Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Un plugin Maven per l'automatizzazione della pubblicazione di documentazione software

Tesi di laurea triennale

Relatore
Prof.Paolo Baldan

Laureanda
Laura Cameran

Anno Accademico 2018-2019

"I made a discovery today. I found a computer.
Wait a second, this is cool. It does what I want it to.
If it makes a mistake, it's because I screwed it up. Not because it doesn't like me."

— The Mentor

Sommario

Il documento corrente descrive il lavoro svolto durante il periodo di stage, della durata di trecentoventi ore, dalla laureanda Laura Cameran presso l'azienda Finantix Pro Unipersonale S.r.l.
L'obiettivo principale da raggiungere era lo sviluppo di un plugin Maven al fine di automatizzare la pubblicazione di documentazione di software sul sistema documentale Atlassian Confluence. Per realizzare tale compito era richiesto inoltre lo studio di API RESTful, mezzo con cui interagire con il sistema aziendale.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Il progetto

Finantix è un'azienda di informatica che vende prodotti software. Il suo prodotto principale è suddiviso in moduli. Ognuno di questi moduli prevede una propria documentazione delle API Java (un archivio zip contente documentazione in formato Javadoc) e la documentazione della API RESTful (un archivio zip contenente documentazione in formato Open API). Questa documentazione viene manualmente caricata sulla piattaforma Confluence, ove cui è consultata dagli sviluppatori dell'azienda.

Il plugin Maven nasce dalla necessità di automatizzare la pubblicazione di questa documentazione su Confluence, in modo da semplificare e velocizzare notevolmente questo processo. Infatti, una volta configurato correttamente il plugin in tutti i progetti relativi ai moduli software, il caricamento avviene direttamente durante la build dei progetti, senza richiedere ulteriore intervento umano.

1.2 Principali problematiche

Durante il corso dello stage non sono stati riscontrati rilevanti problemi che hanno particolarmente influito sull'attività. Nonostante ciò, un problema non banale che è stato affrontato riguarda la documentazione di Maven. Molte pagine relative alla documentazione di plugin Maven infatti, risultano obsolete perché poco aggiornate. Per far fronte a questo problema, un confronto diretto e costante con gli sviluppatori senior del team DevOps, esperti della tecnologia, è stato il metodo di risoluzione determinante.

1.3 Strumenti utilizzati

Gli strumenti adottati per la creazione del plugin Maven sono molteplici. Alcune di questi sono abitualmente adoperati da tutti gli sviluppatori dell'azienda, motivo per cui sono stati utilizzati anche per questo progetto, mentre altri sono stati liberamente scelti dalla candidata. Tra essi, scelti selezionati per i vari motivi sotto elencati, troviamo:

* **GitKraken**: client di Git che presenta un'interfaccia grafica molto intuitiva e interattiva, oltre che semplice da usare
* **JUnit**: framework per i test d'unità che si integra facilmente con Eclipse e Maven

* **Visual Studio Code**: editor di codice che supporta molti linguaggi, tra cui JSON e HTML
* **SequenceDiagram.org**: strumento online che permette la creazione di diagrammi di sequenza in modo semplice e veloce grazie ad una sintassi propria
* **ObjectAid UML Explorer**: strumento d'integrazione a Eclipse che permette la creazione automatica di diagrammi delle classi a partire dal codice Java
* **Meecrowave**: framework consigliato dall'azienda (ma non imposto) che permette la creazione di server velocemente

1.3.1 Riepilogo degli strumenti

Qui di seguito viene riportata una tabella che riassume tutti gli strumenti utilizzati e a quale scopo.

Strumento	Scopo
Eclipse	Ambiente di sviluppo
Maven	Build automation per la gestione di progetti
Confluence	Pubblicazione, creazione e consultazione di documentazione
Jira	Issue tracking system
Jenkins	Continuous integration
Sonarqube	Analisi statica del codice
Bitbucket e GitKraken	Controllo di versione
JUnit	Test di unità
Visual Studio Code	Editor di codice
SequenceDiagram.org	Creazione dei diagrammi di sequenza
ObjectAid UML Explorer	Creazione dei diagrammi delle classi
Meecrowave	Creazione di server

Tabella 1.1: Tabella di tecnologie utilizzate durante il progetto e loro scopo.

1.4 Il prodotto ottenuto

Il plugin Maven riesce a realizzare il caricamento di documentazione grazie ad un plugin di terze parti su Confluence, chiamato *Docs*. Per poter comprendere appieno il funzionamento del plugin Maven, è prima necessario fare luce sul plugin Confluence. Come è possibile vedere dall'immagine sottostante, *Docs* è suddiviso in categorie (in questo caso "JavaDocs" e "Spacification"). Le categorie presentano un nome e hanno il semplice scopo di dare un ordine per gruppi della documentazione. Ogni categoria infatti contiene al suo interno delle pagine web (chiamate anche *doc*) che includono la

documentazione in formato HTML (come per esempio “JDF Specification RC2” per la categoria “Specification”).

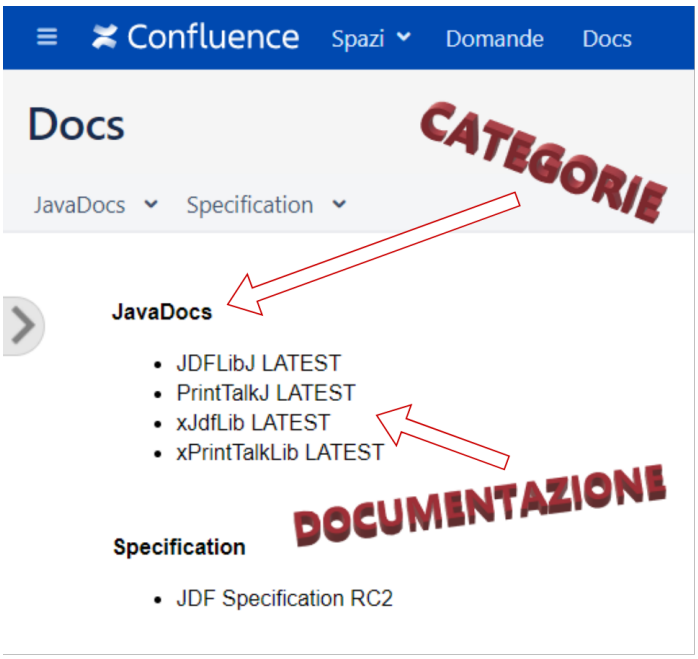

Figura 1.1: Screenshot di un esempio di Docs Plug-in

Ciò che fa il prodotto ottenuto è caricare in maniera automatica la documentazione nella corretta categoria, realizzando un nuovo *doc* o aggiornandone uno esistente. Nel caso in cui l’utilizzatore del plugin Maven fornisca un titolo per la pagina *doc* che non è presente nel *Docs*, ne verrà creata una nuova, altrimenti il *doc* già esistente con quel nome verrà aggiornato.

1.5 Organizzazione del testo

Il secondo capitolo comprende l’analisi dettagliata dei requisiti del prodotto con casi d’uso e il relativo tracciamento dei requisiti individuati.

Il terzo capitolo descrive la progettazione del software.

Il quarto capitolo approfondisce la realizzazione del plugin e come è stata effettuata l’attività di testing.

Il quinto capitolo corrisponde al capitolo conclusivo. Esso riassume il risultato finale ottenuto e attua una valutazine critica del prodotto.

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

* i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*;

- per tutti i concetti che possono essere riassunti, viene fornita una tabella o un elenco puntato.

Capitolo 2

Analisi dei requisiti

Tale capitolo ha l'obiettivo di esporre e analizzare i requisiti espliciti e impliciti per la realizzazione del plugin Maven per la pubblicazione di documentazione software. L'attività di analisi ha funto da base per la fase di progettazione del software, in modo che il prodotto fosse conforme alle richieste dell'azienda.

2.1 Premessa

Il prodotto realizzato è un plugin Maven e possiede come nome ufficiale: *Maven documentation publisher plug-in*. Un plugin Maven è un programma non autonomo che interagisce con la tecnologia Maven per ampliarne o estenderne le funzionalità originarie.

Essendo un plugin Maven, esso deve avere un *goal*, ovvero uno scopo ben preciso. L'obiettivo all'inizio dello stage era il semplice caricamento di documentazione archiviata su *Docs* di Confluence, per questo motivo è stato scelto di creare il goal denominato *publish*. Successivamente, nel corso dello stage, sono state aggiunte delle nuove funzionalità e un nuovo goal, dato che le tempistiche pianificate erano ottimistiche e hanno permesso sufficiente tempo per ampliare il prodotto.

2.2 Descrizione del prodotto

Maven documentation publisher plug-in supporta la pubblicazione di documentazione in formato HTML. Ha due *goal*:

* **publish**: che pubblica la documentazione;
* **cleanup**: che elimina la documentazione contentente SNAPSHOT nel nome.

Il principale è *publish* e si occupa della pubblicazione su *Docs* Confluence della documentazione del codice di un qualunque progetto Maven su cui è configurato il plugin. Questo è possibile perché il plugin *Docs* di Confluence accetta archivi, ovvero file in formato .zip o .jar. La documentazione in questo formato può essere per esempio la documentazione Javadoc (documentazione del codice sorgente scritto in linguaggio Java) o Open API (specifica per file di interfaccia leggibili dalle macchine per descrivere servizi web RESTful, conosciuta anche come specifica Swagger). Entrambe Javadoc e Open API sono il tipo di documentazione di maggior interesse per l'azienda da

pubblicare sul sistema aziendale.
Ogni archivio caricato contribuisce alla creazione di una pagina *doc*. Ogni *doc* viene identificato univocamente all'interno di una categoria, per questo motivo, il titolo deve essere unico. Una pagina viene creata se il titolo della documentazione è nuovo, altrimenti la pagina già esistente viene semplicemente aggiornata.

Maven documentation publisher plug-in è altamente configurabile, in modo da soddisfare qualunque esigenza dello sviluppatore. Innanzitutto esso consente all'utente di inserire:

* la documentazione;
* le proprie credenziali per accedere a Confluence;
* il nome della categoria in cui allocare la documentazione.

La documentazione che fornisce l'utente può essere di tre tipi:

1. archivio (.zip o .jar);
2. cartella (contentente più file HTML);
3. singolo file HTML.

Nei casi 2. e 3. il plugin si occupa anche dell'archiviazione di quei file.

I possibili modi per fornire le credenziali sono molteplici:

* username e password vengono date direttamente nella configurazione

L'utente può inoltre
username and password are users credentials directly supplied by user; serverId corresponds to the server id presents in .m2/settings.xml and permits to get credentials there; In the above example, there are two ways to get credentials. Whether both of them have values, serverId is the used one.
There must be at least one way to get credentials.
Il secondo *goal*, *cleanup*, è nato dalla necessità di eliminare la documentazione relativa ad un prodotto che non è stato rilasciato. Questo tipo di prodotti presentano "SNAPSHOT" nella versione e per questo motivo, anche il titolo della pagina *doc* lo contiene. Si ha quindi qui a che vedere con la pulizia totale dal plugin *Docs* di tutte queste pagine.

2.3 Requisiti

Ad ogni requisito viene assegnato il codice identificativo univoco:

`R[Numero][Tipo][Priorità]`

in cui ogni parte ha un significato preciso:

* **R**: requisito.
* **Numero**: numero progressivo che segue una struttura gerarchica.
* **Tipo**: la la tipologia di requisito che può essere di:

  - **F**: funzionalità.

  - **Q**: qualità.

  - **V**: vincolo.

* **Priorità**: indica il grado di urgenza di un requisito di essere soddisfatto, come:

  - **0**: opzionale.

  - **1**: desiderabile.

  - **2**: obbligatorio.

Esempio: `R2Q1` indica il secondo requisito di qualità ed è desiderabile.

2.3.1 Requisiti di funzionalità

Codice	Requisito	Fonte
R1F2	Il sistema deve fornire delle proprietà per tutti gli elementi configurabili dall'utente	Azienda
R2F2	L'utente deve poter pubblicare la documentazione da lui scelta	Azienda
R2.1F2	L'utente deve poter pubblicare un archivio (.zip o .jar)	Azienda
R2.2F2	L'utente deve poter pubblicare un file html	Azienda
R2.3F2	L'utente deve poter pubblicare una cartella	Azienda
R3F2	Il sistema deve dare un messaggio di errore se l'utente non fornisce nessuna documentazione	Azienda
R4F2	Il sistema deve dare un messaggio di errore se l'archivio dato non esiste	Azienda
R5F2	Inserimento credenziali	Azienda
R5.1F2	Inserimento username	Azienda
R5.2F2	Inserimento password	Azienda
R5.3F2	Inserimento identificativo server	Azienda
R6F2	Il sistema deve dare un messaggio di errore se l'utente non fornisce, in almeno uno dei due modi, le sue credenziali	Azienda
R7F2	L'utente deve poter inserire il nome della categoria Confluence in cui allocare la documentazione	Azienda
R8F2	Il sistema deve dare un messaggio di errore se la categoria non è stata aggiunta	Azienda
R9F2	L'utente deve poter modificare il luogo in cui l'archivio viene salvato all'interno del progetto	Azienda
R10F2	L'utente deve poter modificare le tipologie di file da inserire nella documentazione	Azienda
R11F2	L'utente deve poter modificare le tipologie di file da includere nella cartella	Azienda
R12F2	L'utente deve poter modificare le tipologie di file da escludere dalla cartella	Azienda
R13F2	Il sistema deve fornire il nome del "main entrance file" di ogni pagina Doc del plugin Confluence	Azienda
R14F2	L'utente deve poter modificare il nome del file principale della documentazione	Azienda
R15F2	L'utente deve poter inserire il nome della documentazione	Azienda

Tabella 2.1: Elenco dei requisiti di funzionalità (1)

Codice	Requisito	Fonte
R16F2	L'utente deve poter inserire la versione della documentazione	Azienda
R17F1	Il sistema deve essere in grado di costruire il titolo della pagina contenente la documentazione, a partire da nome e versione della documentazione	Interno
R18F2	L'utente deve poter configurare il plugin in modo che esso non fallisca se avvengono errori del client	Azienda
R19F2	L'utente deve poter configurare il plugin in modo che esso ne salti la propria esecuzione	Azienda
R20F2	L'utente deve poter inserire i tipi di progetto supportati dal plugin	Azienda
R21F2	Il sistema deve permettere il salto dell'esecuzione del plugin, nel caso in cui il progetto compilato non sia tra i tipi supportati	Azienda
R22F2	L'utente deve poter inserie i tipi di progetto a cui il plugin non deve dare messaggi di avvertimento	Azienda
R23F0	L'utente deve poter configurare il plugin in modo che esso non fallisca se l'archivio dato non esiste	Azienda
R24F0	L'utente deve poter eliminare tutta la documentazione con versione "SNAPSHOT" caricata	Azienda

Tabella 2.2: Elenco dei requisiti di funzionalità (2)

2.3.2 Requisiti di qualità

Codice	Requisito	Fonte
R1Q1	La copertura dei test deve essere almeno pari al 70% del codice	Azienda
R2Q1	Le norme presenti sulla wiki aziendale devono essere rispettate	Azienda
R2.1Q1	Ogni commit effettuato deve rispettare la formattazione descritta nella wiki	Azienda
R2.2Q1	Il nome di ogni variabile, classe, ecc nel codice deve essere significativo	Azienda
R2.3Q1	I commenti nel codice devono essere facilmente comprensibili	Azienda
R2.4Q1	Il codice non deve contenere violazioni di SonarQube con alta severità	Azienda
R3Q1	Ogni messaggio di errore del plugin deve essere sufficientemente esplicativo	Interno

Tabella 2.3: Elenco dei requisiti di qualità (1)

Codice	Requisito	Fonte
R4Q2	Deve essere redatto un manuale utente	Azienda
R4.1Q2	Deve essere redatta una pagina Confluence che descriva come configurare il plugin	Azienda
R4.2Q1	Deve essere redatta una pagina di utilizzo Maven “Usage” che descriva tutti i possibili utilizzi del plugin	Azienda
R5Q2	Deve essere redatto un manuale dello sviluppatore	Azienda
R5.1Q2	Deve essere redatta una pagina Confluence che descriva la progettazione del plugin tramite diagrammi	Azienda
R5.2Q2	Deve essere redatta e generata la documentazione Javadoc del plugin	Azienda

Tabella 2.4: Elenco dei requisiti di qualità (2)

2.3.3 Requisiti di vincolo

Codice	Requisito	Fonte
R1V2	Il plugin deve essere sviluppato nel linguaggio di programmazione Java	Azienda
R2V2	Il plugin deve essere testato tramite JUnit	Azienda
R3V2	Come ambiente di sviluppo è necessario utilizzare Eclipse	Azienda
R4V2	Per la build dei progetti è necessario utilizzare Maven	Azienda
R5V2	Per la pubblicazione di documentazione è necessario utilizzare Confluence	Azienda
R6V2	I requisiti identificati devono essere tracciati su Jira	Azienda
R6.1V2	Lo stato di ogni requisito presente su Jira deve sempre essere opportunamente aggiornato	Azienda
R7V2	Come strumento di Continuous integration è necessario utilizzare Jenkins	Azienda
R8V2	Per l'analisi statica del codice è necessario utilizzare SonarQube	Azienda
R9V2	Per il controllo di versione del codice è necessario utilizzare Bitbucket	Azienda

Tabella 2.5: Elenco dei requisiti di vincolo (1)

Codice	Requisito	Fonte
R10V0	Utilizzare GitKraken come client di Git	Interno
R11V0	Utilizzare JUnit per realizzare test di unità	Interno
R12V0	Utilizzare Visual Studio Code come editor per il codice	Interno
R13V0	Utilizzare SequenceDiagram.org per la creazione dei diagrammi di sequenza	Interno
R14V0	Utilizzare ObjectAid UML Explorer per la creazione dei diagrammi delle classi	Interno
R15V0	Utilizzare Meecrowave per la creazione di un semplice server	Azienda

Tabella 2.6: Elenco dei requisiti di vincolo (2)

2.3.4 Riepilogo dei requisiti

Tipologia	Obbligatori	Desiderabili	Opzionali
Di funzionalità	28	1	0
Di qualità	5	8	0
Di vincolo	10	0	6

Tabella 2.7: Riepilogo dei requisiti

Capitolo 3

Progettazione e realizzazione

Il capitolo corrente ha lo scopo di illustrare l'architettura del plugin nel dettaglio con il supporto di diagrammi e le scelte progettuali effettuate.

3.1 Procedura di lavoro

Inizialmente per capire il funzionamento di un plugin Maven, è stato sviluppato un prototipo. Dopodichè è stata effettuata la scelta delle tecnologie più adatte..

3.2 Tecnologie e librerie utilizzate

In questa sezione viene data una panoramica delle tecnologie e librerie principali utilizzate.

JavaX

* javax.annotation (per le annotazioni Nonnull e Nullable)
* javax.ws.rs.core (per la creazione di risorse: Low-level interfaces and annotations used to create RESTful service resources.)
* javax.xml.bind.annotation (per la trasformazione di JSON in oggetti Java)

Codehaus Plexus

Codehaus Plexus è una collezione di componenti usata da Apache Maven. Per ZipArchiver e utilità.

* org.codehaus.plexus.archiver
* org.codehaus.plexus.util

Maven

* org.apache.maven.plugins.annotations (Mojo, Paremeter, ecc)
* org.apache.maven.plugin (Exceptions)

Jersey

Per il client.

3.3 Diagramma dei package

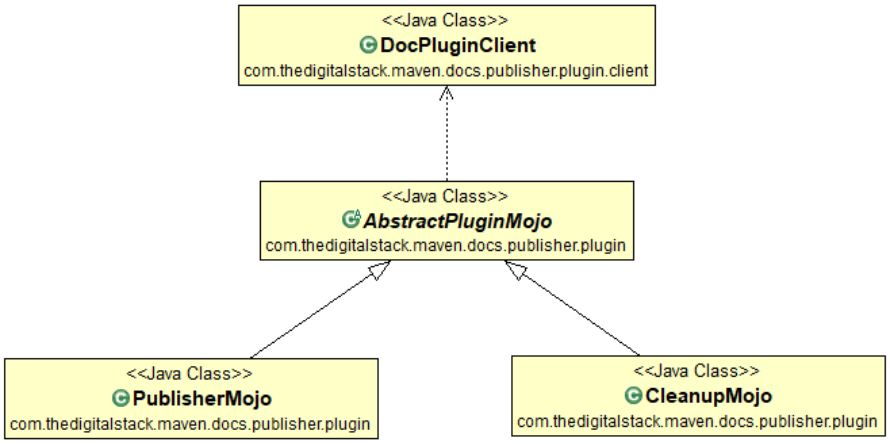

Figura 3.1: Diagramma dei package

3.4 Diagrammi delle classi

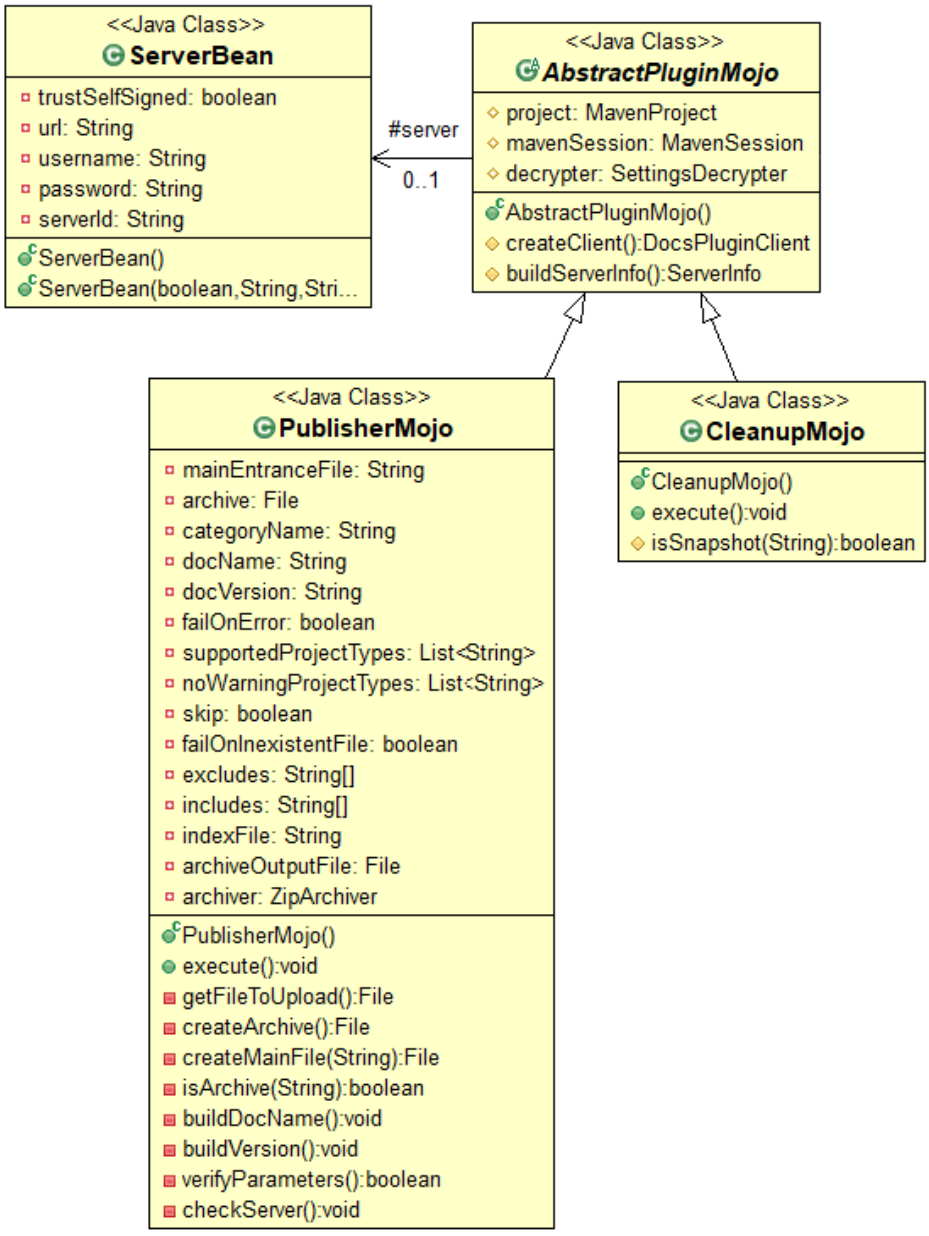

Figura 3.2: Diagramma delle classi relativo alla gerarchia principale del plugin

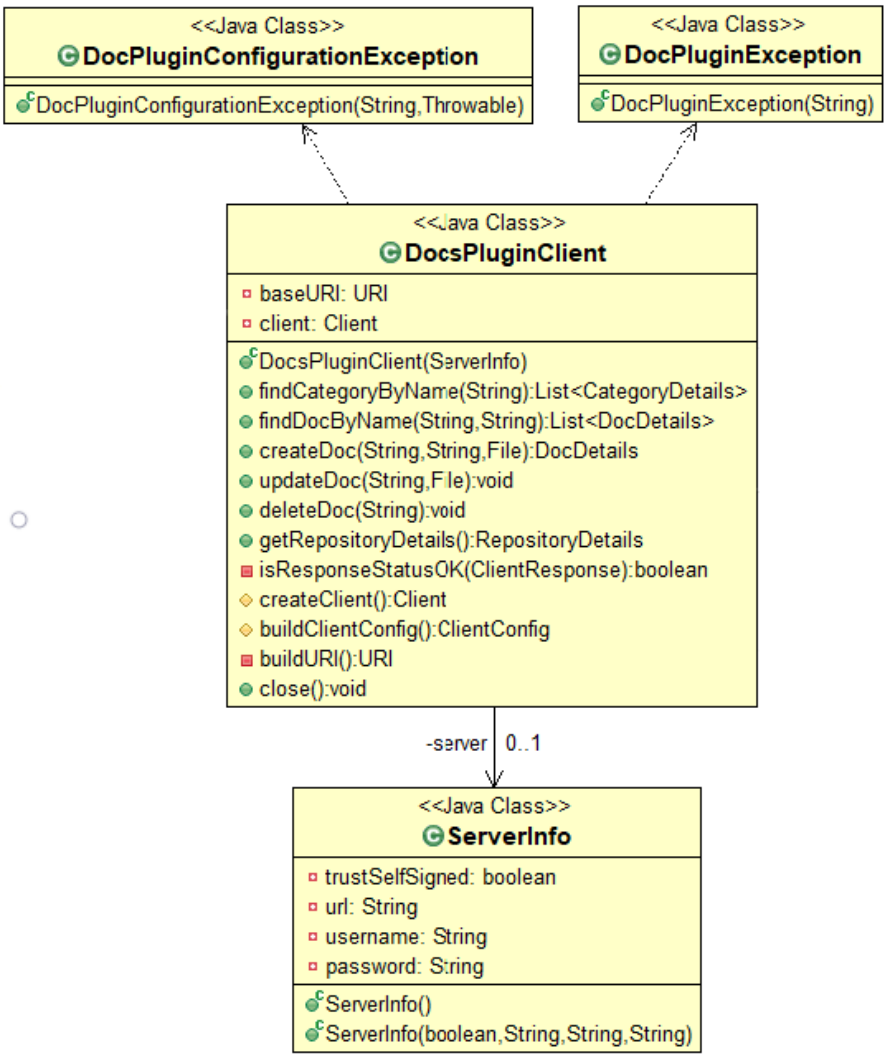

Figura 3.3: Diagramma delle classi relativo al client

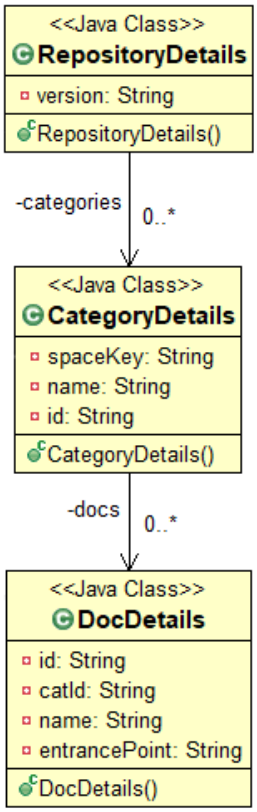

Figura 3.4: Diagramma delle classi relativo ai dettagli di ogni componente del plugin Confluence

3.5 Diagrammi di sequenza

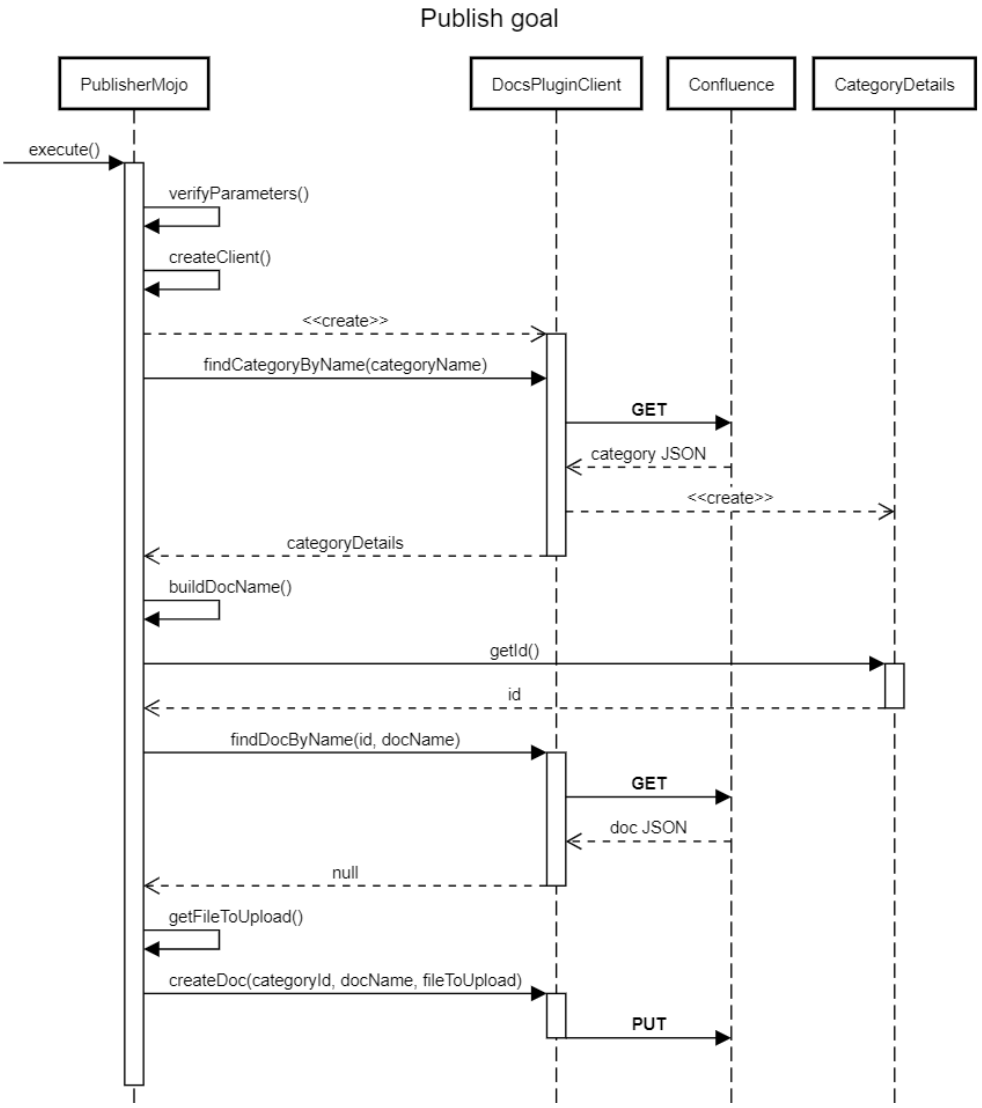

Figura 3.5: Diagramma di sequenza relativo al goal *publish*

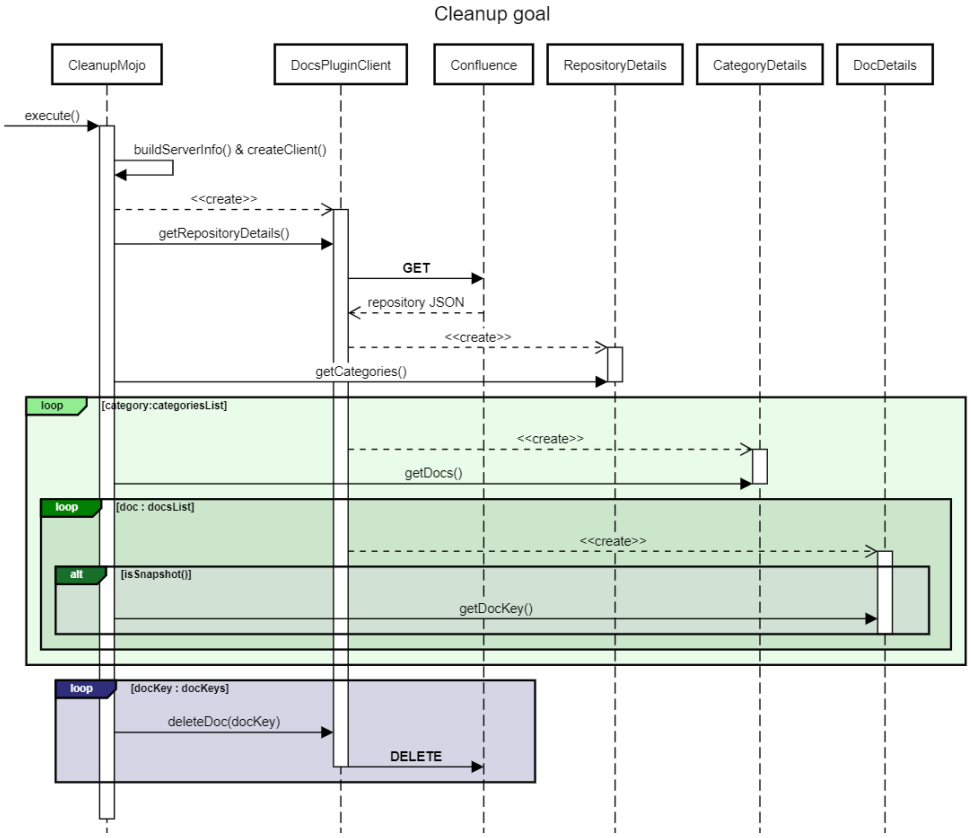

Figura 3.6: Diagramma di sequenza relativo al goal *cleanup*

3.6 Design Pattern utilizzati

3.6.1 Template Method

Capitolo 4

Testing

4.1 Test di unità

Con Mockito

4.2 Test di validazione

Capitolo 5

Conclusioni

5.1 Risultato ottenuto

....

5.2 Analisi critica del prodotto e del lavoro di stage

.....

5.2.1 Utilizzazione del prodotto

Maven documentation publisher plug-in non è ancora stato messo in produzione ma diventerà a breve il metodo ufficiale di pubblicazione della documentazione sul sistema aziendale Confluence.

5.2.2 Valutazione degli strumenti utilizzati

.....

5.2.3 Possibili punti di insoddisfazione

.....

Relativi miglioramenti

.....

5.2.4 Possibili estensioni

.....

Bibliografia