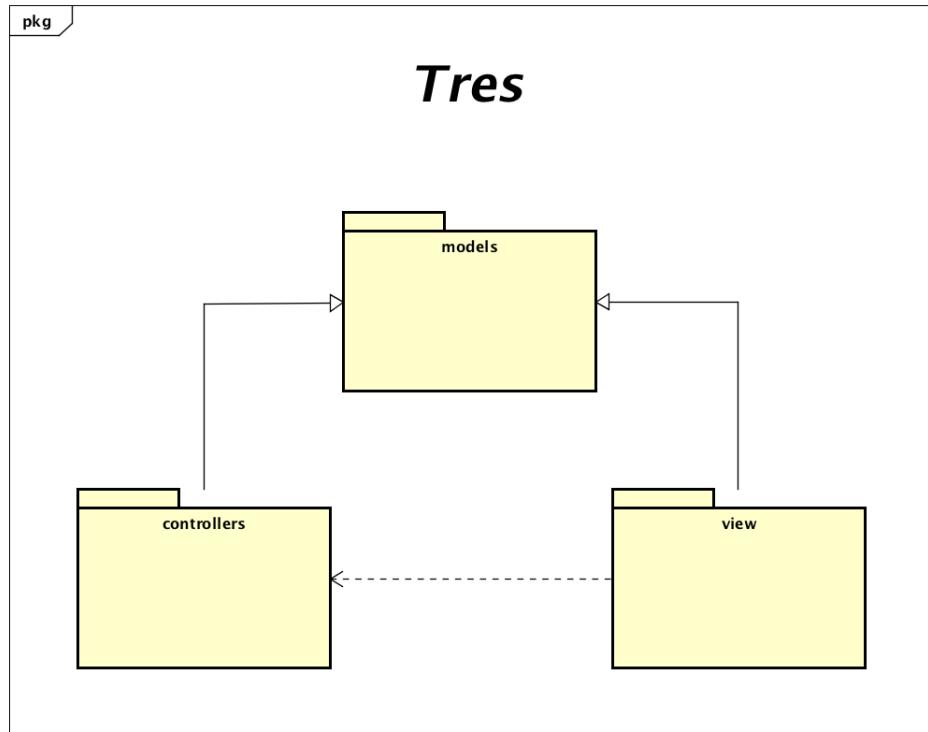


**figura 3.6:** Visione generale dell'architettura

Le due componenti sono indipendenti tra loro e comunicano utilizzando una architettura [REST](#).

### 3.3.2 Architettura

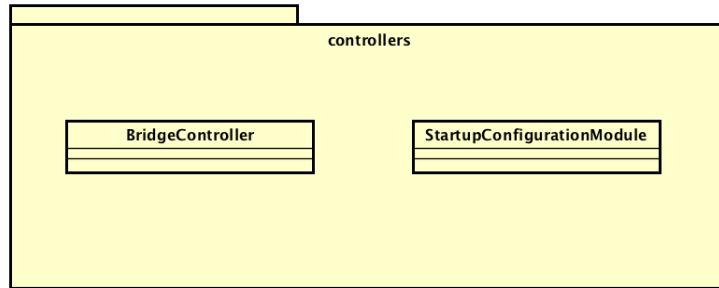
Per la realizzazione di Tres, ho seguito un *design* architetturale [MVC](#). Quindi i *package* principali sono: *model*, *view* e *controller*. Attualmente il *package* *views* non viene utilizzato. Esso tornerà utile in eventuali sviluppi successivi, per la creazione di una interfaccia di amministrazione per Bdrim o altre eventuali funzionalità. Qui di seguito illustro la visione generale e successivamente le componenti del sistema.



**figura 3.7:** Diagramma del *package tres*

### controllers

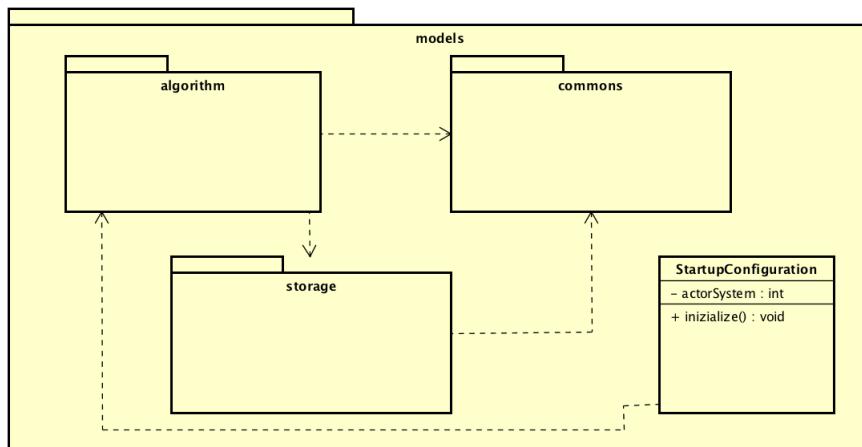
Il *package controllers*, contiene le componenti responsabili della logica di controllo e nello specifico contiene due classi. *BridgeController* è la componente che gestisce le richieste *HTTP*, instradate dalla componente *router* di Play. Il *package* interagisce con *models*, utilizzando i servizi messi a disposizione e le rappresentazioni dei dati.



**figura 3.8:** Diagramma del *package controllers*

### models

Il *package models*, contiene tutta la logica di *business* del sistema e di accesso ai dati. È responsabile di fornire una separazione, dalla rappresentazione interna dei dati del database. Fornisce tutte le interfacce necessarie al *controller* per fornire i servizi.



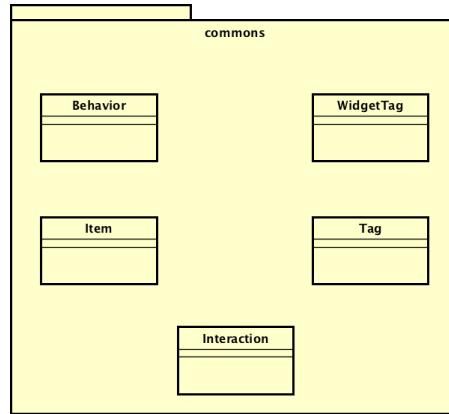
**figura 3.9:** Diagramma del *package models*

### Classi contenute

- **StartupConfiguration:** Questa classe è responsabile di schedulare la generazione dell'albero, per le raccomandazioni. La generazione dell'albero avviene ogni 24 ore, solamente se sono disponibili almeno 100 comportamenti memorizzati.

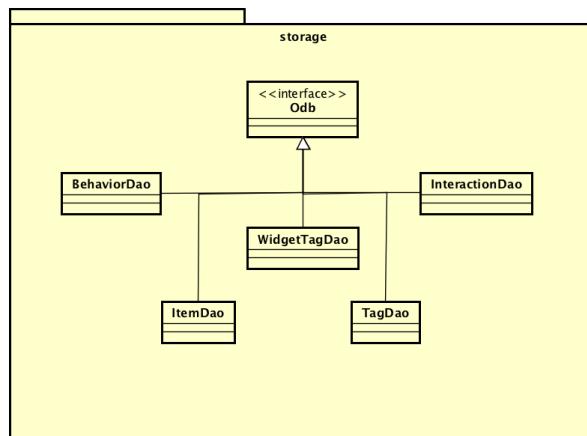
### Package contenuti

**commons** Il *package commons* contiene le componenti per la rappresentazione interna dei dati. All'interno sono presenti dei convertitori impliciti, che permettono la validazione e la conversione in formato **JSON**. Funzionalità utili per le componenti nel *controllers*.



**figura 3.10:** Diagramma del *package commons*

**storage** Il *package storage* è responsabile dell'accesso ai dati. Fornisce una interfaccia per le funzionalità *CRUD*. Il *package* dipende dalle componenti in *commons*, e per ognuna implementa l'interfaccia per l'interazione con OrientDb. Le componenti che gestiscono la persistenza in OrientDb utilizzano le **API Blueprints**<sup>1</sup>, perchè OrientDb aderisce a questo standard di default. TinkerPop Blueprints fornisce delle **API** per manipolare *database* a grafo ed è incluso nello *stack* dei progetti di TinkerPop<sup>2</sup>

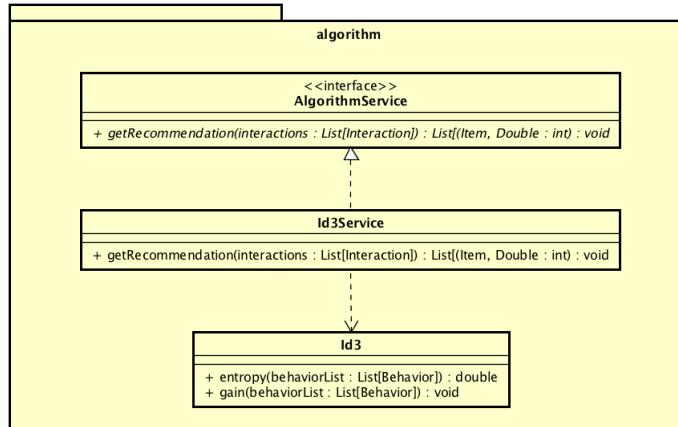


**figura 3.11:** Diagramma del *package storage*

<sup>1</sup><https://github.com/tinkerpop/blueprints>.

<sup>2</sup><http://tinkerpop.incubator.apache.org/>.

**algorithm** Il package *algorithm*, contiene le componenti *software* che realizzano l’algoritmica, basata essenzialmente su *ID3* ed è fortemente dipendente dalle componenti presenti in *commons*. Fornisce la funzionalità per la costruzione dell’albero decisionale, sulla base dei *behavior* in *input*. Offre la funzionalità per ricevere in *output* la raccomandazione, sulla base di un *behavior* in *input*.



**figura 3.12:** Diagramma del package *algorithm*

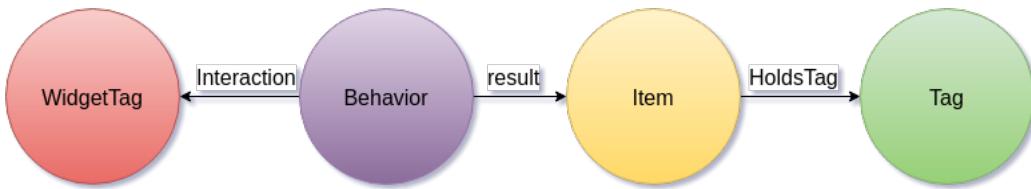
### 3.3.3 Database

Rispetto alle esperienze precedenti, come nel mio corso di laurea dove ho utilizzato tecnologie più tradizionali come i *database* relazionali. Questa occasione mi ha permesso di utilizzare un database *NoSQL*, come OrientDb. Questo *database* supporta diversi modelli di persistenza. Per questo progetto ho utilizzato il modello a grafo, che fa riferimento direttamente alla *teoria dei grafi*. In particolare la variante di grafo che OrientDb utilizza è *property graph*. Per la modellazione ho tenuto presente i seguenti concetti:

- I vertici sono delle entità, mentre gli spigoli rappresentano le relazioni;
- Una entità contiene un chiave identificativa e delle proprietà;
- Una relazione può contenere delle proprietà, è espressa tramite una *label* e collega due entità. La direzione della relazione è importante per la semantica della relazione.

In OrientDb è possibile aggiungere attributi agli spigoli, ma è fortemente sconsigliato per motivi prestazionali<sup>3</sup>. Per la persistenza dei dati ho scelto una definizione dei dati di tipo *schema-full*, per evitare un controllo dei dati a *run-time* nelle componenti del *package storage*.

<sup>3</sup><http://orientdb.com/docs/last/Performance-Tuning-Graph.html>



**figura 3.13:** Modello a grafo del database

La fase di modellazione a grafo è simile alla tecnica di modellazione relazionale. All'inizio ho individuato le entità del dominio e le modalità con cui si relazionano. Successivamente, invece di trasformare la rappresentazione in tabelle, ho arricchito lo schema andando ad individuare gli attributi interni delle entità e delle relazioni. Le entità principali che ho individuato sono:

- **Behavior:** questa entità rappresenta il comportamento di un utente all'interno di un sito web.
- **Item:** questa entità rappresenta in genere un oggetto in vendita nel sito web.
- **WidgetTag:** questa entità rappresenta un elemento generico all'interno di un sito web. Serve per individuare l'oggetto su cui l'utente svolge una azione.
- **Tag:** questa entità è una etichetta generica, che serve a descrivere un *Item*.

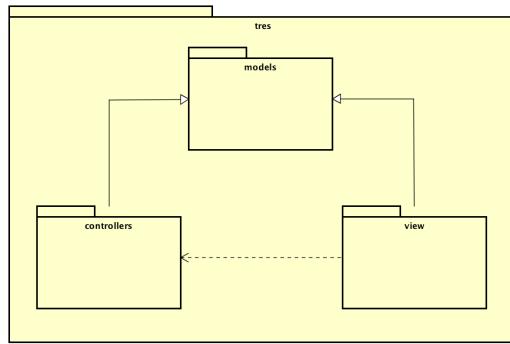
Le relazioni individuate sono:

- **Interaction:** Questa relazione parte dalla entità *Behavior* e arriva a *WidgetTag*. Esprime principalmente che un *Behavior* ha interagito sul *WidgetTag* collegato, con una azione specifica;
- **Result:** Questa relazione parte da entità *Behavior* e arriva a *Item*. Esprime il risultato di un *Behavior*;
- **HoldsTag:** Questa relazione parte da *Item* e arriva a *Tag*. Esprime i *Tag* che descrivono l'entità *Item*.

### 3.3.4 Design pattern

Qui di seguito elenco i *design pattern* utilizzati per la realizzazione di Tres.

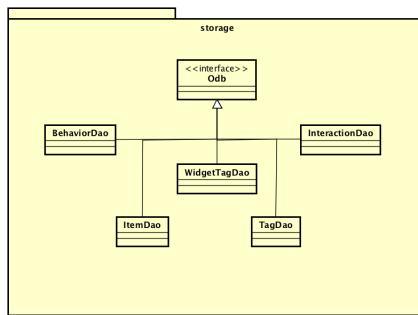
#### MVC



**figura 3.14:** Contesto di utilizzo del *design pattern* **MVC**

- **Scopo dell'utilizzo:** questo *design pattern* viene utilizzato per separare i compiti delle diverse componenti *software* dell'applicazione. separa *business logic*, *application logic* e viste, rappresentate rispettivamente da *models*, *controller* e *views*;
- **Contesto di utilizzo:** questo *pattern* viene implementato nativamente dal framework Play. In particolare implementa una tipologia di **MVC pull-model**, che permette un salto tecnologico tra la *view* e il *controller*. Questo *pattern* architettonico è importante, perché permette ai programmatore di lavorare sulle componenti di logica applicativa e ai grafici sulle componenti di interfaccia grafica.

#### DAO

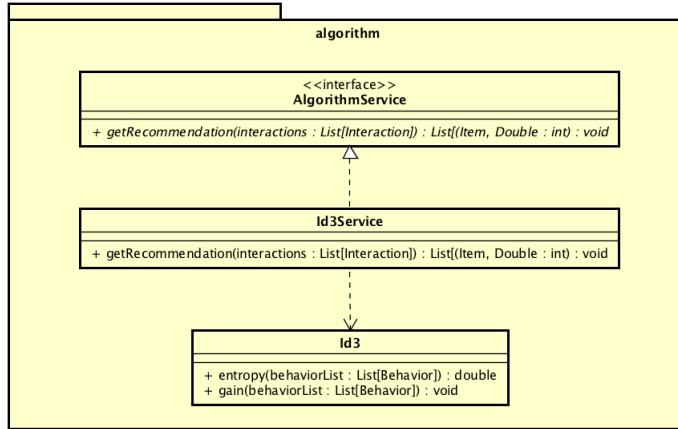


**figura 3.15:** Contesto di utilizzo del *design pattern* **DAO**

- **Scopo dell'utilizzo:** questo *design pattern* è utilizzato per disaccoppiare la logica di *business* dalla logica di persistenza. Rende così le componenti indipendenti dal sistema di *database*;

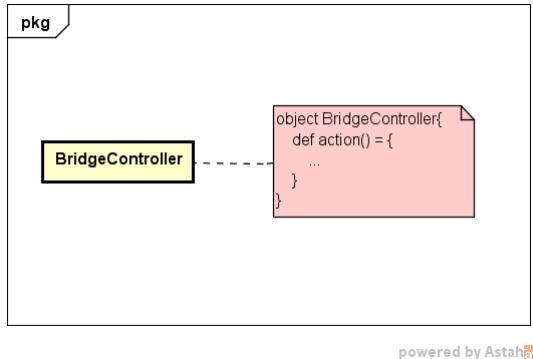
- **Contesto di utilizzo:** viene utilizzato dalle componenti presenti nel *package storage*, per interagire col *database* OrientDb. L'azienda utilizza diverse tecnologie di persistenza dei dati, quindi per essere facilmente interscambiabile in scenari futuri, è importante l'utilizzo di questo pattern che permette una facile transizione da una tecnologia all'altra.

### Strategy



**figura 3.16:** Contesto di utilizzo del *design pattern strategy*

- **Scopo dell'utilizzo:** il *design pattern strategy* definisce una famiglia di algoritmi, incapsulati e resi intercambiabili. *Strategy* permette agli algoritmi di variare indipendentemente dai *client* che ne fanno uso;
- **Contesto di utilizzo:** viene utilizzato dalle componenti presenti nel *package algorithm*. E' presente una interfaccia per l'algoritmo. Essa viene implementata da *Id3Service*, che realizza l'algoritmo *ID3*. Ho ritenuto importante applicare questo *pattern* nell'ottica di rendere l'algoritmo facilmente interscambiabile, con altri algoritmi di raccomandazione, come ad esempio degli algoritmi basati sulla teoria dei giochi.

**Singleton**

powered by Astah

**figura 3.17:** Contesto di utilizzo del *design pattern singleton*

- **Scopo dell'utilizzo:** il *design pattern singleton* assicura che una certa classe abbia una sola istanza e fornisce un punto d'acceso globale a tale istanza;
- **Contesto di utilizzo:** viene utilizzato nelle componenti del *package controller*. Viene implementato nativamente dal linguaggio Scala dichiarando la classe come *object*. L'utilizzo di questo pattern è fondamentale in questo contesto perché permette il controllo completo delle modalità e tempistiche di accesso del *client*.

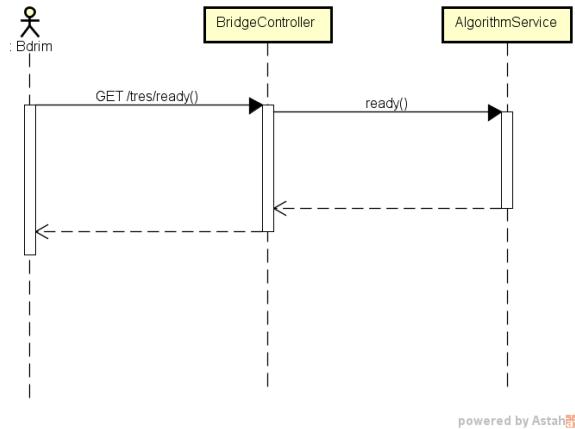
### 3.4 Progettazione di dettaglio

La progettazione di dettaglio, con la metodologia *agile*, avviene di pari passo durante la codifica, il che è rischioso perché introduce molti errori di codifica. E' necessario avere una forte conoscenza del dominio tecnico, per poter procedere e non cadere nel modello *code and fix*. Rispetto alle metodologie tradizionali, il focus si sposta nella codifica anziché nella progettazione. Quindi le ore risparmiate della progettazione di dettaglio, sono state reimpiegate nel correggere eventuali errori di codifica. Riporto qui di seguito i diagrammi di sequenza *UML* più significativi:

#### 3.4.1 Controllo stato

In fase di *startup* il sistema non possiede nessun dato, quindi non è in grado di fornire raccomandazioni. Mediante questa chiamata dell'interfaccia **REST**, è possibile verificare lo stato operativo del modulo:

Tipo	Chiamata
GET	/tres/ready



**figura 3.18:** Diagramma di sequenza: controllo stato.

La richiesta *HTTP* giunge fino al metodo *ready* della classe *BridgeController*, mediante il *router*. Questo metodo controllerà la presenza dell'albero tramite *AlgorithmService*. Al termine ritorna in *output* una risposta *HTTP*, contenente un oggetto **JSON**.

### 3.4.2 Inserimento behavior

Per istruire l'algoritmo di selezione, è necessario fornire dei dati per l'apprendimento. Ho implementato una procedura di inserimento dei comportamenti, mediante questa chiamata dell'interfaccia REST:

Tipo	Chiamata
POST	/tres/behavior

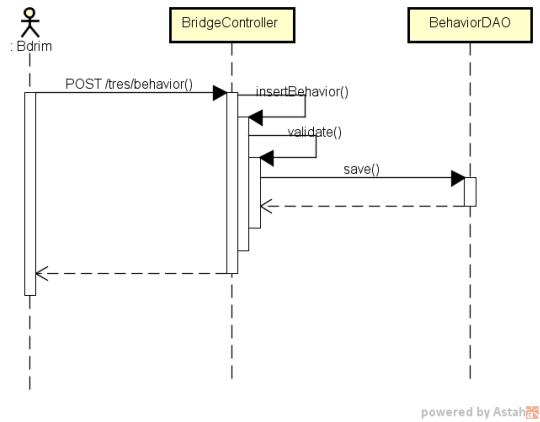


figura 3.19: Diagramma di sequenza: inserimento comportamento.

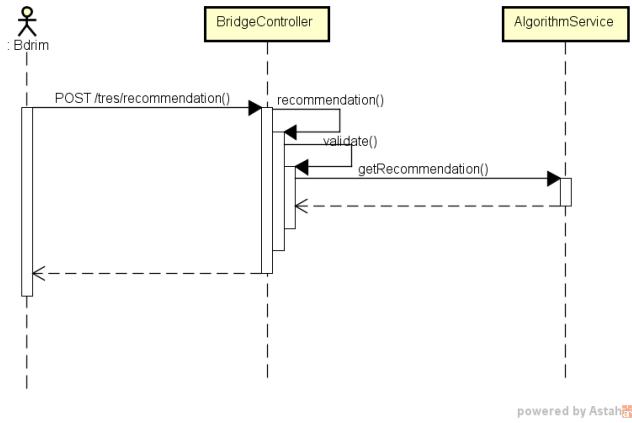
Da Bdrim arriva una richiesta *HTTP POST*, che comprende un **JSON** contenente un *Behavior*. Il *router* di Play accoglie la richiesta, che la reindirizza al metodo della classe *BridgeController insertBehavior*. Il flusso continua con la validazione del **JSON**. Successivamente il metodo inserisce il comportamento, invocando il metodo *save* della classe *BehaviorDAO*.

### 3.4.3 Raccomandazione

Una volta raccolti almeno 100 *Behavior* e che l'algoritmo ha generato l'albero, il sistema è in grado di fornire una raccomandazione. Ho implementato la procedura di raccomandazione, mediante questa chiamata dell'interfaccia REST:

Tipo	Chiamata
POST	/tres/recommendation

Da Bdrim arriva una richiesta *HTTP POST*, che comprende un **JSON** contenente le interazioni dell'utente. Il *router* convoglia la richiesta al metodo *recommendation*, di *BridgeController*. Il flusso procede con la validazione del **JSON** ricevuto, che estrapola una lista di interazioni dell'utente. Questa lista viene elaborata dall'albero, tramite la chiamata al metodo *getRecommendation* di *AlgorithmService*, che ritornerà le probabilità delle raccomandazioni.



**figura 3.20:** Diagramma di sequenza: raccomandazione.

### 3.5 Verifica e Validazione

Questa sezione descrive l'attività di verifica e validazione svolta durante la realizzazione di Tres. Sono elencati gli strumenti utilizzati, le metriche utilizzate e infine un resoconto finale dei test. L'attività di verifica, mi ha permesso di individuare prontamente errori di codifica e ridurre i tempi di sviluppo. Ho cercato di automatizzare il più possibile questa attività, per focalizzarmi nello sviluppo. In primis, ho effettuato dei test di unità per le componenti più complesse. Successivamente ho verificato la loro integrazione. Per motivi di tempo, non ho potuto realizzare tutti i test di unità e di integrazione. Mi sono, quindi, focalizzato sulle componenti principali.

#### Strumenti

Durante lo sviluppo, ho utilizzato il più possibile degli automatismi. Risparmiando più tempo possibile durante l'analisi statica e dinamica . Ho adottato i seguenti strumenti:

- **IntelliJ IDEA:** questo IDE fornisce funzionalità per identificare errori di programmazione. Come ad esempio sintassi errata, incompatibilità tra tipi e variabili non definite. Fornisce la percentuale di documentazione dell'intero progetto.
- **TravisCI:** è uno strumento che fornisce un automatismo, per la compilazione e l'esecuzione dei *test* del progetto. Se la compilazione o almeno un test fallisce, invia una notifica via *email*, con la possibilità di consultare il *log*.
- **Codecov:** questo strumento fornisce la percentuale delle linee di codice coperte durante i *test*.
- **Specs2:** è una libreria di *testing*, fornita direttamente da Play Framework. Mi ha permesso di testare l'intera applicazione, anche con le chiamate HTTP.

### Metriche adottate

Per la realizzazione di Tres ho adottato le seguenti metriche:

- **Attributi per classe:** un numero troppo elevato di attributi, potrebbe indicare la necessità di suddividere la classe in una gerarchia. Ho fissato un *range* ottimale con i seguenti valori: un minimo 1 fino ad un massimo di 8;
- **Complessità ciclomatica:** è un indice utilizzato per misurare la complessità di: funzioni, metodi, moduli e classi di un programma. Misura direttamente il numero di cammini linearmente indipendenti, attraverso il grafo di controllo di flusso. I nodi del grafo corrispondono a gruppi indivisibili di istruzioni, mentre gli archi connettono due nodi, se il secondo gruppo di istruzioni può essere eseguito immediatamente dopo il primo gruppo. Per questa metrica ho fissato un *range* ottimale, da un minimo di 1 fino ad un massimo di 10;
- **Copertura dei commenti:** è indice della percentuale di classi e metodi corredati di commenti. Ho fissato un range ottimale da 80% a 100%;
- **Copertura dei test:** è una metrica utilizzata per misurare l'efficacia del collaudo. Si utilizza un indice, che tiene traccia di quante volte è stata eseguita ogni istruzione, durante il *testing*. Le istruzioni eseguite almeno una volta sono dette "coperte". L'obiettivo è coprire il maggior numero possibile di istruzioni. In modo da avere una minore quantità di errore. Per questa metrica ho fissato un valore ottimale minimo di 70%.

#### 3.5.1 Resoconto risultati

##### Metriche misurate

Qui di seguito sono riportate le metriche software misurate:

- **Complessità ciclomatica:** per il calcolo della complessità ciclomatica ho utilizzato Sbt, al suo interno è disponibile il *tool* styleCheck. Questo strumento segnala, con un *warning*, se la complessità supera il valore 10. Il *tool* ha restituito esito positivo e non ha segnalato alcun *warning*;
- **Attributi per classe:** ho rilevato per questa metrica come valore medio 2, mentre come valore massimo ho rilevato 4;
- **Copertura commenti:** il valore di copertura è pari al 100%;
- **Copertura dei test:** il valore di copertura dei *test* è del 73%.

### Test di unità e di integrazione

I *test* di unità vengono svolti incrementalmente, durante la codifica di ogni componente. I primi *test* che ho effettuato, riguardano principalmente le componenti base del *models* e quelle responsabili delle iterazioni con il *database*. Successivamente sono passato al *testing* della parte di algoritmica, presente nel *package algorithm*. Per i *test* di unità, ho evitato di utilizzare *mock* e *stub* per le componenti base presenti in *commons*. In quanto ritenevo quelle componenti, molto elementari e che non avrebbero introdotto errori o comportamenti non attesi. Per l'integrazione delle componenti, ho seguito un approccio *bottom-up*. Questo mi ha aiutato a ridurre i tempi di *testing*, evitando sprechi di risorse utilizzando *mock* e *stub*. Ho focalizzato i *test* di integrazione nella parte di algoritmica e in quelle di interazione col *database*. Perché l'incertezza del comportamento di quelle componenti era altissima. Al netto dell'attività di *testing*, i *test* di unità eseguiti ammontano ad un totale di 10, di cui 10 hanno esito positivo. Per quanto riguarda l'integrazione delle componenti, sono stati eseguiti un totale di 3 *test* di integrazione, di cui 3 hanno avuto esito positivo.

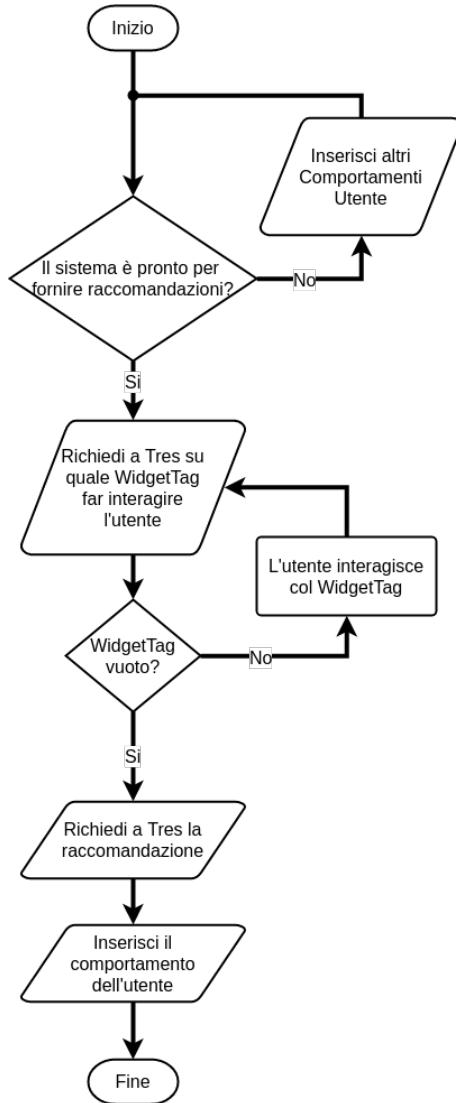
### Test di sistema

Ho effettuato alcuni *test* di sistema inerenti alle funzionalità di: inserimento dati, calcolo di una raccomandazione e infine calcolo probabilità.

- **Inserimento dati:** viene verificato il corretto comportamento del sistema alla richiesta di inserimento dati;
- **Calcolo raccomandazioni:** viene verificato il calcolo della raccomandazione da fornire;
- **Calcolo probabilità:** viene verificato il calcolo delle probabilità associate ai vari *item*.

## 3.6 Sommario funzionamento

Questa sezione descrive il funzionamento del sistema realizzato durante lo stage. Qui di seguito viene illustrato il funzionamento dal punto di vista del *client* che utilizza il modulo.



**figura 3.21:** Diagramma di sequenza: raccomandazione.

In fase di *startup* il sistema non possiede nessun dato. Esso quindi non è in grado di fornire raccomandazioni. Il *client* deve dunque verificare lo stato di operatività del sistema, attraverso una chiamata *HTTP* a:

GET /tres/ready

Il client ottiene così un oggetto **JSON** simile al seguente.

```

1 {
2   "ready": false
3 }
```

La risposta *HTTP* ritorna un oggetto **JSON** contenente un *booleano*, che indica l'operatività del modulo. L'operatività dipende essenzialmente dal numero di comportamenti utente inseriti nel sistema. Se il modulo non è operativo, si continua con la procedura di inserimento dei comportamenti tramite la chiamata:

POST /tres/behavior

L'oggetto da inviare tramite questa chiamata deve contenere un **JSON** simile al seguente.

```

1 {
2   "item": [
3     "tags": [
4       {
5         "name": "tag:uno"
6       },
7       {
8         "name": "tag:due"
9       }
10    ],
11   "interactions": [
12     {
13       "widgetTag": [
14         "name": "Income"
15       ],
16       "action": "High"
17     },
18     {
19       "widgetTag": [
20         "name": "PreviousCustomer"
21       ],
22       "action": "No"
23     }
24   ]
25 }
26 }
```

Un comportamento si registra solamente quando le azioni dell'utente hanno prodotto un risultato all'interno del sito (come ad esempio un acquisto nel caso di un *e-commerce*). Una volta generato l'albero, Tres è in grado di fornire raccomandazioni agli utenti. Il *client*, quindi, può iniziare la procedura per richiedere una raccomandazione. Esso, innanzitutto, deve richiedere a Tres quale argomento con cui interrogare l'utente. Mediante la seguente chiamata *HTTP* è possibile richiedere l'argomento:

POST /tres/answer

La chiamata deve comprendere un oggetto **JSON**, che contiene le interazioni che l'utente ha già svolto.

```

1 {
2   "interactions": [
3     {
4       "widgetTag": {
5         "name": "District"
6       },
7       "action": "Urban"
8     },
9     {
10      "widgetTag": {
11        "name": "HouseType"
12      },
13      "action": "Semi-Detached"
14    }
15  ]
16 }
```

Questa chiamata fornisce come risposta il prossimo *WidgetTag* da utilizzare, per interrogare l'utente del sito come il seguente oggetto **JSON**.

```

1 {
2   name: "Income"
3 }
```

L'utente va ad interagire con il *WidgetTag* con cui lo interroghiamo. Dopo di che il *client* aggiunge l'interazione alle interazioni precedentemente salvate. Il *client* deve continuare a reiterare questa chiamata, aggiungendo le interazioni. Finché Tres continua a fornire dei *WidgetTag*, per interrogare l'utente. Se Tres non ne fornisce altri, significa che la classificazione del comportamento è terminata. È quindi possibile per il *client*, procedere con la richiesta della raccomandazione. La chiamata *HTTP* da effettuare è la seguente:

POST /tres/recommendation

Nella chiamata dev'essere presente il **JSON** contenente le interazioni dell'utente. La risposta di Tres comprende un oggetto **JSON**, dove vengono indicate le percentuali dei prodotti/oggetti/post da raccomandare all'utente.

```

1 {
2   "items": [
3     {
4       "item": {
5         "tags": [
6           {
7             "name": "tag:uno"
8           },
9           {
10              "name": "tag:due"
11            }
12          ],
13        },
14        "percentage": 100
15      }
16    ]
17 }
```

Quando il comportamento dell'utente ha prodotto un risultato, il *client* inserisce il suo comportamento nel sistema e termina la procedura.



# Capitolo 4

## Valutazione Retrospettiva

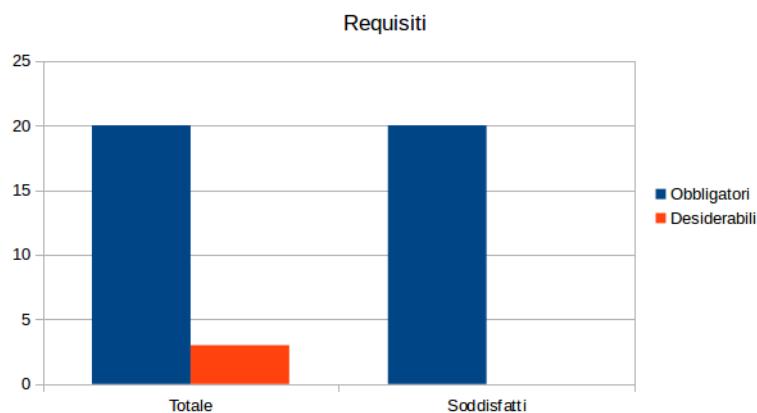
*Questo capitolo riporta un bilancio finale su quanto svolto durante lo stage.*

### 4.1 Bilancio sui risultati

In questa sezione riassumo gli obiettivi aziendali e gli obiettivi personali raggiunti durante lo stage.

#### 4.1.1 Obiettivi conseguiti

Gli obiettivi fissati all'inizio dello stage, hanno subito delle modifiche. L'azienda ha voluto privilegiare il *porting* di DRE. Il *porting* dell'applicativo è parzialmente completato. L'unica funzionalità non ancora implementata è una procedura di *map-reduce*. Questa funzionalità non l'ho completata, perché le mie competenze per comprendere quel codice erano insufficienti. Le parti su cui intervenire erano scritte in javascript, sistemarle avrebbe introdotto un ulteriore ritardo per lo sviluppo di Tres. L'obiettivo di migliorare la fase di apprendimento di questo modulo non l'ho raggiunto. Durante lo sviluppo di Tres ho individuato 23 requisiti, di cui, come raffigurato in 4.1, 20 obbligatori e 3 desiderabili.



**figura 4.1:** Riassunto requisiti

I requisiti desiderabili non soddisfatti riguardavano l'implementazione di algoritmi di *clustering*, pertanto questo scopo non è stato raggiunto. Tuttavia ho raggiunto l'obiettivo di implementare algoritmi per l'individuazione dei gusti dell'utente, grazie all'implementazione di *ID3*. In questo stage ho conseguito il *target* minimo riguardante l'utilizzo di OrientDb. Durante le due attività di porting e sviluppo, ho avuto modo di utilizzare questa tecnologia.

#### 4.1.2 Obiettivi personali

All'inizio dello stage, il mio scopo era di imparare nuove tecnologie da aggiungere alle mie conoscenze. Ritenevo il mio bagaglio professionale insufficiente, per affrontare il mondo del lavoro. Lo stage mi ha permesso di padroneggiare molte tecnologie, quali: OrientDb, Scala e un *web framework* di concezione moderna. Scala mi ha dato la possibilità di apprendere le nozioni per un corretto stile di programmazione funzionale. Mi ero prefissato di approfondire argomenti quali, intelligenza artificiale e i sistemi di raccomandazione. Ritengo questo parzialmente soddisfatto. Non ho trovato il supporto necessario per approfondire queste competenze, soprattutto per i sistemi di raccomandazione.

### 4.2 Bilancio formativo

In questa sezione riporto le competenze acquisite durante lo stage.  
Lo stage in Nextep Srl è stata la mia prima esperienza nel settore informatico. Mi ha permesso di apprendere nuove competenze e consolidare competenze già apprese durante il corso di studi. Ho trovato molto positivo lavorare collaborativamente al progetto, perché questo mi ha portato a migliorare le mie capacità di *team working*. Questa abilità è molto importante nel mondo del lavoro, soprattutto nei contesti lavorativi dove si utilizza una metodologia *agile*. Ho migliorato le mie capacità di *project management* per concludere il progetto nei tempi fissati e raggiungere gli obiettivi minimi che mi sono prefissato. In conclusione, ho terminato lo stage con un bagaglio professionale più ricco e di essere in grado di farmi carico di responsabilità più importanti.

### Scala

Durante lo stage ho avuto modo di apprezzare i vantaggi derivanti dall'uso di questo linguaggio. Ho utilizzato questa tecnologia nelle attività di *porting* del modulo DRE e nell'attività di sviluppo di Tres. Per studiare Scala ho seguito un corso *online*<sup>1</sup> tenuto direttamente dal professore Martin Odersky<sup>2</sup>, ovvero il *designer* del linguaggio stesso. Un corso validissimo per capire i meccanismi e le peculiarità di questo linguaggio. All'inizio è stato un po' ostico mettere in pratica quanto appreso, ma quando ho cominciato a padroneggiare Scala ho potuto constatare un aumento della mia produttività. Ho trovato un po' macchinoso gestire la compatibilità con librerie Java di terze parti. Mi ritengo soddisfatto di aver imparato questa tecnologia e di poter reinvestire nel mondo del lavoro questa conoscenza. Scala mi ha introdotto al paradigma di programmazione funzionale. Questo paradigma mi ha fornito una modalità diversa di affrontare e risolvere problemi, rispetto alla classica programmazione orientata agli oggetti. Nella programmazione funzionale vengono evitati i dati di stato e modificabili, mentre viene data invece una maggiore enfasi all'applicazione di funzioni. Mantenere gli stati immutabili facilita la suddivisione del codice per l'esecuzione parallela, garantendone la correttezza. Questa caratteristica è importantissima al giorno d'oggi, dove il focus si sta via via spostando sempre di più verso contesti multi-threading.

### OrientDb

Questa tecnologia è la soluzione di persistenza adottata durante il progetto. La curva di apprendimento è stata molto rapida, grazie al corso *online*<sup>3</sup>, tenuto durante l'attività di formazione. Mi ha permesso di imparare le differenze tra i modelli più tradizionali, come i relazionali e i modelli a grafo. I modelli a grafo si prestano molto bene per soluzioni nel dominio web, soprattutto in questo momento storico, dove il web si sta evolvendo in web semantico. E' stato un po' difficoltoso all'inizio passare dal concetto di *Join* delle tabelle al concetto di *Traverse* dei grafi. Tuttavia la modellazione del dominio con un grafo, è stata molto più semplice ed intuitiva rispetto ad un modello relazionale.

### Play Framework

L'apprendimento di questo *framework* non è particolarmente difficile. La documentazione presente nel sito è molto esaustiva, fornisce un supporto completo per la configurazione di una web application. Trovo molto vantaggioso la compatibilità con i linguaggi Java e Scala, in modo da poter sviluppare una applicazione con entrambi i linguaggi. La prima configurazione iniziale è stata molto semplice, questo mi ha permesso di focalizzarmi subito nello sviluppo del modulo. Purtroppo ho trovato complicato testare parti del mio modulo, perché era necessario, per il test, avere l'istanza del server avviata.

---

<sup>1</sup><https://www.coursera.org/course/progfun>.

<sup>2</sup><http://lampwww.epfl.ch/~odersky/>.

<sup>3</sup><https://www.udemy.com/orientdb-getting-started/>.

### 4.3 Distanza tra formazione universitaria e lavoro

In questa sezione espongo la valutazione personale, della distanza tra la formazione ricevuta durante il corso di studi e lo stage formativo. L'insegnamento che mi ha maggiormente preparato ad affrontare lo stage, è sicuramente *Ingegneria del Software*. Le competenze mancanti ad inizio stage non sono state molte. Il *gap* formativo è stato adeguato per essere colmato durante le attività svolte. Soprattutto durante l'attività di studio della prima settimana, che mi ha permesso di apprendere le conoscenze necessarie per affrontare le attività successive. L'attività di modellazione del *database* è stata la difficoltà maggiore incontrata durante il percorso. La modellazione di una base di dati a grafo, richiede un approccio differente al problema rispetto alle soluzioni relazionali. Una difficoltà minore riguarda la programmazione funzionale. Questo paradigma all'inizio mi è risultato difficoltoso da assimilare, perché abituato ai linguaggi orientati agli oggetti.

# Glossario

**API** in informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni 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. [49](#)

**CMS** strumento software installato su un server web studiato per facilitare la gestione dei contenuti dei siti web, svincolando l'amministratore da conoscenze tecniche di programmazione. [49](#)

**Core business** Il core business è l'attività principale di un'azienda, ovvero la base su cui si basa il fatturato e il guadagno a fine esercizio. [1](#), [47](#)

**DAO** è un pattern architettonale per la gestione della persistenza: si tratta fondamentalmente di una classe con relativi metodi che rappresenta un'entità tabellare di un RDBMS, usata principalmente in applicazioni web, per stratificare e isolare l'accesso ad una tabella/record tramite query (poste all'interno dei metodi della classe) ovvero al data layer da parte della business logic creando un maggiore livello di astrazione ed una più facile manutenibilità. I metodi del DAO con le rispettive query dentro verranno così richiamati dalle classi della business logic. [49](#)

**Framework** in informatica, e specificatamente nello sviluppo software, un framework è un'architettura logica di supporto su cui un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore. [3](#), [4](#), [13–16](#), [47](#)

**ICT** sono l'insieme dei metodi e delle tecnologie che realizzano i sistemi di trasmissione, ricezione ed elaborazione di informazioni (tecnologie digitali comprese). [49](#)

**IDE** software che fornisce allo sviluppatore numerosi aiuti durante lo sviluppo del codice. Esempi di aiuti sono la segnalazione degli errori di sintassi, interazione con strumenti di debug, suggerimenti di completamento del codice e strumenti per le build automatiche. [49](#)

**JSON** formato adatto per lo scambio dei dati in applicazioni client-server. Presenta una struttura semplice e di facile comprensione per l'uomo. Tale caratteristica ha contribuito alla sua rapida diffusione.. [49](#)

**MVC** il Model-View-Controller, in informatica, è un pattern architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti, in grado di separare la logica di presentazione dei dati dalla logica di business. [49](#)

**REST** un tipo di architettura software per i sistemi di ipertesto distribuiti come il World Wide Web. [49](#)

**UML** in ingegneria del software UML, Unified Modeling Language è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'UML svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. [49](#)

# Acronimi

**API** Application Program Interface. 17, 23, 28, 47

**CMS** Content Management System. 4, 47

**DAO** Data Access Object. ix, 31, 47

**ICT** Information Communication Technology. 2, 47

**IDE** Integrated Development Environment. 8, 13, 14, 36, 47

**JSON** Javascript Object Notation. 16, 24, 28, 34, 35, 40, 41, 47

**MVC** Model View Controller. ix, 3, 15, 26, 31, 48

**REST** Representational State Transfer. 17, 24, 25, 34, 35, 48

**UML** Unified Modeling Language. 22, 48



# Bibliografia

Riferimenti bibliografici

Siti Web consultati