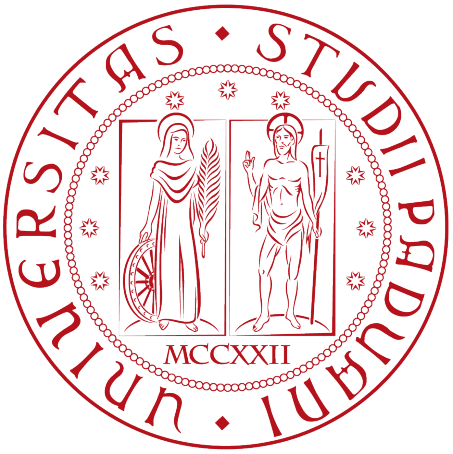
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Sperimentazione di soluzioni di sincronizzazione di applicazioni aziendali ERP contabile ed e-Commerce

Tesi di laurea

Relatore

Prof.Tullio Vardanega

Laureando

Jacopo Fichera

Anno Accademico 2021-2022

Le persone imparano solo quando le cose si mettono male.

— Hayao Miyazaki

Dedicato a quel bulletto di un carciofo che non è cattivo ma ha bisogno di affetto.

Indice

1 L'Azienda Ospitante **1**
- 1.1 Introduzione all'Azienda Ospitante . . . 1
- 1.2 Processi Aziendali . . . 2
  - 1.2.1 Metodologie di Sviluppo . . . 2
  - 1.2.2 Strumenti di Gestione e Versionamento . . . 3
- 1.3 Tecnologie Utilizzate . . . 3
  - 1.3.1 SIA . . . 3
  - 1.3.2 GitLab . . . 3
  - 1.3.3 Prestashop . . . 4
  - 1.3.4 Java . . . 4
  - 1.3.5 Spring . . . 5
  - 1.3.6 Altre Tecnologie . . . 6
- 1.4 Integrazione delle Tecnologie al Flusso di Lavoro . . . 7
- 1.5 Ricerca di Indipendenza e Innovazione . . . 8

2 La Proposta di Stage **9**
- 2.1 Il Problema . . . 9
- 2.2 Lo Scopo . . . 10
- 2.3 Obiettivi . . . 11
- 2.4 Vincoli Imposti . . . 11
- 2.5 Pianificazione Iniziale . . . 12
- 2.6 Obiettivi Personali . . . 14

3 Realizzazione del Prodotto **15**
- 3.1 Studio Preliminare delle Tecnologie . . . 15
  - 3.1.1 Motivazioni sulla scelta di Prestashop . . . 15
  - 3.1.2 Motivazioni sulla scelta di Java . . . 16
- 3.2 Analisi dei Requisiti . . . 16
- 3.3 Progettazione . . . 20
  - 3.3.1 Modello e Comunicazione . . . 20
  - 3.3.2 Operazioni e Web Server . . . 21
  - 3.3.3 Design Pattern Utilizzati . . . 23
- 3.4 Codifica . . . 24
  - 3.4.1 Accesso ai Dati a Prestashop e Sql Server . . . 24
  - 3.4.2 Dependency Injection e classi di Configurazione in Spring . . . 24
  - 3.4.3 Software dall'esecuzione in parallelo su Spring . . . 26
  - 3.4.4 RESTful Webservice per la ricezione di comandi . . . 27
- 3.5 Verifica e Validazione . . . 28

3.6 Prodotto Risultante 28

4 Valutazione retrospettiva sulle attività **31**
4.1 Raggiungimento degli Obiettivi 31
4.1.1 Obiettivi Aziendali 31
4.1.2 Obiettivi Personali 33
4.2 Conoscenze Acquisite 33
4.3 Considerazioni Finali 34

Bibliografia **35**

Acronimi **37**

Glossario **39**

Elenco delle figure

1.1 Schema riassuntivo di Scrum 2
1.2 Struttura riassuntiva GitLab IMTEAM 4
1.3 Struttura di Spring Runtime Framework 5
1.4 Integrazione delle Tecnologie al Flusso di Lavoro 7

2.1 Cattura di schermo di Ad Hoc Revolution 10

3.1 Schema riassuntivo di Scrum 17
3.2 Schema riassuntivo di Scrum 18
3.3 Struttura di ModelComponent e interfaccia di adattamento 20
3.4 Struttura di ModelComponent e interfaccia di adattamento 21
3.5 Struttura alla radice di *Action* in relazione a *Status* 22
3.6 Struttura di *Action* in relazione alle sue implementazioni astratte . . . 22
3.7 Struttura del sistema per la gestione univoca di chiavi 23
3.8 Estratto codice Get di PrestashopDao 25
3.9 Estratto codice Get di PrestashopDao 25
3.10 Classe di configurazione per ProductXmlDao 26
3.11 Classe di definizione di ProductXmlDao 26
3.12 Funzione di mappatura del *@RestController* per gli articoli 27
3.13 Classe di configurazione per ProductPrestashopDao 29

Elenco delle tabelle

3.1 Tabella di associazione tra richiesta *HTTP* e annotazione *Spring* . . . 27

4.1 Tabella dei Requisiti Funzionali effettivamente soddisfatti 31

Capitolo 1

L'Azienda Ospitante

1.1 Introduzione all'Azienda Ospitante

SIATEC è un'azienda del gruppo IMTeam, fondata nel 2007. L'impresa si specializza nella rivendita di sistemi gestionali prodotti dalla Zucchetti, di cui è partner ufficiale, e allo sviluppo di *software* per l'automazione a essi integrati.
Per l'azienda l'interazione con un cliente non termina con la vendita, infatti SIATEC prende in carico, a meno che su specifica richiesta, la gestione del Enterprise Resource Planning (ERP) venduto e propone un servizio di assistenza clienti completo.
I progetti per il settore di automazione di SIATEC sono integrazioni tra macchinari monitorabili e il sistema ERP del cliente, come definito dalle normative dell'industria 4.0. Vengono perciò sviluppati software per la gestione del lavoro sui dispositivi in rete e per la collezione di dati emessi dalle apparecchiature di produzione.

L'azienda fornisce diversi sistemi gestionali, quello di maggior rilievo è *Ad Hoc Revolution*, uno strumento per gestione per piccole e medie imprese che integra una vasta gamma di processi di *business* d'interesse per un'azienda. Questi, ad esempio, potrebbero essere l'organizzazione di magazzini, la logistica o come l'organizzazione dei magazzini e la logistica o le vendite ai clienti.
Il servizio è interamente localizzato su una piattaforma unica per un efficace controllo delle risorse della compagnia.

SIATEC si mette anche a disposizione dei clienti per lo sviluppo di funzionalità su misura per questi sistemi, in modo da dare allo specifico cliente un'esperienza più comoda possibile. Grazie all'appartenenza a un gruppo di collaborazione che copre diversi settori, le richieste dei clienti anche meno attinenti alle competenze dell'azienda tendono a essere soddisfatte grazie al coinvolgimento di personale esterno. La visione aziendale operata punta infatti a soddisfare il cliente coinvolgendolo frequentemente per ricevere *feedback* sui servizi prestati. Da queste interazioni possono eventualmente emergere problemi che ha riscontrato, per cui SAITEC propone delle soluzioni in modo da migliorare la sua esperienza con il *software* gestionale. L'azienda propone in queste interazioni soluzioni a problemi che, eventualmente, emergono.

Quanto riguarda l'organizzazione interna, SIATEC non sottopone i dipendenti a rigide regole, nonostante la presenza di una gerarchia lavorativa. L'ambiente che si è formato cerca infatti di porre tutti gli elementi allo stesso livello per un confronto più sincero ed efficace. Questo per non mettere pressioni sul lavoratore, che è libero di esprimere i propri dubbi o difficoltà. Il fatto si riflette nella gestione autonoma del lavoro, dove i dipendenti si trovano ad avere un alto grado di libertà.

1.2 Processi Aziendali

1.2.1 Metodologie di Sviluppo

Lo sviluppo di un progetto in SIATEC passa una fase preliminare di analisi, il cui livello di dettaglio è relativo alla complessità del problema da risolvere.
La soluzione a certe richieste potrebbe essere facilmente realizzata, per cui un'analisi troppo approfondita, sebbene in sè non un male, non è sempre necessaria. Data questa motivazione, in accordo con il responsabile d'ufficio, si decide a priori quanto si desidera approfondire l'analisi. In questa fase vengono redatti Casi d'uso o *User stories*, quale dei due adottare viene stabilito in base alla valutazione effettuata in precedenza. Da essi vengono poi stabiliti i requisiti da raggiunger al fine di realizzare il prodotto.

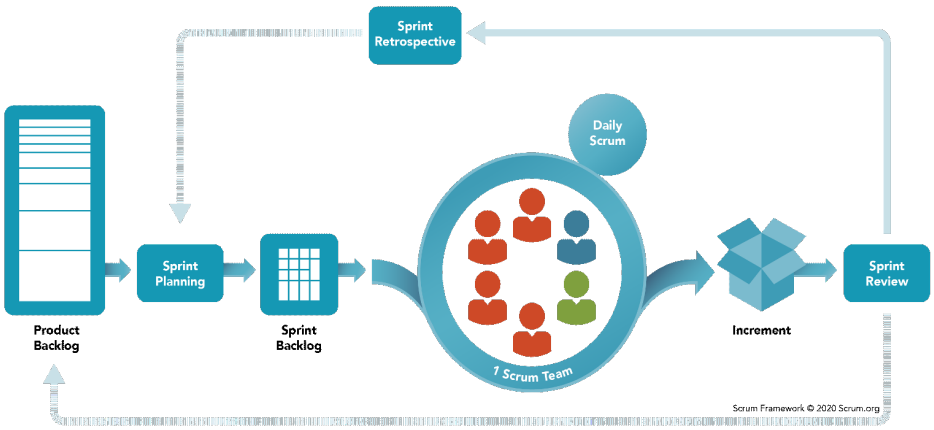


Figura 1.1: Struttura schematica del normale flusso di di un progetto da *Scrum.org*.

SIATEC adotta per lo sviluppo i suoi progetti un modello Agile. In particolare l'azienda adopera in maniera non rigorosa il framework *Scrum*.

Seguendo la rappresentazione in figura 1.1 a partire da sinistra, dai requisiti tracciati viene realizzato un *Backlog*, una raccolta di *task* da svolgere per realizzare il prodotto. Questa è la base alla pianificazione degli *Sprint* per tutta la durata del progetto. L'organizzazione del lavoro interessa l'intero *staff* coinvolto nella realizzazione del prodotto che fissa un obbiettivo da raggiungere e divide il lavoro tra le diverse componenti.
Terminata questa fase organizzativa inizia lo *Sprint*, la cui durata è sempre la medesima in un progetto ed è relativa alla complessità del problema.

Poiché in SIATEC le attività di ogni membro del *team* vengono registrate durante il corso di uno *Sprint* non si effettua un *Daily Scrum*, *meeting* giornaliero, ma eventuali incontri, se ritenuti necessari, devono essere esplicitamente richiesti. All'occorrenza di uno di questi solamente i membri del *team* richiesti a trarre conclusioni sono tenuti a essere presenti, gli altri sono liberi di partecipare. Lo Scrum Master è sempre disponibile per un confronto se necessario.

Al termine di un incremento si esegue una *review* interna e, ad alterni, avviene un incontro con il cliente per aggiornarlo sullo stato dei lavori. A questo SAITEC mostra un potenziale rilascio dell'applicativo per ricevere del *feedback*, rendendolo un soggetto direttamente coinvolto nello sviluppo.

In azienda, prima di avviare una nuova fase di pianificazione, è possibile avvenga una valutazione retrospettiva delle attività. Questo solitamente avviene nel caso siano state riscontrate criticità nell'incremento passato.

1.2.2 Strumenti di Gestione e Versionamento

In SIATEC il lavoro viene gestito mediante l'uso della piattaforma SIA, la cui descrizione è riportata in sezione 1.3.1. Mediante il *software* ogni dipendente del gruppo IMTEAM ha modo di tracciare i propri programmi e condividerli con altri utenti che potrebbero esservi interessati. Questo favorisce inoltre l'interazione tra aziende distinte del gruppo di collaborazione.

Il servizio di gestione è integrato al sistema di controllo versione *GitLab*. Sulla piattaforma vengono anche definiti i progetti Agili a livello organizzativo in quanto essa mette a disposizione vari strumenti utili alla gestione del lavoro.

I due servizi sono gestiti internamente al gruppo aziendale, per cui SIATEC ha un ambiente unico per la realizzazione di progetti interamente localizzato sulla rete locale.

Il *Branching* di una *repository* di progetto avviene sulle *features* da integrare all'applicativo che vengono, una volta terminate, portate al ramo di sviluppo. Le modifiche raggiungono il *branch* di produzione solo in seguito a un collaudo del sistema.

1.3 Tecnologie Utilizzate

Nel corso del progetto ho usato diverse tecnologie per la realizzazione del prodotto, alcune delle quali non ancora adoperate nel contesto aziendale della SIATEC.

1.3.1 SIA

SIA è una *webapp* realizzata internamente al gruppo aziendale IMTEAM pensata per fornire ai dipendenti delle varie aziende strumenti per l'organizzazione del lavoro. Questo permette di tenere tracciati eventi programmati e attività svolte condividendole, eventualmente, con altri utenti che potrebbero voler essere messi al corrente.

Il servizio integra funzionalità di *GitLab* mediante la *API* esposta. Da esso vengono generate due possibili pagine: una sullo stato più generale di svolgimento di un progetto e una sul lavoro pianificato per lo specifico dipendente.

SIA è un ambiente più contenuto rispetto a Gitlab e ha come scopo quello di riassumere semplicemente lo stato del lavoro, motivo per cui la gestione del progetto resta su *GitLab*.

SIA, al momento, è a solo uso interno del gruppo e in continuo sviluppo.

1.3.2 GitLab

Gitlab è un potente software *web* di controllo versione *Open Source* prodotto e mantenuto da GitLab basato sulla tecnologia *Git*. Esso si distingue dai competitori per la sua attiva comunità di sviluppatori e dall'articolato sistema di organizzazione che fornisce. Questo lo rende infatti un tool adatto anche alla gestione di progetti Agili.

L'uso della piattaforma è condiviso tra le aziende del gruppo, come accennato in 1.2.2 e per evitare che qualcuno acceda a file non di sua competenza l'azienda fa uso dei *Groups*. Questi sono dei raggruppamenti effettuati secondo la convenzione interna di un gruppo definito per azienda di IMTEAM, come illustrato in figura 1.2 per SIATEC.

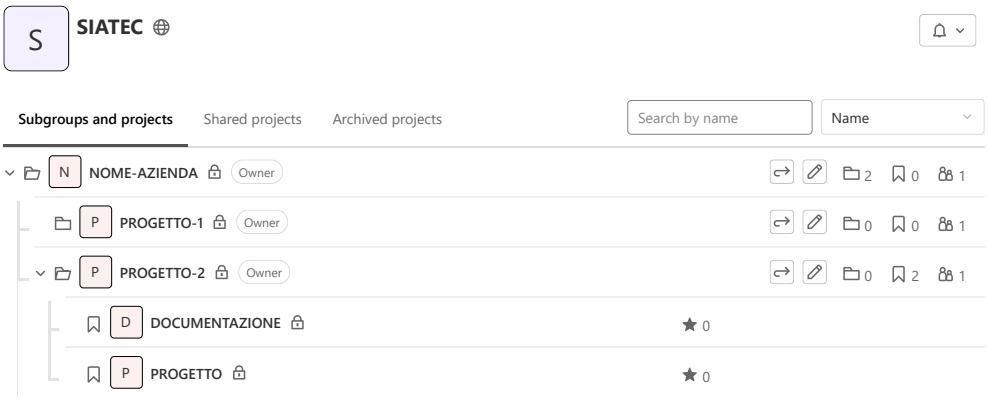


Figura 1.2: Struttura esemplificativa per gestione di progetti *GitLab*.

L'azienda lavora molto con clienti abituali per cui ha deciso di definire sotto-gruppi per ognuno di essi. Inoltre, data la limitazione di una sola repository per progetto sulla piattaforma, per ogni prodotto *software* è a sua volta definito un gruppo. Questo in quanto SIATEC lavora differenziando la documentazione del prodotto dalla sua codifica, infatti in immagine vediamo i due progetti "PROGETTO" e "DOCUMENTAZIONE".

La gestione di progetti Agili segue le linee guida fornite direttamente da *GitLab*.

1.3.3 Prestashop

Prestashop è il Content Management System (CMS) *Open Source* per la realizzazione di siti di *E-Commerce* leader del mercato Europeo e Latino-Americano studiato per:

> *"Distribuire facilmente le proprie attività su scala più ampia."*
> (Prestashop Team)

Il Software, realizzato in PHP, è strutturato in modo da essere di facile utilizzo a qualsiasi tipo di utenza, sia persone con discrete capacità informatiche che sviluppatori. Ai primi la gestione è infatti permessa mediante un *Back Office* completo e articolato. Per gli sviluppatori invece sono disposte diverse tecnologie, come una REST API, oltre che un'esaustiva documentazione dei sorgenti e della struttura del software nel suo complesso.

Nell'ambito del progetto è stata scelta come tecnologia da integrare al sistema gestionale del cliente; le motivazioni di questa decisione sono discusse in: 3.1.1.

1.3.4 Java

Java è un linguaggio di programmazione interamente orientato agli oggetti di alto livello fortemente tipato. L'esecuzione di software in questa tencologia è vincolata all'omonima VM, motivo per cui gli applicativi sviluppati in *Java* sono eseguibili su qualsiasi dispositivo compatibile con essa.

Per via della sua versatilità Java si è ben presto affermato tra gli sviluppatori, diventando il linguaggio di programmazione più diffuso al mondo.

Per lo sviluppo è stata adottata la Open JDK 14 a libera licenza GPL 2.0 e *Maven* 3.8.2, un *tool* per la gestione delle dipendenze dei progetti.
La scelta del linguaggio è stata parte dell'analisi preliminare come discusso in: 3.1.2.

1.3.5 Spring

Spring è il framework per Java *Open Source* più diffuso al mondo. Esso è pensato per rendere la creazione e manipolazione di progetti più rapido e più semplice sia a chi è coinvolto che a coloro che potrebbero doverci mettere mano in un secondo momento.

La tecnologia alla radice, *Spring Framework*, propone un set di concetti e regole dalle quali sono poi state sviluppate idee distinte. Da queste sono stati realizzati dei progetti che forniscono complessi strumenti per diversi campi di applicazione. Un esempio di ciò è *Spring Boot*, che semplifica la realizzazione di applicazioni *standalone* senza la necessità di fare una *deploy* dei file. La realizzazione di questi progetti *Spring* indipendenti è possibile grazie alla natura modulare del *Spring Framework Runtime*, come rappresentato in 1.3. Esso infatti è composto da venti moduli principali combinabili in base alle necessità di uno sviluppatore. Questo rende la dipendenza dal framework vincolata a quanto realmente utilizzato al fine del progetto e non dall'intera struttura della tecnologia. Nell'attività di stage ho ad esempio ampiamente sfruttato classi del modulo *Web* per la definizione di richieste *REST* alla tecnologia di *E-Commerce*, mentre *AOP*, non interessante ai fini del problema, non l'ho utilizzato.

I concetti proposti da *Spring* cambiano il modo di gestire un progetto che risulta distinguersi notevolmente da come sarebbe se non fosse sviluppato in *Vanilla*

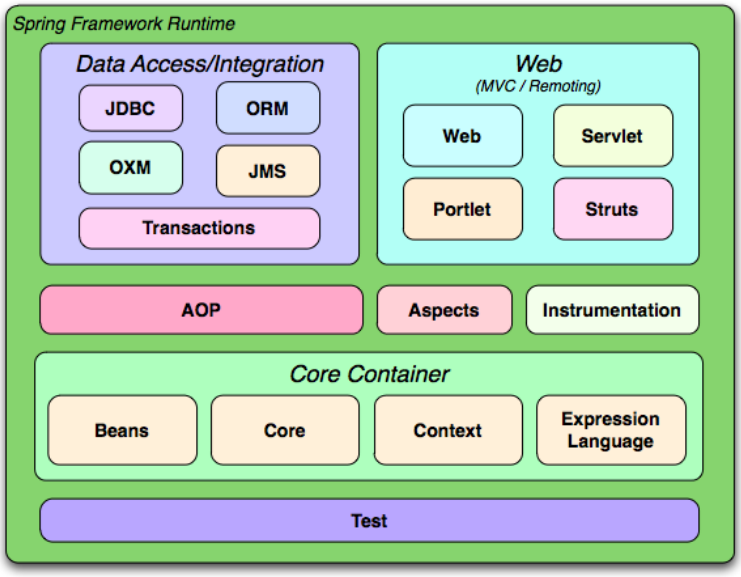


Figura 1.3: Struttura di *Spring Runtime Framework*, i venti moduli principali. Immagine tratta dalla documentazione ufficiale di *Spring*.

Sono numerose le caratteristiche notevoli di *Spring*, forse la più utile e interessante è la gestione della *Dependency Injection*. Questa viene eseguita automaticamente mediante l'inversione di controllo grazie a un *IoC Container*.

Nei progetti del framework è possibile definire dei *Beans*, non da confondere con quelli comunemente usati in Java, che sono tutte quei oggetti il cui contesto è volutamente programmato mediante file di configurazione XML o delle annotazioni.

I *Beans* così definiti vengono costruiti dalla *BeanFactory*, una sofisticata implementazione del *Factory Pattern*. Questo strumento dà ai progetti *Spring* la possibilità di slegare la configurazione e le dipendenze del software dall'attuale logica del programma, in quanto il ciclo di vita di questi oggetti viene delegato allo IoC Container.

Nel progetto ho sfruttato una molteplicità di funzionalità integrate al *framework* di *Spring*, come: funzionalità web integrate, servizi di comunicazione REST, funzionalità di accesso a banche dati. A queste ho combinato l'uso di alcuni pacchetti, ad esempio *Spring Boot*, citato precedentemente, e *Spring Data*.

SIATEC ha adoperato per la prima volta *Spring* con l'attività di stage con l'intento di velocizzare il lavoro e di sperimentare l'applicabilità del *framework* ad eventuali progetti futuri.

1.3.6 Altre Tecnologie

Per quanto Spring sia completo sono state utilizzate tecnologie supplementari:

* JAXB: *Framework* per la generazione di oggetti in istanze di Classi da XML e viceversa, utile per alla comunicazione con la REST API esposta da Prestashop.
* MSSQL-JDBC: *Driver* per la comunicazione verso una banca dati di tipo SQL gestita da un Database Management System (DBMS) *Sql Server*.
  Nel progetto essa era la sorgente di dati del ERP della Zucchetti.
* IntelliJ Idea Enterprise: Integrated development environment (IDE) di sviluppo realizzato della JetBrains per Java, *Kotlin* e lo sviluppo *Web*.
  Esso ha modo di integrare sistemi di controllo versione basati su Git.
  La versione *Enterprise* dispone di funzionalità per gestire progetti *Spring*.

1.4 Integrazione delle Tecnologie al Flusso di Lavoro

Dopo aver effettuato una prima definizione del *Backlog*, il flusso di lavoro di un progetto *Scrum* in SIATEC è come mostrato in figura 1.4.

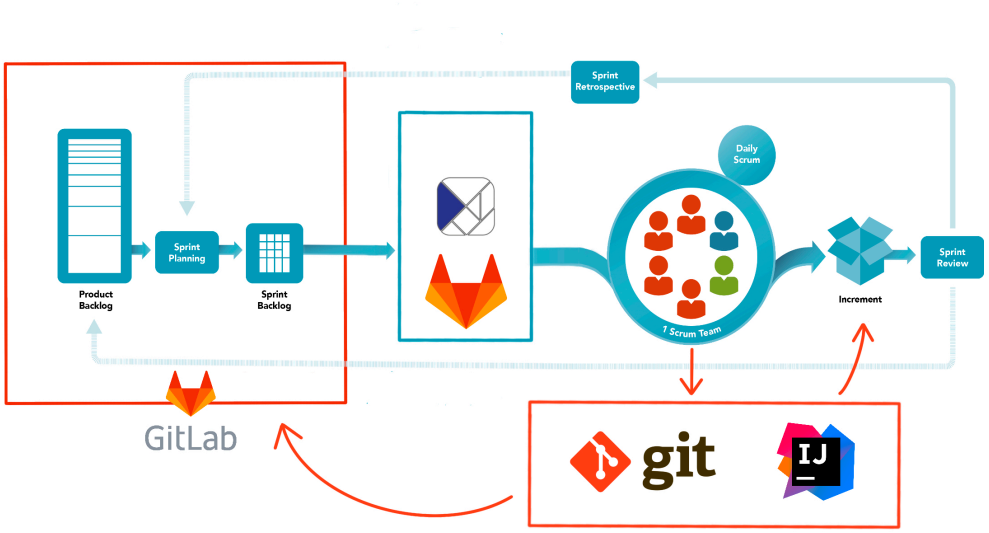


Figura 1.4: Schema esemplificativo del flusso di lavoro e delle tecnologie adoperate per un progetto scrum in SIATEC (modifica dell'immagine di *scrum.org*)

Dall'immagine notiamo tre principali riquadri:

* **Riquadro Rosso** (Gitlab): La tecnologia alla radice dei progetti *Agile*, essa, infatti espone molti strumenti utili per la definizione di questi.
  GitLab è alla base un servizio di controllo versione, motivo per cui si può considerare la fondazione di ogni progetto sviluppato in SIATEC.

* **Riquadro Azzurro**: Evidenzia gli strumenti utili alla gestione del lavoro.
  Un utente ha modo di registrare le proprie attività e tracciare quelle altrui interfacciandosi direttamente al SIA. Quelle legate a *task* e diversi altri elementi di sviluppo *Agile* sono ancora vincolati alla piattaforma *Gitlab*, per questo motivo è impossibile il solo uso del sito SIA al momento.

* **Riquadro Arancione**: Strumenti per lo sviluppo.
  GitLab è alla radice un servizio *Git*, per cui è possibile interfacciare il proprio ambiente di sviluppo, se questo lo consente, alle repository. In alternativa è possibile utilizzare il terminale nativo della tecnologia. Quale dei due metodi adottare è a discrezione dello sviluppatore, in entrambi i casi il prodotto si troverà poi localizzato sulla piattaforma di *GitLab*.

Dopo aver definito uno *Sprint* assegnando i task ai membri del *team* SIATEC da il via allo sviluppo. Questo si svolge come quanto accennato nella sezione 1.2.2.
Il codice prodotto sarà poi localizzato in opportuni branch della piattaforma per la durata dell'incremento.

Inoltre, sia per quantificare il tempo richiesto per lo svolgimento dei *task* che per l'organizzazione interna dell'azienda, le ore svolte da un membro del team, quali che siano, vengono tracciate su SIA dove sono dotate di una breve descrizione dell'attività.

1.5 Ricerca di Indipendenza e Innovazione

SIATEC, da quel che sono riuscito a cogliere durante il corso dello stage, sta cercando di ridurre la dipendenza che ha dai prodotti Zucchetti. Al tal fine essa offre delle soluzioni proprie, acquistando in libertà di sviluppo con il vantaggio di non dover investire ulteriormente nelle pubblicazioni dell'azienda socia.

Volendo l'azienda proporre ai propri clienti soluzioni sviluppate internamente tanto accattivanti quanto moderne, nasce la necessità per SIATEC di ricercare le più attuali tecnologie sul mercato. L'azienda ha perciò posto molto interesse nelle mie opinioni sulle pratiche interne del lavoro oltre che proprio alla progettazione del prodotto e allo studio delle tecnologie. Questo mi ha dato l'idea che l'azienda abbia dato un forte valore alla mia personale formazione accademica. Da questo nasce il desiderio di SIATEC di avvicinare giovani menti e lasciarle pensare liberamente. Questo al fine di permettere alla compagnia di non restare indietro rispetto ai competitori, incitando i dipendenti a utilizzare le conoscenze apprese durante la loro personale formazione.

Capitolo 2

La Proposta di Stage

2.1 Il Problema

L'esposizione mediatica sul web è diventata, per molti *business*, necessaria per poter essere competitivi nel mercato odierno. Gli *E-Commerce* sono un'interessante opzione per molte aziende. Il loro utilizzo tuttavia comporta delle problematiche relative al bisogno d'interfacciare la piattaforma di *E-Commerce* al sistema ERP per poter eseguire correttamente i processi interni dell'azienda.

SIATEC, al momento, vende e gestisce principalmente gestionali della linea *Ad Hoc*. L'azienda punta a sfruttare la struttura regolare di questi software per la realizzazione di un sito di *E-Commerce* generico in grado di essere riadattato al ERP del singolo cliente. I software *Ad Hoc* sono infatti elaborati Enterprise Resource Planning (ERP) che gestiscono tutti i processi interni di un'azienda. L'immagine 2.1 espone solo alcune delle numerose possibili catture di schermo per la gestione di un *business*. In figura si può intuire la grande varietà di funzionalità presenti, con menu dedicati a vari ambiti aziendali, a partire dalla gestione archivi fino ad arrivare al controllo della produzione.

Il gestionale*Ad Hoc* è modulare, perciò è possibile definire nuovi componenti specifici alle necessità del cliente. Ogni elemento esposto ha inoltre una finestra dedicata che si integra con altri oggetti del ERP per una rapida gestione delle attività aziendali.

Per motivi pratici l'azienda ha deciso di appoggiarsi a un CMS *Open Source* come sito di vendita *online* i cui dati sono reperiti dal ERP del cliente. Per sua , una piattaforma d'*E-Commerce* integra una sua banca dati dalla struttura tabellare definita. Da questa caratteristica sorge la problematica di dover adattare le informazioni tra i due *database* nella fase di sincronizzazione, in modo da renderle compatibili.

Durante l'attività di stage mi è stato richiesto di svolgere una prima fase di analisi delle tecnologie da adottare. Ho quindi eseguito un breve studio per la scelta del CMS e una valutazione sul linguaggio di sviluppo per il servizio di sincronizzazione.

In seguito mi è stata chiesta la realizzazione del suddetto servizio di sincronizzazione tra i *database*, che deve poter avvenire per i dati in entrambe le direzioni. In particolare, per l'ERP saranno di rilievo i nuovi ordini registrati, mentre per il CMS potrebbero esserlo i nuovi articoli dell'azienda.

SIATEC, in un primo momento, ha vincolato il commercio del sito di *E-Commerce* al Business To Business (BTB). Questo elimina la fase di registrazione degli acquirenti alla piattaforma, in quanto tutti questi saranno pre-registrati sul gestionale del cliente e perciò opportunamente generati sullo store *online*.

Parte dell'attività comprendeva anche la scelta di un dominio applicativo per dare

un concreto punto di partenza al software, questo ambito è definito dal cliente a cui è stato proposto il prodotto.

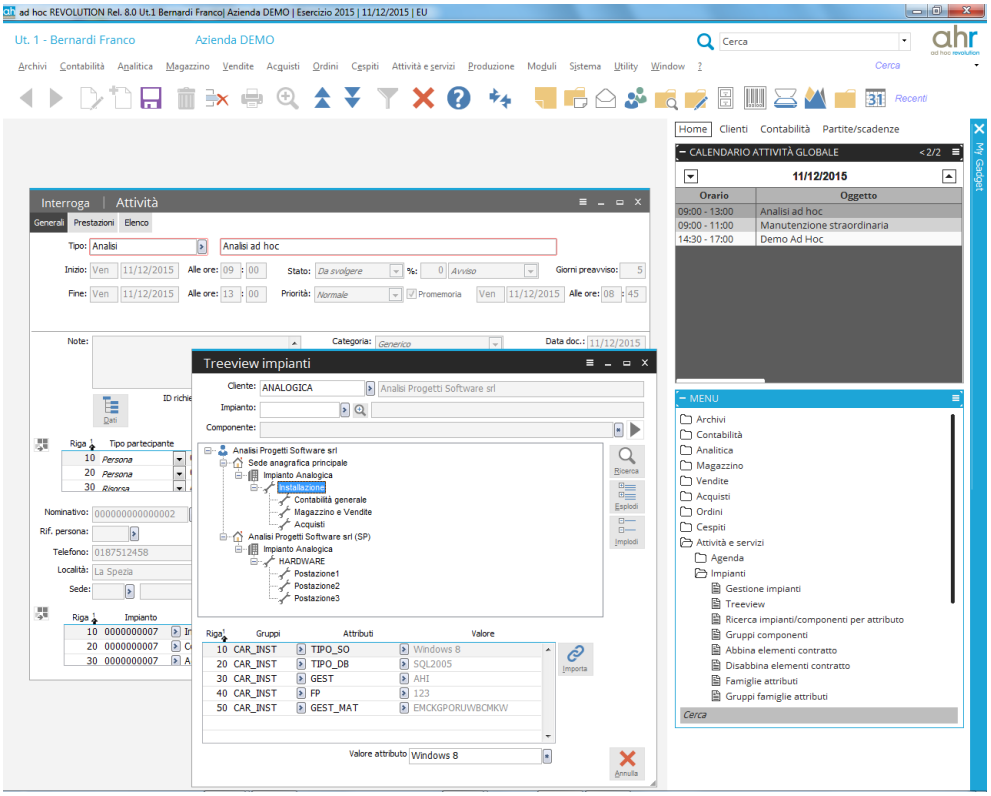


Figura 2.1: Cattura di schermo della *UI* di *Ad Hoc Revolution* tratto dal sito della Zucchetti.

2.2 Lo Scopo

L'azienda, con l'attività di stage, desiderava investire tempo nello studio e nella valutazione di nuove tecnologie per lo sviluppo di progetti futuri e al tempo stesso produrre un software che, integrato a un CMS, fosse a tutti gli effetti commercializzabile.

SIATEC lavora principalmente con prodotti della famiglia *Ad Hoc* che forniscono strumenti su misura per gli sviluppatori, come ad esempio un editor grafico, permettendo comunque la codifica del *software* mediante un dialetto di *Java*. L'azienda è tuttavia in cerca di alternative per eventuali sviluppi futuri, perciò la scelta del linguaggio da adottare nel progetto è stata di rilievo. Essa è stata dettata da due principali fattori: la semplicità e il supporto che la tecnologia riceve.

Per questo motivo risulta evidente che per l'azienda non è solo importante il *software* prodotto, ma tutte le attività coinvolte nello svolgimento dello stage. Mi è infatti stata richiesta anche un'analisi delle esigenze funzionali, gestionali e tecniche specifiche al progetto.

Al termine del progetto, dal punto di vista del cliente finale, abbiamo proposto una pagina *web* per la vendita *online* generata dai dati del suo sistema ERP. Questo

processo è trasparente, il cliente quindi non deve essere realmente conscio dell'esistenza di due diversi *database* per i suoi servizi per continuare a utilizzare il suo gestionale.

SIATEC, tramite lo stage, ha invece richiesto, come prodotto finale, un software che fosse il più generico possibile. Questo fatto si traduce nella possibilità che il servizio di sincronizzazione sia facilmente modificabile in base alle particolari esigenze. I cambiamenti sul *software* potrebbero essere dettati dalle necessità di uno specifico cliente, oppure dal desiderio di SIATEC di sostituire una, o entrambe, le piattaforme coinvolte.

2.3 Obiettivi

Dalla definizione del problema, la proposta di stage prevedeva il raggiungimento di questi obiettivi principali:

* **Produzione di una descrizione funzionale**
  Un documento che descrivesse le funzionalità richieste al prodotto in relazione al dominio del problema. Questa analisi doveva far riferimento a un ambito di applicazione reale la cui scelta è stata parte dell'attività di stage. Era mio compito individuare gli elementi da gestire al fine di realizzare il servizio. Questi sono tutti quelli che danno un significato operativo alla piattaforma di *E-Commerce*, ad esempio: i clienti, gli ordini e gli articoli. Da questi ho dovuto effettuare uno studio sulle funzioni richieste sia a lato CMS che ERP al fine di trasferire solo i dati utili alle due piattaforme.

* **Produzione di una descrizione tecnica**
  Un elaborato che descrivesse l'architettura *software* dell'applicativo e delle eventuali API esposte, includendo le scelte tecnologiche.

* **Studio sulle tecnologie da adottare per il progetto**
  Lo studio delle tecnologie da adottare per il progetto in cui ho discusso la scelta relativa al CMS e al linguaggio di sviluppo, con eventuali proposte sui *framework* da utilizzare.

* **Realizzazione del prodotto**
  Il *software*, l'obbiettivo finale del progetto, che doveva essere basato sulle due analisi effettuate.

2.4 Vincoli Imposti

L'azienda, per lo sviluppo del progetto, mi ha posto dei vincoli da rispettare.

Vincoli di Progettazione. SIATEC ha richiesto la produzione di una documentazione esaustiva del prodotto che includesse il flusso dei dati con i due *database*. In particolare, era necessario che fosse esplicita la natura della comunicazione verso i due servizi, che in base alla loro posizione relativa in rete si interfacciano tra loro.

La produzione documentale richiesta prevedeva la stesura di un breve studio sulle possibili tecnologie di CMS adottabili. Tra queste poi doveva essere selezionata la più valida, evidenziandone caratteristiche notevoli e le motivazioni che ne hanno portato alla scelta.

Per il prodotto è stata richiesta anche la stesura di un'analisi funzionale in cui sono stati definiti i casi d'uso, vista la natura complessa del servizio. Il *software* doveva prima essere progettato e poi realizzato. Dunque, oltre al processo di codifica, è stata richiesta la produzione di un diagramma delle classi secondo lo standard Unified Modeling Language (UML) 2.0.

La scrittura sulla banca dati gestionale deve avvenire solo mediante *Stored Procedures* le quali, se non già presenti sul sistema, sono state opportunamente prodotte.

Vincoli di Pianificazione L'azienda ha richiesto che lo sviluppo del progetto procedesse in concordanza con quanto definito dai processi interni, come descritto in 1.2.1. La codifica del prodotto è quindi stata preceduta da una prima analisi e poi dalla realizzazione del *backlog* su cui si sono infine basati gli *Sprint*.

La durata di un incremento è stata fissata a una settimana lavorativa.

Vincoli Tecnologici. Volendo SIATEC utilizzare tecnologia nuove per la loro realtà aziendale, mi sono stati posti solo pochi vincoli su questo fronte.
Per il progetto è stato richiesto l'uso di *Ad Hoc Revolution* come prodotto ERP che si appoggia al DBMS *Microsoft Sql Server*. Era mio compito invece quello di proporre tecnologie da adottare nello sviluppo, queste sono state sottoposte alla valutazione del mio *tutor* aziendale.

Inoltre, se un servizio si trova in rete locale con l'applicativo, è concesso che i due comunichino mediante una connessione diretta. Se invece non fosse questo il caso, è richiesto che i due si interfaccino mediante *Webservice* opportunamente definito.

2.5 Pianificazione Iniziale

Lo stage si è svolto a partire dal 01/09/2021 per un totale di 320 ore che dovevano essere divise in otto settimane lavorative.

Durante lo svolgimento dell'attività, alcune settimane sono stato assente. Questo lo avevamo precedentemente pattuito perciò il termine che sarebbe stato il 27/10 è slittato al 5/11.
SIATEC ha proposto una pianificazione dell'attività che viene riportata in seguito.

Ogni settimana del programma è dotata, oltre che di singoli punti da svolgere, di un breve riassunto dell'attività nel suo complesso o di osservazioni degne di nota.

Prima Settimana. Ambientamento in azienda; selezione e preparazione degli strumenti per il successivo sviluppo del prodotto.

- Predisposizione degli spazi e strumenti di lavoro
- Studio e valutazione tra i diversi CMS *Open Source* che porta alla scelta della piattaforma da adottare per il progetto
- Creazione dell'infrastruttura in collaborazione con il gruppo di sistemistica ed i referenti ERP per l'installazione delle tecnologie
- Installazione del sito *web* di commercio e del sistema ERP

Seconda Settimana. Studio del problema e scelta del campo di applicazione, che comporta quindi la proposta del prodotto a un cliente interessato al servizio.

- Studio e analisi delle funzioni applicative di un *E-Commerce*

* Studio e analisi delle funzioni applicative di un ERP
* Proposta e incontro con utenti finali per la definizione funzionale

Terza Settimana. Progettazione del prodotto finale.

* Progettazione della comunicazione tra i due sistemi
* Redazione del documento funzionale dall'analisi precedente
* Progetto delle componenti software con definizione delle relative funzioni e proposta delle tecnologie da adottare

Quarta Settimana. Valutazione della progettazione fatta da parte di altro personale e successivo sviluppo delle basi del software come punto di partenza.

* Riesame della progettazione funzionale con gli utenti finali
* Riesame della progettazione software con i referenti tecnici
* Formazione per le tecnologie di sviluppo
* Sviluppo della struttura alla radice del software

Quinta Settimana. Inizio degli *Sprint* di *Scrum* perciò la pianificazione preventiva del lavoro, a partire da questa settimana, è meno rigida.

* Programmazione dell'attività settimanale
* Studio o approfondimento delle tecnologie di sviluppo
* Sviluppo del software

Sesta Settimana. Secondo *Sprint*.

* Valutazione dello stato di avanzamento
* Programmazione dell'attività settimanale
* Studio o approfondimento delle tecnologie di sviluppo
* Sviluppo del software

Settima Settimana. Terzo *Sprint* e valutazione sull'avanzamento dello sviluppo del software con il cliente.

* Valutazione dello stato di avanzamento con gli utenti finali
* Programmazione dell'attività settimanale
* Sviluppo del software

Ottava Settimana. Quarto *Sprint* e preparazione di una demo finale.

* Valutazione dello stato di avanzamento
* Sviluppo del software
* Presentazione al cliente comprensiva di demo del prodotto
* Revisione e rilascio della documentazione funzionale e implementativa

2.6 Obiettivi Personali

Nella ricerca dell'attività di stage puntavo a un ambiente lavorativo in cui poter esplorare e sperimentare tecnologie che fossero di rilievo nel mercato odierno, in quanto le ritengo un fattore di crescita personale sia a livello formativo che lavorativo. Dato che la ricerca di questo tipo di tecniche di sviluppo è stata una richiesta esplicita di SITACE, mi sono sentito incentivato ad accettare la proposta da loro fatta.

Per me imparare a utilizzare una tecnologia non significa solamente saperla usare ma comprenderne i concetti alla radice e applicarla nella maniera più efficace possibile, senza quindi ricorre a schemi ripetitivi e *bad practices* dovuti a una limitata comprensione degli strumenti.

Inoltre, essendo l'esperienza svolta in un contesto aziendale, era mio desiderio poter apprendere dalle persone che poi mi avrebbero affiancato durante l'attività. Questo in quanto, avendo loro maturato più esperienza di me nel settore, possono far luce su un approccio più pragmatico ai problemi che io, al momento, potrei non avere.

Saper interagire in modo da apprendere al meglio dalle idee altrui ma rimanendo comunque in grado di criticarle apertamente, è per me un fattore importante. Questo in quanto, lavorando in gruppo, è necessario saper comunicare in maniera efficace con i membri del *team*, cosa in cui sento il bisogno di dover migliorare.

Capitolo 3

Realizzazione del Prodotto

3.1 Studio Preliminare delle Tecnologie

Come richiesto dall'azienda, ho aperto l'attività di stage con una valutazione sulle possibili tecnologie da adottare per la piattaforma di *E-Commerce Open Source* e l'ambiente di sviluppo.

3.1.1 Motivazioni sulla scelta di Prestashop

Nella ricerca di un CMS per il commercio *online*, SIATEC necessitava di una piattaforma che fosse semplice da gestire e possibilmente dotata di una buona documentazione da consultare rapidamente in caso di dubbi o necessità.

Il software gestionale *Ad Hoc Revolution* è pensato per imprese di piccole o medie dimensioni e si presta essenzialmente alla moderazione di tutti i processi dell'azienda. Questo è il motivo per cui nella scelta della piattaforma di *E-Commerce* sofisticati strumenti di gestione a lato CMS non sono effettivamente necessari.

Per le richieste fatte da SIATEC ho deciso di proporre *Prestashop* come piattaforma di *E-Commerce*, questa scelta l'ho valutata sui vantaggi che la tecnologia porta.

Prestashop è il secondo CMS per il commercio *web* più adottato in Italia, è leggero, modulare e supportato sia dagli sviluppatori ufficiali che da un'attiva comunità di utenti. Quest'ultimi producono moduli applicabili alla piattaforma. Il suo più grande vantaggio è la separazione tra contenuti, grafica e programmazione. Questo fa sì che ogni figura professionale che partecipa alla realizzazione del servizio abbia una sezione diversa sulla quale poter lavorare. Dal punto di vista dello sviluppo software, ogni componente di *Prestashop* è personalizzabile e ben documentato sulla *API* di riferimento. Un ulteriore vantaggio del sito è che, per via della sua struttura, si presta bene alle realtà di piccole e medie aziende. Questa è una funzionalità richiesta da SIATEC.

Sono invece limitate le configurazioni native sul BTB, che sono gestite spesso in maniera troppo semplicistica su alcuni aspetti. Questo problema si può ovviare adottando moduli di altri sviluppatori, anche se non è una vera soluzione al problema, che nasce dal modo in cui è strutturato *Prestashop*. Un fattore decisivo alla selezione del prodotto è stata la *API* di *Webservice* per la gestione dei dati, la quale è adatta alle richieste del problema. La piattaforma è realizzata interamente in PHP e l'azienda ha anche pubblicato una libreria nel medesimo linguaggio per un più rapido accesso al *Webservice*.

Prestashop non è stata una scelta immediata, un altro valido candidato era *Magento*, questo tuttavia risultava troppo pesante e complesso per le finalità del progetto. Il CMS selezionato non ha vincolato in ogni caso la produzione del software che ricordiamo essere voluto, da esplicita richiesta, il più generico possibile. Da qui, in un eventuale futuro, sarà sempre possibile sostituire la piattaforma *E-Commerce* con un'opportuna alternativa. Un possibile sostituto potrebbe essere *Magento*.

3.1.2 Motivazioni sulla scelta di Java

Per il progetto finale abbiamo adottato come linguaggio di sviluppo *Java*. Inizialmente avevamo considerato il *PHP* in quanto *Prestashop* fornisce una libreria già scritta per la comunicazione tramite *webservice*. Il prodotto finale richiedeva un grande *set* di funzionalità: la comunicazione *web*, l'accesso a banche dati e un punto al quale potersi interfacciare al fine di eseguire i comandi. Tutte queste componenti sono implementabili in *PHP*, tuttavia abbiamo voluto valutare in alternativa *Java*. Infatti è il linguaggio su cui è definito il dialetto sviluppato dalla Zucchetti per la linea *Ad Hoc*.

I vantaggi portati da Java sono in: sicurezza, presenza di numerosi *framework* per lo sviluppo e diverse tecnologie integrate. Tutto questo è valorizzato da un buon sistema di documentazione. Una problematica che emerge ed è relativa allo sviluppo in questo linguaggio, sono le tempistiche di sviluppo più lente, che vengono però ridotte notevolmente grazie all'uso di *Spring*.

La scelta finale è stata pilotata principalmente dalla familiarità che lo staff in SIATEC ha con il linguaggio, vista la relazione con *Ad Hoc*. La proposta di adoperare *Spring* ha infine consolidato la scelta, grazie ai vantaggi riconosciuti.

3.2 Analisi dei Requisiti

Il prodotto finale è un servizio di aggiornamento tra due banche dati distinte. In questa interazione sono stati individuati i seguenti attori:

Notificatore Attore primario del software, è colui che si interfaccia all'applicativo e richiede lo svolgimento delle attività di sincronizzazione tra i due sistemi. Osserviamo che questo soggetto non è necessariamente un essere umano.

Gestionale Attore secondario, rappresenta la banca dati del ERP che può essere il *target* o la sorgente dei dati da trasferire.

E-Commerce Attore secondario, rappresenta la banca dati del CMS il cui ruolo è analogo all'attore Gestionale.

A livello funzionale l'applicativo richiede la possibilità di aggiornare le informazioni dei due servizi. Deve essere quindi possibile selezionare su quale dei due sistemi andare a scrivere. Da questo riconosciamo la necessità di UC2:

UC2 - Selezione del *Target*.

* **DESCRIZIONE:** L'utente seleziona il target delle sue manipolazioni;
* **ATTORE PRIMARIO:** Notificatore;
* **ATTORI SECONDARI:** Nessuno ;
* **PRE-CONDIZIONE:** L'ambiente *E-Commerce* è stato creato (UC1);

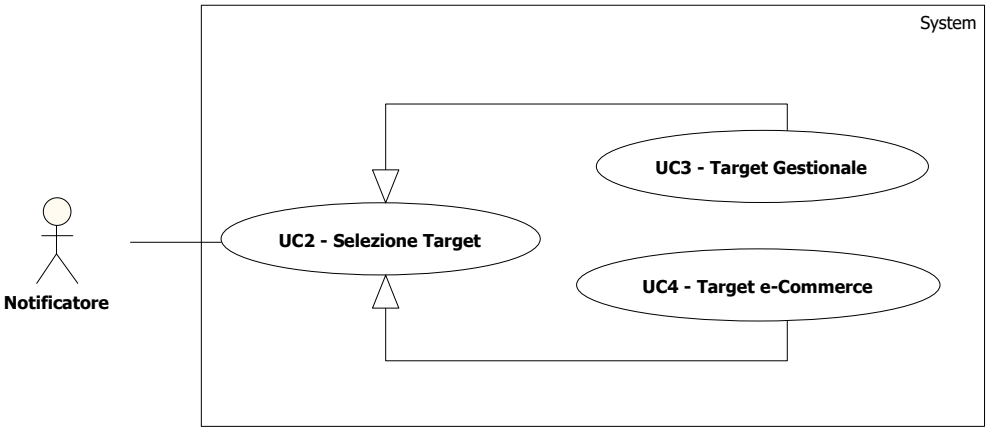


Figura 3.1: Schema UML della selezione del *target*.

* **POST-CONDIZIONE:** Viene selezionato un *target*;
* **SCENARIO PRINCIPALE:**
  1. L'utente seleziona uno dei target tra quelli resi disponibili;
* **GENERALIZZAZIONI:**
  - *Target* Gestionale (UC3);
  - *Target E-Commerce* (UC4);

L'applicativo deve eseguire procedure su elementi di dominio, in particolare:

* **Articoli:** I prodotti venduti dall'azienda.
  Essi sono d'interesse per la piattaforma di *E-Commerce* in quanto definiscono la vetrina virtuale e gli elementi di un acquisto. Eventuali modifiche effettuate sui prodotti a lato CMS dovrebbero essere corrette poiché la loro definizione e gestione è competenza del ERP.

* **Clienti:** Sono tutti i soggetti che hanno la possibilità di effettuare acquisti sulla piattaforma di *E-Commerce*.
  In primo luogo SIATEC ha richiesto il solo commercio BTB. Da questa motivazione al fine di registrare un acquisto è necessario che un acquirente sia già registrato nel sistema gestionale dell'azienda.

  Prestashop è un portale in cui è possibile effettuare modifiche al proprio profilo. Da questo fatto nasce la problematica di dover aggiornare le informazioni di un cliente sul gestionale in base alle modifiche apportate sul *E-Commerce*. Abbiamo evitato questa criticità semplicemente impedendo le azioni di modifica al cliente.

  Il prodotto è stato proposto a un'azienda che ha fatto sorgere, sebbene relativamente tardi nello sviluppo, la necessità di poter commercializzare i prodotti anche ai privati. In luce di questo problema è stata aggiunta la possibilità di registrarsi al portale. Questa interazione viene gestita in maniera del tutto analoga a un'utenza commerciale, con la sola differenza nella replicazione dei nuovi a lato gestionale.
  Anch'essi non hanno la possibilità di modificare le loro informazioni una volta presenti sullo *store online*.

* **Ordini:** L'elemento più critico della piattaforma essendo d'interesse del ERP. Un ordine è strutturato da: un indirizzo di spedizione, una lista di prodotti, l'utente a cui fa rifermento e altri dati anagrafici.

  Per il commercio BTB non è interessante la modalità di pagamento in quanto il cliente è noto a priori. Egli infatti ha già associato una modalità preferenziale. Da questa considerazione ci si può porre un problema: cosa dovrebbe fare un cliente che desidera esprimere una preferenza sul metodo di pagamento? Il dubbio è stato posto durante l'analisi a SIATEC che ha deciso di risolvere la problematica in sviluppi futuri.

  I clienti del settore privato hanno come canale di pagamento esclusivamente il bonifico bancario al momento della realizzazione del prodotto. Questa scelta è dovuta al fatto che il cliente non aveva ancora espresso una preferenza. SIATEC, in un primo momento, ha proposto *Paypal* e carte di credito.

* **Spedizioni:** Elemento meno interessante dell'analisi. Questo in quanto l'ERP gestisce le pratiche a esse relative, notificando all'occorrenza il cliente. Resta comunque desiderabile mostrare all'acquirente l'avanzamento della sua spedizione dal portale del sito.
  Dalla discussione che ho svolto con i referenti funzionali è emerso che è necessario mostrare al cliente una stima a rialzo delle tempistiche previste al fine di ricevere un prodotto.

Considerando la lista di elementi un esempio di modifica in direzione del ERP potrebbe essere il Caso d'uso per l'aggiornamento dei clienti sul *E-Commerce*:

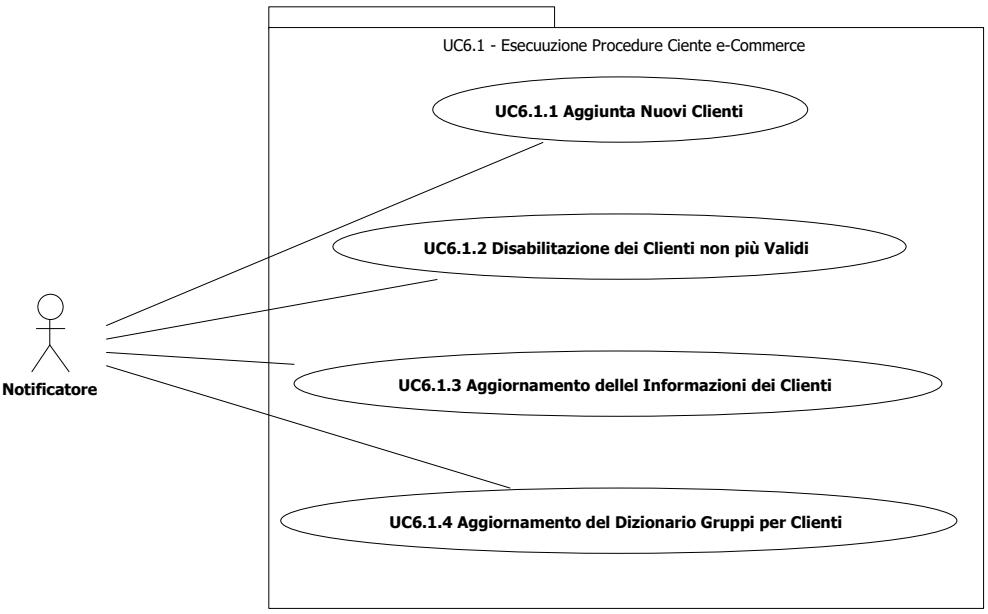


Figura 3.2: Schema UML delle operazioni disponibili per un cliente con flusso *E-Commerce*.

UC6 - Esecuzione Procedure Cliente *E-Commerce*.

* **DESCRIZIONE:** L'utente esegue le procedure che desidera tra quelle rese disponibili per l'e-Commerce, aggiornando le informazioni sui clienti.

- **ATTORE PRIMARIO:** Notificatore;
- **ATTORI SECONDARI:** Gestionale, E-Commerce;
- **PRE-CONDIZIONE:** Il target selezionato è un *E-Commerce* (UC5);
- **POST-CONDIZIONE:** : Le informazioni sui clienti e-Commerce vengono aggiornate;
- **SCENARIO PRINCIPALE:**

  1. Le info sui clienti e-Commerce vengono aggiornate;

UC6.1.2 - Disabilitazione dei Clienti non più Validi

- **DESCRIZIONE:** Vengono resi inaccessibili i Clienti non più validi sulla fonte di dati di e-Commerce da quelli della sorgente Gestionale;
- **ATTORE PRIMARIO:** Notificatore;
- **ATTORI SECONDARI:** Gestionale, E-Commerce;
- **PRE-CONDIZIONE:** Il target selezionato è un *E-Commerce* (UC5);
- **POST-CONDIZIONE:** : Vengono disabilitati i clienti obsoleti sul CMS;
- **SCENARIO PRINCIPALE:**

  1. L'utente seleziona la voce corrispondente;
  2. L'applicativo esegue la procedura e aggiorna i dati;

UC6.1.3 - Aggiornamento Informazioni dei Clienti

- **DESCRIZIONE:** Vengono aggiornate le informazioni dei clienti esistenti sulla fonte di dati di e-Commerce in base ai dati del Gestionale.

  Le informazioni di un cliente includono i gruppi di appartenenza, gli sconti specifici e le informazioni anagrafiche.
- **ATTORE PRIMARIO:** Notificatore;
- **ATTORI SECONDARI:** Gestionale, E-Commerce;
- **PRE-CONDIZIONE:** Il target selezionato è un *E-Commerce* (UC5);
- **POST-CONDIZIONE:** : Vengono sincronizzate le informazioni sui clienti del e-Commerce;
- **SCENARIO PRINCIPALE:**

  1. L'utente seleziona la voce corrispondente;
  2. L'applicativo esegue la procedura e aggiorna i dati;

I due casi d'uso accennati fanno riferimento al UC6 che è "Procedure di Aggiornamento con *Target E-Commerce*". Da queste definizioni risulta che un Notificatore può decidere di aggiornare i dati sulla piattaforma CMS selezionata come *target* solo se prima è stato generato un opportuno ambiente di *E-Commerce*. Trovandosi quindi in una situazione in cui ha individuato un punto da aggiornare, l'attore primario può decidere di svolgere le procedure adatte al target. Ad esempio sul gestionale sono state chiuse le collaborazioni con diversi clienti, perciò il Notificatore decide di disabilitare le utenze non più valide sul *sito*. Per fare questo lui avvia l'esecuzione della procedura

6.1.2. In caso non ci fossero account da disabilitare sul CMS, non ci sarebbe differenza in quanto la procedura semplicemente non causa alcuna modifica.

I casi d'uso non individuano un flusso ma punti d'interazione con l'applicativo di cui l'attore primario ignora il funzionamento. Da questo risulta evidente che l'esempio precedente non cerca di porre un senso temporale alle azioni ma di esprimere come il variare del set di condizioni permettono l'interazione con il software.

In conclusione, dall'analisi ho riconosciuto un totale di 23 requisiti da svolgere.

3.3 Progettazione

Con l'attività di progettazione era mio obiettivo la realizzazione di un software la cui architettura fosse il più possibile generica alla radice e quindi estensibile.

3.3.1 Modello e Comunicazione

La struttura principale del programma è individuata da un numeroso insieme di classi modello che non sono dotate di un vero comportamento. Ognuna di queste è un contenitore di dati, basato su *ModelComponent* (rappresentato in 3.3), da replicare. Nella codifica questo fattore si è specchiato nella definizione di *Java Beans*.
Ogni oggetto tramite ModelComponent espone in *template* il campo identificativo dell'elemento. Questo per via del fatto che nel trasferimento dei dati è necessario saper individuare univocamente un elemento. Ogni componente, inoltre, è dotato di data di creazione e di ultima modifica.

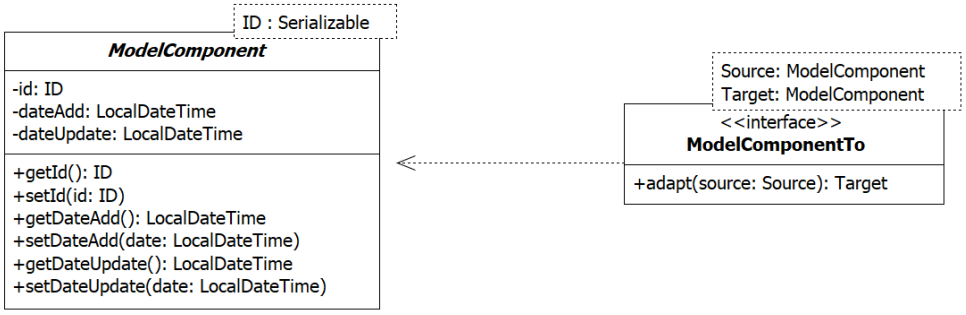


Figura 3.3: Struttura di ModelComponent e interfaccia di adattamento.

Il fine del servizio di sincronizzazione è l'invio di dati da un punto a un altro. Il formato dei quali può cambiare in base al contesto, ciò ha richiesto l'adattamento di due elementi strutturalmente diversi. Al fine di distinguere la struttura dal comportamento in questa operazione ho definito l'interfaccia *ModelComponentTo*. Essa espone un metodo per la conversione tra due oggetti alla radice *ModelComponent*, come illustrato in 3.3.

Molte parti del prodotto sono strutturate in modo da lavorare con *template*, questo è stato applicato anche alle interfacce che diventano a tutti gli effetti dei contratti puri di comportamento.

Un elemento del modello deve essere reperito da una sorgente dati, per cui ho definito dei contratti per il singolo *ModelComponent* al fine di reperirli. In particolare ci sono due livelli di astrazione nella comunicazione:

* ***DAO (Data Access Object):*** Contratto per l'accesso diretto ai dati. Esso produce un elemento che estende la classe *ModelComponent*, è il livello più basso d'interazione con la sorgente dati all'interno dell'applicativo.

* ***Repository:*** Interfaccia per l'esecuzione di operazioni sulla sorgente dati di riferimento. Questa propone un *set* di operazioni simili a quelle di un *Dao* con la differenza che astrae l'interazione a un livello più alto. Essa è per questo motivo intesa come un meccanismo d'interazione con una sorgente dati che mira all'emulazione di una collezione di oggetti. Un esempio di questo potrebbe essere per un elemento ordine che ha associati degli articoli. Utilizzando in questo caso un *Dao* per poter reperire l'ordine insieme ai prodotti a esso associati avremmo bisogno di una implementazione specifica della classe. In alternativa dovremmo usare due *Dao* distinti, uno per il reperimento dell'ordine e l'altro per gli articoli associati. Usando una*repository* invece è possibile risolvere automaticamente la relazione tra ordine e articolo. Questa operazione eventualmente può essere definita con l'utilizzo di opportuni *Dao* nella sua implementazione.

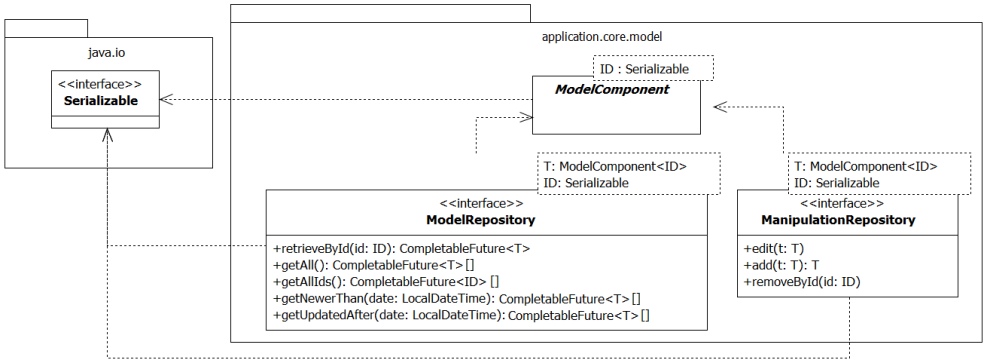


Figura 3.4: Struttura di ModelComponent e interfaccia di adattamento.

Per le *Repository* sono stati definiti due contratti che la realizzano: *ModelRepository* e *ManipulationRepository*. Questo in quanto l'esecuzione del flusso applicativo è direzionato. Non siamo mai nella situazione in cui il *software* deve aggiornare due sorgenti contemporaneamente per cui, volendo fare *separation of concerns*, sono state definite le due come illustrato in figura 3.4.

3.3.2 Operazioni e Web Server

La definizione degli elementi del modello è stata abbastanza macchinosa e di poco interesse, più notevole è invece la struttura generica che ho prodotto per l'esecuzione di procedure. In particolare era richiesta l'esecuzione di comandi di aggiornamento dei dati tra i due domini.
L'applicativo nel contesto corrente ignora la vera struttura degli oggetti di modello, in quanto la conversione avviene mediante l'implementazione della prima citata *ModelComponentTo*.

Una *Action* è un'azione programmata che in seguito della sua esecuzione ritorna uno stato, quest'ultimo è un'istanza definita dalla classe *Status*. Questo, come definito in 3.5 è un elemento comunicato al Notificatore per dargli un *feedback* su come si è svolta l'operazione.

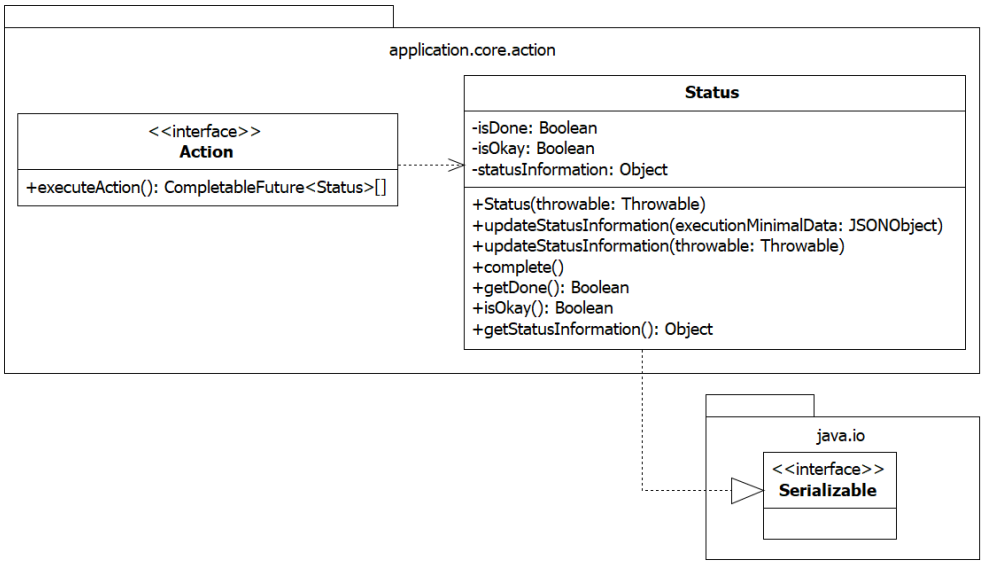


Figura 3.5: Struttura alla radice di *Action* in relazione a *Status*.

Molte azioni sono strutturalmente identiche motivo per cui ho provveduto alla definizione di classi astratte per la loro esecuzione. Ad esempio l'aggiunta e l'aggiornamento di elementi come illustrato in figura 3.6.

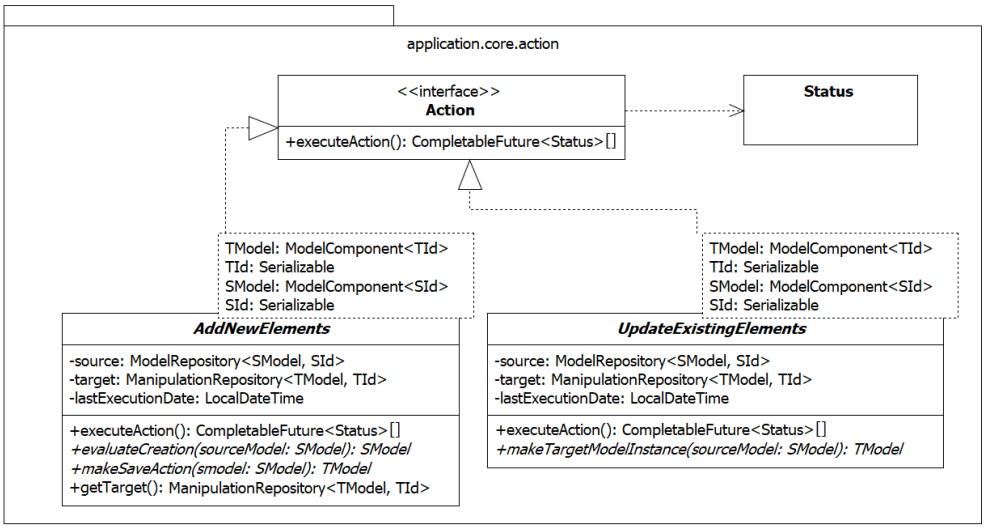


Figura 3.6: Struttura di *Action* in relazione alle sue implementazioni astratte.

Grazie alla struttura del prodotto è possibile in ogni caso definire azioni specifiche. Nasce la problematica di poter associare direttamente due elementi distinti in maniera efficace, cosa che ho risolto grazie alla definizione di *IdBound*. Essa è una classe che associa due *ModelComponent* mediante il loro identificatore, come illustrato in 3.7. Questa viene poi replicata in una banca dati d'appoggio.

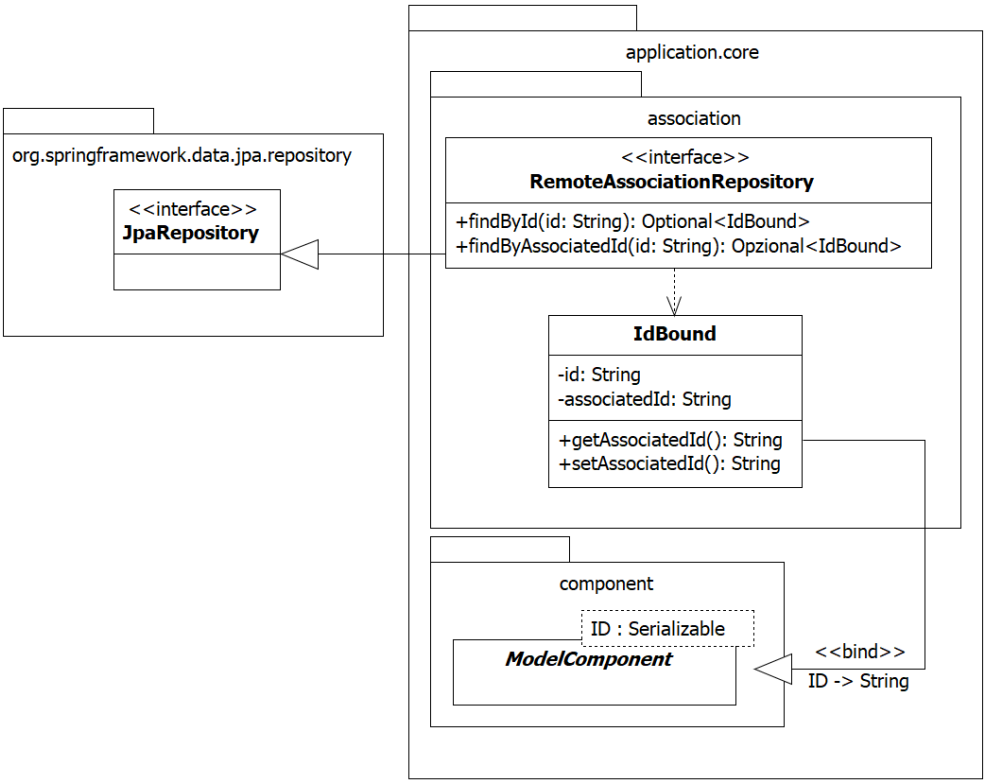


Figura 3.7: Struttura di *IdBound* e relazione ad una repository.

I punti di accesso al *software* sono definiti sulla singola risorsa che può essere aggiornata. La struttura di questi è specifica alla classe del modello, questo deriva dal fatto che *Spring* permette una facile gestione del *RestController*, come illustrato in seguito. L'idea alla base di questi elementi d'interfacciamento è che ogni metodo della classe di gestione viene mappato univocamente una richiesta *HTTP*. Essa rappresenta un'azione da svolgere e fornisce in *output* al Notificatore lo stato finale dell'esecuzione per ogni singolo trasferimento.

3.3.3 Design Pattern Utilizzati

I design pattern direttamente applicati al progetto sono stati i seguenti:

* ***Dependency Injection (DI)*** Pattern della OOP per risolvere le dipendenze al fine di migliorare la testabilità del software e delegare la costruzione di oggetti a un livello più alto. Questo si realizza facendo differenza su quelle che sono le responsabilità della classe che ha la dipendenza da quella che ha il compito di gestirne la creazione.

  In questo *pattern* un sostanziale vantaggio che si ha, oltre alla *separation of concerns*, è una riduzione delle dipendenze all'interno delle diverse classi.
  La *DI* quindi favorisce la stesura d'interfacce per slegare ulteriormente la dipendenza tra oggetti che possono eseguire la *Injection* via costruttore o *setter*.

* ***Template Method*** Pattern ampiamente utilizzato nella stesura delle *Action*. Il pattern prevede la definizione di una classe algoritmo il cui comportamento generale è strutturato ma alcune parti sono da implementare.

* ***Data Access Object (DAO)*** Pattern architetturale che permette la delega, l'accesso e la costruzione di oggetti a delle classi realizzate a tal fine, come descritto in precedenza.

3.4 Codifica

3.4.1 Accesso ai Dati a Prestashop e Sql Server

Un elemento interessante dello sviluppo del software è come ho gestito la comunicazione con i due servizi distinti. L'accesso al ERP è in rete locale e diretto alla banca dati *Microsoft Sql Server* mentre per *Prestashop* è realizzato con l'uso dell'interfaccia *webservice* esposta.

Prestashop Per la comunicazione con *Prestashop* ho fatto uso del *RestTemplate* fornito da Spring per l'esecuzione di richieste *HTTP*.
Il *webservice* della piattaforma espone i propri dati in lettura in*XML* e *Json* mentre è solamente possibile mandare in scrittura oggetti *XML*. In particolare per la comunicazione con *Prestashop* ho implementato le classi del servizio dalla definizione di un *Dao* generico, *PrestashopTDao*.

Il trasferimento dei dati è reso possibile dal *RestTemplate*, definito da *Spring*, che permette una rapida costruzione e invio di richieste *HTTP*. Gli oggetti di ritorno in questi scambi vengono poi opportunamente convertiti grazie alle annotazioni di *javax.xml* sulle classi *target*. Per meglio comprendere facciamo un esempio seguendo 3.8.
Per la natura del servizio, che è in rete, avviene in primo luogo la costruzione di una richiesta mediante l'uso del *RestTemplate*, la cui risposta è poi una stringa *XML*.
Da questo documento è possibile, grazie alle annotazioni sulla classe dell'elemento da costruire generare un oggetto *Java* mediante l'uso di *JAXB*.

Microsoft SQL Server L'interazione con la banca dati in questione è stata semplice grazie al fatto che ho utilizzato *JDBC*, un connettore a *database*. Il *driver* è anche la base su cui è sviluppato il modulo *Spring Data JPA*, il quale permette l'esecuzione di richieste alla banca dati mediante la definizione d'interfacce dotate di annotazioni. Queste *Query* possono essere realizzate in due modi distinti che illustriamo facendo rifermento a 3.9, in cui mostriamo ad esempio alcune procedure per l'entità Product.

Il primo metodo descritto è realizzato con la definizione della stringa da valutare, direttamente annotata sulla sua firma con il tag *@Query*. Nelle altre due rappresentazioni si sfrutta la possibilità di definire, seguendo una rigida sintassi, delle stringhe autogenerate che danno un valore semantico al metodo. In questo caso *findBy* è chiave che va a cercare tutti gli elementi per cui la condizione successiva della scrittura è valida.

3.4.2 Dependency Injection e classi di Configurazione in Spring

Avendo utilizzato *Spring* nella produzione del software, sono riuscito a slegare la logica del programma dal suo contesto. Questo mi ha anche persmsso di risolvere le

```
@Override
@Async
public CompletableFuture<T> get(ID id) {
    return CompletableFuture.supplyAsync(() -> {
        try {
            UriComponents uri = UriComponentsBuilder.fromHttpUrl(resourcePath + "/{target}/{id}/")
                    .buildAndExpand(targetAsPlural, id.toString());

            ResponseEntity<String> responseEntity = restTemplate.getForEntity(uri.toUri(), String.class);
            if (!responseEntity.getStatusCode().is2xxSuccessful() || responseEntity.getBody() == null) return null;

            Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();

            /* Intero documento convertito. (Non efficiente ma non importa) */
            Document document = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(
                    new InputSource(new StringReader(responseEntity.getBody()))
            );

            return unmarshaller.unmarshal(document.getElementsByTagName(targetResource).item( index: 0));
        } catch (Exception e) { return CompletableFuture.failedFuture(e); }
    }).thenApply(o -> (getModelClass().cast(o)));
}
```

Figura 3.8: Estratto codice *get* della classe astratta PrestashopDao per il riperimento di un oggetto dal suo identificatore univoco.

```
@Repository
public interface ProductEntityRepository extends CrudRepository<ProductEntity, String>,
        ModelRepository<ProductEntity, String>, ManipulationRepository<ProductEntity, String> {

    @Async @Query(value = "select entity.id from ProductEntity entity")
    CompletableFuture<List<String>> getAllIds();

    @Async CompletableFuture<List<ProductEntity>> findByDateAddAfter(LocalDateTime dateTime);
    @Async CompletableFuture<List<ProductEntity>> findByDateUpdAfter(LocalDateTime dateTime);
}
```

Figura 3.9: Estratto codice Get della classe astratta PrestashopDao.

dipendenze mediante *Dependency Injection*. Il risultato di questa operazione è stata la definizione di classi di *Configurazione* per i *Component*. Essi sono *Bean* a livello di classe, e quindi necessariamente mappabili dallo *IoC Container*.
Una classe di configurazione è quella in immagine 3.10 dove vengono definiti campi utili alla costruzione del componente *ProductXmlDao*.

In particolare nell'immagine sono definiti i campi necessari alla comunicazione mediante la *REST API* esposta da *Prestashop*. In questi elementi è presente una configurazione preliminare della struttura delle richieste *HTTP*, dove poniamo una chiave di accesso.

La risoluzione delle dipendenze dei *Beans* di *Spring* avviene sul costruttore di una classe *Component*. In occorrenza di un campo di una classe mappabile a più elementi omonimi è necessario che essa definisca mediante l'annotazione *@Qualifier* quale *Bean* riferire. Esso avrà un suo name univoco impostato. Questo si può vedere per *JAXBContext* in immagine 3.11 che è definito per ogni implementazione di *PrestashopDao*. Data questa motivazione il *@Qualifier* fa rifermento alla corretta istanza dalla configurazione, che è quella in figura 3.10.

```java
@Configuration
public class ProductConfiguration {

    @Bean public String productTargetResource() { return "product"; }
    @Bean public String productPlural() { return "products"; }

    /**
     * The context to retrieve data and send via REST requests.
     * @return Instance for marshalling / unmarshalling of objects of type ProductXml
     * */
    @Bean(name = "productJaxb")
    public JAXBContext jaxbContext() throws JAXBException {
        return JAXBContext.newInstance(ProductXml.class);
    }

    /**
     * Creates a valid RestTemplate to access the resource by mapping the correct authorization key.
     * @param restTemplateBuilder Object to build the RestTemplate, it also is a Bean.
     * @return a RestTemplate that enables the communication with the product resource.
     */
    @Bean(name = "productRest")
    public RestTemplate restTemplate(RestTemplateBuilder restTemplateBuilder) {
        RestTemplate restTemplate = restTemplateBuilder.build();
        restTemplate.getInterceptors().add((httpRequest, bytes, clientHttpRequestExecution) -> {
            httpRequest.getHeaders().set("Authorization", "Basic SVk4UkU3RklMN0dOUTk0WkhCQ0hDMjhUS0U5UURKVFQ
            return clientHttpRequestExecution.execute(httpRequest, bytes);
        });
        return restTemplate;
    }
}
```

Figura 3.10: Classe di configurazione per ProductXmlDao

```java
@Component
@Qualifier("productXmlDao")
@EnableAsync
public class ProductXmlDao extends PrestashopTDao<ProductXml, Long> {
    public ProductXmlDao(String resourcePath, String productTargetResource, String productPlural,
                         @Qualifier("productRest") RestTemplate restTemplate,
                         @Qualifier("productJaxb") JAXBContext jaxbContext) {
        super(resourcePath, productTargetResource, productPlural, restTemplate, jaxbContext);
    }

    @Override public Class<ProductXml> getModelClass() { return ProductXml.class; }
    @Override public Long stringToIdConverter(String s) { return Long.valueOf(s); }
}
```

Figura 3.11: Classe di definizione di ProductXmlDao

3.4.3 Software dall'esecuzione in parallelo su Spring

Volendo SIATEC sperimentare nuove tecnologie e metodi di sviluppo, ho proposto all'azienda la realizzazione di un software asincrono. Questa proposta è nata per un mio interesse personale e non da una necessità pratica dal punto di vista della realizzazione del *software*.

Lo sviluppo di un prodotto asincrono è molto semplice in *Java* grazie alle tecnologie integrate. Un esempio di questo è la classe *CompletableFuture* che al termine

HTTP Request Type	*org.springframework.web.bind.annotation*
GET	@GetMapping
POST	@PostMapping
PUT	@PutMapping
DELETE	@DeleteMapping
PATCH	@PatchMapping

Tabella 3.1: Tabella di associazione tra richiesta *HTTP* e annotazione *Spring*

dell'esecuzione di codice asincrono da al suo oggetto di definizione un *Object* o un *Throwable*. A questa complessità si combina la struttura di *Spring* che permette una gestione automatica dei *Thread* e quindi riduce la mole di lavoro dello sviluppatore.

L'approccio asincrono alla programmazione non si è prestato bene al contesto del progetto, in quanto il prodotto finale appartiene a una realtà aziendale in cui il servizio non richiede che sia ottimizzato. Da questa motivazione l'uso dell'asincronicità nel servizio è stato più un esercizio didattico che una necessità del prodotto. SIATEC nonostante questo ha approvato l'approccio in quanto aveva interesse nell'applicare questi strumenti in possibili progetti futuri.

Al termine dello stage ho comunque provveduto a fornire delle interfacce non asincrone del prodotto in caso SIATEC volesse in futuro percorrere una strada più pragmatica.

3.4.4 RESTful Webservice per la ricezione di comandi

La produzione di un punto di acceso per l'avvio delle procedure di aggiornamento è stata la parte del software che ha messo più in risalto le potenzialità di *Spring*. Questo infatti si vede dalla semplice configurazione per creare un percorso di mappatura come da immagine 3.12. Dalla figura possiamo capire la facilità con cui si può mappare un comando all'avvio di una procedura.

Sono disponibili comandi per diversi tipi di richiesta *HTTP*, l'annotazione *@GetMapping* ad esempio lega l'indirizzo definito a una richiesta di tipo *GET*.

In tabella 3.1 sono riportati tutti i comandi possibili.

In caso non si voglia adottare una di queste tipologie di richiesta si può far uso del generico *@RequestMapping* nel quale si può esplicitare il metodo in un campo *method*. In generale quest'ultima annotazione è alla base della gestione dei percorsi di un *RestController*.

```
@GetMapping("/addNewProducts")
public ResponseEntity<List<Status>> addAllNewProducts() {
    try {

        var list : List<Status> = this.addAllNewProductsPrestashop.executeAction().get();
        return ResponseEntity.status(HttpStatus.OK).body(list);

    } catch (InterruptedException | ExecutionException e) { e.printStackTrace(); }
    return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body( t: null);
}
```

Figura 3.12: Funzione di mappatura di un *@RestController* per gli articoli

Ogni classe che definisce un *RestController* è gestito da *Spring* come se fosse un componente e quindi un *Bean*. Esso viene automaticamente associato al servizio web dato dall'avvio di *Spring Boot* e perciò pronto all'uso. Nel progetto ho definito un *RestController* per ogni elemento di dominio da trasferire.
In questo tutte le *Action* relative al modello che rappresenta sono parametro di costruzione del *Controller*. Ognuno di questi parametri è risolto da *Spring* in quanto il *RestController* è a tutti gli effetti un *@Component*.

3.5 Verifica e Validazione

In SIATEC il processo di verifica è molto approssimativo in quanto essa non vede la necessità d'investire tempo nel processo, motivo per cui la produzione del software non presenta test di unità o d'integrazione.

La ragione che spinge l'azienda ad adottare questo approccio alla verifica è dato dal fatto che lavora molto su personalizzazioni per *Ad Hoc*. Lo sviluppo di questi *software* avviene in un ambiente robusto e controllato, in quanto il prodotto finale è un modulo che viene applicato a un ERP funzionante.

Questa mancanza, nel progetto, viene attenuata dalla struttura del prodotto. Dalla progettazione infatti molte classi agiscono come semplici *placeholder* di dati e quindi non hanno un vero comportamento da verificare.
Molte procedure sono anch'esse basilari essendo alla radice delle assegnazioni.
Resta tuttavia necessario per l'azienda, a mio parere, introdurre un sistema di produzione di test che sia coerente con le loro abitudini attuali.

Nonostante la mancanza di verifica software dinamica, l'analisi statica è stata garantita dall'ambiente di sviluppo. Questo infatti, come per gran parte degli editor odierni, esegue questo processo automaticamente notificando lo sviluppatore.

SIATEC prevede per i suoi prodotti un processo di validazione dei requisiti per valutare la correttezza del software. Questo è stato effettuato sul progetto in concordanza ai requisiti tracciati dall'analisi sia in fase di sviluppo che a un collaudo del prodotto.

3.6 Prodotto Risultante

Il prodotto risultante è un servizio localizzato nella rete locale del software gestionale, che espone degli indirizzi per l'esecuzione delle procedure.
Il vero potenziale del prodotto si ha continuandone lo sviluppo per poter definire sempre più elementi utili alla generazione della piattaforma di *E-Commerce*.

Il *software* non è completo al momento del termine dello stage, in quanto non sono riuscito a raggiungere tutti i requisiti prefissati. Questo perché che gran parte della complessità nel realizzare il prodotto è stata nello studio delle due rappresentazioni distinte per un elemento di *business*.

L'immagine 3.13 mostra come il prodotto può essere impiegato in un'infrastruttura. Il software si trova in rete con la banca dati gestionale e comunica con *Prestashop* mediante i servizi *webservice* che questo fornisce. Avendo questi punti di accesso il software ha modo di moderare correttamente il trasferimento dei dati richiesti dalle operazioni che mette a disposizione.
In questo modo le modifiche che il prodotto effettua sui due *database* si riflettono su tutti i software che vi accedono.

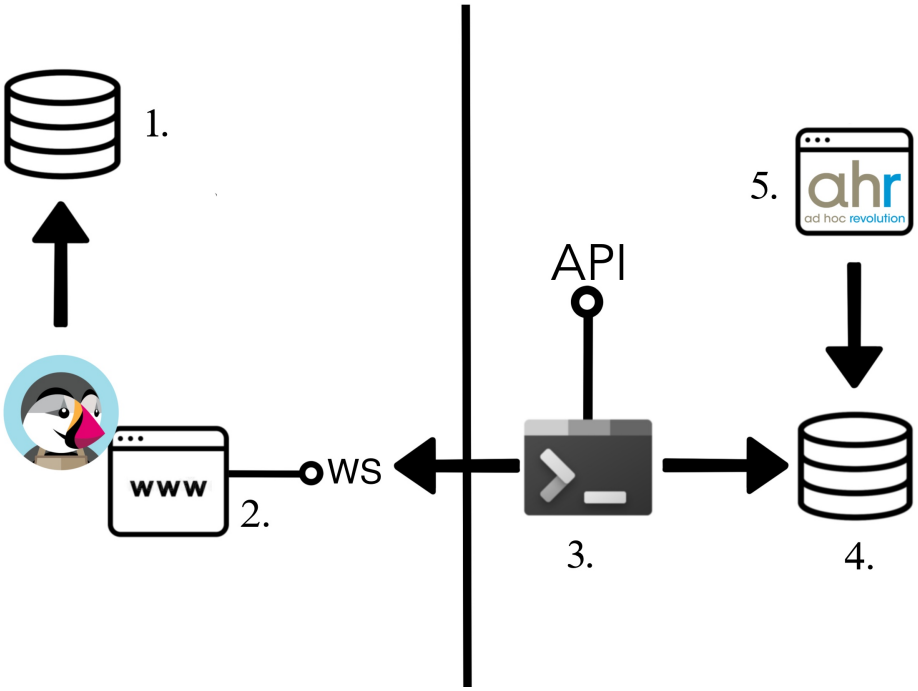


Figura 3.13: Il flusso di rete possibile dell'applicativo.

> Il prodotto (3) comunica direttamente con la banca dati gestionale (4). Questo *database* è l'infrastruttura su cui si basa *Ad Hoc Revolution* (5). L'applicativo prende i dati importanti al sito di *E-Commerce* e dopo averli adattati chiama il *webservice* di *Prestastashop* (2) in maniera opportuna. Le modifiche vengono salvate sulla banca dati del CMS (1) che mostra i contenuti aggiornati alterando così la visualizzazione del sito.
>
> Il procedimento può avvenire anche in direzione opposta su determinati elementi del dominio; i nuovi ordini effettuati sulla piattaforma di *E-Commerce* ad esempio vengono replicati tra quelli non evasi della banca dati gestionale.

Il servizio espone inoltre un gruppo d'indirizzi a cui poter richiedere l'esecuzione delle procedure integrabili da altri applicativi. Questo, ad esempio, potrebbe in futuro permettere che le procedure siano direttamente eseguibili dall'interfaccia di *Ad Hoc Revolution* mediante una personalizzazione apposita.

Capitolo 4

Valutazione retrospettiva sulle attività

4.1 Raggiungimento degli Obiettivi

4.1.1 Obiettivi Aziendali

Ho concluso l'attività di stage senza soddisfare tutti i requisiti concordati. Questo è dovuto al fatto che ho riscontrato una grande complessità nel riconoscere le relazioni tra gli oggetti dei due diversi domini. In particolare ho dovuto studiare quali sono le informazioni effettivamente da trasferire e come queste sono codificate nelle due banche dati. Un'ulteriore causa all'incompletezza del prodotto è dovuta alle richieste dell'azienda a cui abbiamo proposto il *software* finale. Essa desiderava infatti adattare l'uso della piattaforma in modo da essere accessibile anche da clienti privati. L'introduzione della funzionalità ha comportato un ulteriore analisi di come introdurre il concetto nel prodotto che era in fase già di sviluppo.

Tabella 4.1: Tabella dei Requisiti Funzionali effettivamente soddisfatti

Requisito	Descrizione	Importanza
RFO1	Il notificatore deve poter avviare la creazione di ambiente di *E-Commerce* dai dati del sistema ERP	Obbligatorio
RFO2	Il notificatore deve poter selezionare un target tra quelli resi disponibili per l'esecuzione delle procedure	Obbligatorio
RFO2.1	Il notificatore deve poter selezionare il sistema gestionale come target di sincronizzazione	Obbligatorio
RFO2.2	Il notificatore deve poter selezionare l'*E-Commerce* come target di sincronizzazione	Obbligatorio
RFO3	Il notificatore deve poter aggiornare le informazioni del Cliente del *E-Commerce* dai dati del gestionale	Obbligatorio
RFO3.1	Il notificatore deve poter aggiungere i nuovi clienti	Obbligatorio

RFD3.2	Il notificatore deve poter aggiornare i dati dei clienti	Desiderabile
RFO3.3	Il notificatore deve poter disabilitare i clienti invalidi	Obbligatorio
RFD3.4	Il notificatore deve poter aggiornare il dizionario dei gruppi (categorie in cui si distinguono i diversi clienti)	Desiderabile
RFO4	Il notificatore deve poter aggiornare le informazioni degli ordini su ERP dai dati di *E-Commerce*	Obbligatorio
RFO4.1	Il notificatore deve poter replicare i nuovi ordini	Obbligatorio
RFO5	Il notificatore deve poter aggiornare le informazioni degli ordini su *E-Commerce* dai dati del gestionale	Obbligatorio
RFO5.1	Il notificatore deve poter aggiornare lo stato assunto dagli ordini in elaborazione	Obbligatorio
RFD5.2	Il notificatore deve poter aggiornare i dati anagrafici degli ordini esistenti	Desiderabile
RFD5.3	Il notificatore deve poter aggiornare il dizionario degli stati assumibili dagli ordini	Desiderabile
RFO6	Il notificatore deve poter aggiornare le informazioni degli articoli su *E-Commerce* dai dati del gestionale	Obbligatorio
RFO6.1	Il notificatore deve poter aggiungere gli articoli mancanti sulla banca dati *target*	Obbligatorio
RFO6.2	Il notificatore deve poter aggiornare i dati alla più recente versione i anagrafici degli articoli	Obbligatorio
RFO6.2	Il notificatore deve poter aggiornare i dati alla più recente versione i anagrafici degli articoli	Obbligatorio
RFO7	Il notificatore deve poter aggiornare le informazioni del Cliente del gestionale dai dati del *E-Commerce*	Obbligatorio
RFO7.1	Il notificatore deve poter aggiungere i nuovi clienti	Obbligatorio

I requisiti che non ho soddisfatto sono tutti quelli relativi alle spedizioni. Questi non erano prioritari al fine del progetto in quanto il sistema gestionale è responsabile di notificare i clienti ad ogni avanzamento dei loro acquisti. Da questa motivazione il risultato del progetto è comunque stato in linea con quanto desiderato da SIATEC. L'altro unico requisito che non ho raggiunto associava i prodotti a delle categorie. Esso era previsto in fase di progettazione, tuttavia il cliente ha espresso di non voler sfruttare la funzionalità, motivo per cui l'abbiamo messa in secondo piano.

SIATEC era comunque soddisfatta del prodotto risultante che sarà ulteriormente sviluppato, anche al termine dell'attività di stage, al fine di essere commercializzato.

4.1.2 Obiettivi Personali

Nella sezione 2.6 ho presentato degli obiettivi personali che mi ero posto per lo stage. Questi, per certi aspetti, sono riuscito a raggiungerli, in particolare:

Studio di Tecnologie. Un mio obiettivo era lo studio di nuove tecnologie. Questo l'ho fatto in maniera solo sufficiente, in quanto non ritengo di aver appreso le tecniche al livello di dettaglio che mi ero prefissato. Tale considerazione è dovuta al fatto che le tecnologie che ho usato sono molto complesse ed è quindi irrealistico pensare di poterle apprendere rigorosamente in soli due mesi.
Reputo comunque di aver applicato gli strumenti a mia disposizione con criterio anche se, in certe situazioni, avrei potuto svolgere un lavoro migliore.

Apprendimento e Interazione con un *Team*. Un mio ulteriore desiderio con l'attività di stage era quello di apprendere con criterio dalle persone che mi hanno affiancato durante il corso del lavoro. Grazie alla collaborazione di altri gruppi aziendali ho avuto molti spunti sia per la strutturazione del software che la sua implementazione. Internamente all'azienda ho invece avuto meno *feedback* sullo sviluppo, in quanto SIATEC non aveva ancora utilizzato queste tecnologie in passato.
Ho in ogni caso fatto tesoro dei consigli a me dati che mi hanno spesso aiutato a considerare il problema da più angolazioni diverse.

Sul fronte della comunicazione in una squadra reputo di aver migliorato il mio approccio, ho infatti iniziato a prendere i tempi che reputo necessari al fine di esprimere in maniera semplice i concetti che voglio dire.

4.2 Conoscenze Acquisite

Durante lo svolgimento del progetto ho avuto modo di apprendere molto su diverse tecnologie e su meccanismi interni di un'attività commerciale. Dalla produzione del servizio, ho infatti avuto di farmi insegnare molto su quelli che sono i processi di un'azienda dal punto di vista contabile. Non sono riuscito tuttavia ad assimilare tutto ciò che mi è stato spiegato, in quanto la gestione delle aziende è una tematica molto complessa. Reputo infatti necessario dover studiare approfonditamente questa attività al fine comprenderla davvero.

Con lo stage ho anche avuto modo di vedere come a tutto esiste un approccio pragmatico. Questo mi è stato fatto notare in particolare in fase di analisi dai miei referenti funzionali. Essi mi hanno infatti mostrato come anche nell'uso di software gestionali si prendono delle "scorciatoie". Molti clienti di SIATEC infatti fanno un uso improprio di alcune funzionalità dei prodotti della linea *Ad Hoc*. L'errore per molti è intenzionale, in quanto il software fornisce spesso una gestione dei processi troppo rigida e complessa rispetto a quella che è la vera realtà aziendale.

Importante dal punto di visto della mia formazione è stato l'apprendimento di nuove tecnologie. Questo è stato infatti il mio primo progetto *Java*, il linguaggio mi era comunque familiare in quanto l'avevo visto in precedenza nel contesto accademico. Esso dispone di un ambiente di sviluppo molto simile a quello di altri linguaggi di programmazione ad oggetti, per cui non ho avuto grosse difficoltà ad ambientarmi.

Ho scoperto con lo sviluppo che *Java* è uno strumento notevolmente articolato e fornisce già da solo molti strumenti utili al programmatore. L'altra notevole tecnologia utilizzata nello sviluppo è stata *Spring*. Questa mi ha incentivato a gestire le dipendenze con la *Dependency Injection* e mi ha fornito numerosi strumenti che hanno facilitato

lo sviluppo. Da quest'esperienza ho deciso che continuerò a studiare *Spring* anche in futuro, in quanto desidero approfondire la tecnologia e imparare ad utilizzarla al meglio.

4.3 Considerazioni Finali

Lo stage che ho svolto in SIATEC è stato per me un'esperienza positiva.

Mi rendo conto, ora che ho terminato il progetto, che il prodotto poteva essere realizzato in maniera diversa e, sicuramente, più corretta. In particolare ritengo di aver approcciato in maniera troppo rigida la definizione degli elementi di dominio dove ho forse forzato l'uso dell'ereditarietà. In alternativa avrei potuto optare per un approccio più furbo, come ad esempio incapsulando i dati di trasferimento in oggetti *JSON* o *XML* dalla struttura più elastica in modo da focalizzare l'esecuzione del *software* interamente nella conversione. Non sminuisco però il lavoro svolto in due mesi che si è basato su uno studio eseguito con criterio, ma ritengo i miei errori uno spunto di riflessione per progetti futuri.

Ripensandoci, la proposta di sviluppare un prodotto la cui esecuzione non è sempre sequenziale ma dotata di momenti di parallelismo, come discusso in 3.4.3, è stato un errore. Questo poiché il dominio del problema non era il più adeguato alla decisione. Sono comunque soddisfatto dell'aver potuto sperimentare queste tecnologie, che ho visto solo di sfuggita nel mio corso di studi.

Quanto appreso in questa esperienza mi ha aiutato a comprendere che in ogni attività sbagliare è parte del processo di miglioramento personale. Non ritengo ci sarà mai una situazione in cui potrò essere convito di aver svolto un lavoro perfetto. Da questa motivazione l'unica cosa che a mio parere si può fare è autocritica e cercare di riconoscere gli errori commessi, in modo da imparare da essi.

Bibliografia

Siti web consultati

APACHE LICENSE, VERSION 2.0. URL: https://www.apache.org/licenses/LICENSE-2.0.

Documentazione su Prestashop. URL: https://devdocs.prestashop.com.

Dominio di definizione Scrum. URL: https://www.scrum.org/.

GNU General Public License, Version 2. URL: https://www.gnu.org/licenses/old-licenses/gpl-2.0.html.

Manifesto Agile. URL: http://agilemanifesto.org/iso/it/.

Panoramica su Prestashop. URL: https://www.prestashop.com/it.

Team, Spring Development. *Spring Framework Documentation*. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/.

Acronimi

ERP Enterprise Resource Planning. 1, 6, 9, 10, 11, 12, 13, 16, 17, 18, 31, 32

CMS Content Management System. 4, 9, 10, 11, 12, 15, 16, 17, 19, 20, 29

VM *Macchina Virtuale (Virtual Machine)*. 4

GPL 2.0 General Public License, Version 2.
Per maggiori informazioni consultare Bibliografia. 4, 39

IoC *Inversion of Control*.
Inversione del flusso di controllo. 5, 6

DBMS *Database Management System*. 6, 12

IDE *Integrated development environment*.
Ambiente di Sviluppo Integrato. 6

BTB *Business to Business*. 9, 15, 17, 18

UML Unified Modeling Language. 12, 39

OOP *Object Oriented Programming (Programmazione ad Oggetti)*. 23

Glossario

Caso d'uso Tecnica per la definizione rigorosa dell'interazione di un utente con un sistema rappresentabili in UML mediante *Use Case Diagram*, il diagramma non è un sostituto della definizione ma un ausilio . 2, 12, 18, 19, 20

User Story Tecnica per definire dal punto di vista dell'utente il risultato di una sua interazione con il sistema senza individuarne dei veri vincoli . 2

Scrum Framework di sviluppo Agile creato da Ken Schwaber e Jeff Sutherland che si basa sulla creazione di un ricco *Product Backlog* dal dominio del problema da cui si pianificano incrementi che rilasciano un potenziale prodotto, chiamati *Sprint* dalla durata fissata da qualche giorno fino ad un mese . 2, 13

Git Sistema di controllo versione *Open Source* progettato per la gestione di progetti . 3, 7

E-Commerce Nel dominio del documento attuale è uno store di rivendita virtuale. 4, 5, 9, 11, 12, 15, 16, 17, 18, 19, 28, 29, 31, 32

Back Office Sezione di un CMS dedicata alla gestione del sito da un amministratore dell'attività commerciale con diversi strumenti . 4

Open JDK 14 Implementazione Open Source dell'ambiente di sviluppo della piattaforma Java disponibile al pubblico dal 17 Marzo 2020 a licenza GPL 2.0. 4

Vanilla Software realizzato senza l'uso di framework o librerie, quindi solamente creato dagli strumenti forniti dall'ambiente di sviluppo e dal linguaggio . 5

***Enterprise Resource Planning* (ERP)** Software di gestione che integra tutti i processi di *business* di un'azienda come ad esempio la gestione del magazzino, delle vendite e dalla contabilità in un solo sistema, si appoggia ad una banca dati unica. Per comodità riferito anche come "sistema gestionale" . 37

Content Management System Software su server web che facilità la creazione e gestione dei contenuti per un sito . 37, 39

Macchina Virtuale Software che emula un ambiente di esecuzione. 37

Business to Business Attività la cui clientela non è del settore privato. 37

UML Linguaggio di modellazione per la progettazione di software basati sul paradigma ad oggetti che definisce standard di rappresentazione utili ad un progetto . 37