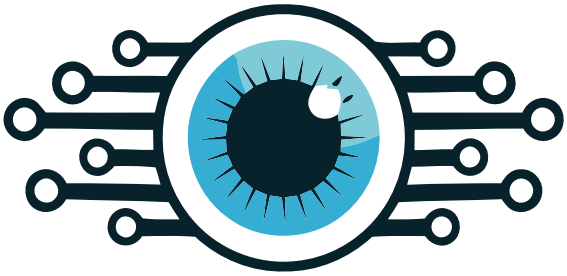
PREMI



Pragma

Norme di Progetto

Informazioni sul documento

Versione	3.0.0
Redazione	Giacomo Manzoli
	Gianmarco Midena
	Stefano Munari
Verifica	Daniele Marin
	Giacomo Manzoli
Approvazione	Andrea Ongaro
	Massimiliano Baruffato
Uso	Interno
Distribuzione	Pragma
Destinato a	Prof. Vardanega Tullio
	Prof. Cardin Riccardo

Sommario

Norme di lavoro stabilite dal gruppo Pragma per la realizzazione del progetto Premi.

A.A. 2014-15

pragma.swe@gmail.com

Diario delle modifiche

Versione	Descrizione modifica	Autore	Ruolo	Data
3.0.0	*Approvazione del documento*	Gianmarco Midena	Responsabile	2015-05-23
2.8.1	*Verifica del documento*	Gianmarco Midena	Verificatore	2015-05-23
2.8.0	*Aggiornamento lista degli errori frequenti (§A)*	Fabio Vedovato	Amministratore	2015-03-22
2.7.0	*Aggiornamento tabella relativa alla rotazione dei ruoli di progetto per decisione E6-1 (tabella 2)*	Fabio Vedovato	Amministratore	2015-05-16
2.6.0	*Aggiunta screenshot relativo alla pagina dei test di* PragmaDB *(figura 15)*	Fabio Vedovato	Amministratore	2015-05-09
2.5.0	*Aggiornamento screenshot relativo alla pagina principale di* PragmaDB *(figura 8)*	Fabio Vedovato	Amministratore	2015-05-09
2.4.0	*Aggiunta classificazione per i test (§3.2.1.2)*	Fabio Vedovato	Amministratore	2015-05-09
2.3.0	*Aggiunta sezione relativa alla notazione utilizzata per modellare i controller di AngularJS per decisione E5-1 (§2.1.2.3.3.1.1.2)*	Fabio Vedovato	Amministratore	2015-05-08
2.2.0	*Aggiornamento sezioni §2.1.2.3 e §2.1.2.3.3.1.1 con nuove convenzioni per la modellazione UML del JavaScript e di Angular*	Fabio Vedovato	Amministratore	2015-05-08
2.1.0	*Aggiunta sezione relativa alle norme progettuali presa da* Definizione di Prodotto v1.0.0 *(§2.1.2.3)*	Fabio Vedovato	Amministratore	2015-05-08
2.0.0	*Approvazione del documento*	Massimiliano Baruffato	Responsabile	2015-03-29
1.26.1	*Verifica del documento*	Daniele Marin	Verificatore	2015-03-28
1.26.0	*Aggiunta descrizione delle nuove funzionalità di* PragmaDB *(§2.1.3.1.6, §2.1.3.1.7, §2.1.3.1.9)*	Giacomo Manzoli	Amministratore	2015-03-12
1.25.0	*Aggiornamento sezione "Editor UML" (§3.1.3.2)*	Giacomo Manzoli	Amministratore	2015-03-12
1.24.0	*Aggiunta norme di codifica relative ad AngularJS (§2.1.2.5.1)*	Giacomo Manzoli	Amministratore	2015-03-12
1.23.0	*Aggiunta norme relative alla progettazione (§2.1.1.2)*	Giacomo Manzoli	Amministratore	2015-03-12

Versione	Descrizione modifica	Autore	Ruolo	Data
1.22.0	*Aggiunta norma riguardo i nomi dei componenti del team in (§3.1.2.2.1)*	Stefano Munari	Amministratore	2015-03-12
1.21.0	*Aggiunta diagramma di attività relativo alla rilevazione dei rischi (diagramma 5)*	Stefano Munari	Amministratore	2015-03-12
1.20.2	*Approvazione del documento*	Andrea Ongaro	Responsabile	2015-03-08
1.20.1	*Verifica del documento*	Giacomo Manzoli	Verificatore	2015-03-08
1.20.0	*Correzione procedura per la rilevazione dei rischi (§4.1.2.4)*	Gianmarco Midena	Amministratore	2015-03-07
1.19.0	*Modifica titolo da "Nomi e norme stilistiche" a "Nomi e norme stilistiche nel codice" (§2.1.2.5)*	Gianmarco Midena	Amministratore	2015-03-07
1.18.0	*Modifica titolo da "Codifica e convenzioni" a "Codifica e convenzioni nei file" (§2.1.2.4)*	Gianmarco Midena	Amministratore	2015-03-07
1.17.1	*Verifica del documento*	Giacomo Manzoli	Verificatore	2015-03-07
1.17.0	*Aggiunta diagramma di attività relativo alla formalizzazione di documenti (diagramma 1)*	Gianmarco Midena	Amministratore	2015-03-06
1.16.0	*Aggiunta lista degli errori frequenti (§A)*	Gianmarco Midena	Amministratore	2015-03-06
1.15.0	*Aggiornamento norma relativa al diario delle modifiche (§3.1.2.4.2)*	Gianmarco Midena	Amministratore	2015-03-06
1.14.0	*Aggiunta descrizione rotazione ruoli di progetto (§4.1.3.3)*	Gianmarco Midena	Amministratore	2015-03-06
1.13.0	*Aggiunta procedura per la rilevazione dei rischi (§4.1.2.4)*	Gianmarco Midena	Amministratore	2015-03-05
1.12.0	*Aggiornamento meccanismo di versionamento per decisione E1-2 (§3.1.2.6)*	Gianmarco Midena	Amministratore	2015-03-05
1.11.0	*Aggiornamento struttura verbali ufficiali per decisione E1-3 (§3.1.2.5.2.1)*	Gianmarco Midena	Amministratore	2015-03-05
1.10.0	*Aggiunta diagramma di attività relativo al rigetto di ticket da parte del Responsabile e alla chiusura di ticket (diagramma 4)*	Gianmarco Midena	Amministratore	2015-03-04
1.9.0	*Aggiunta diagramma di attività relativo al rigetto di ticket da parte del Verificatore e all'approvazione di ticket (diagramma 2)*	Gianmarco Midena	Amministratore	2015-03-04

Versione	Descrizione modifica	Autore	Ruolo	Data
1.8.0	*Aggiunta diagramma di attività relativo alla lavorazione di ticket (diagramma 6)*	Gianmarco Midena	Amministratore	2015-03-04
1.7.0	*Aggiunta diagramma di attività relativo all'apertura di ticket (diagramma 3)*	Gianmarco Midena	Amministratore	2015-03-04
1.6.0	*Aggiunta sezioni "Regole generali di ticketing" e "Protocollo di utilizzo dei ticket" all'interno della sezione relativa alle norme del processo di gestione*	Gianmarco Midena	Amministratore	2015-03-03
1.5.0	*Aggiunta sezione "Procedure" al processo di verifica (§3.2.2)*	Gianmarco Midena	Amministratore	2015-03-03
1.4.0	*Aggiunta sezione "Procedure" al processo di gestione (§4.1.2)*	Gianmarco Midena	Amministratore	2015-03-03
1.3.0	*Aggiornamento descrizione strumento di sviluppo* "PragmaDB*" (§2.1.3.1)*	Gianmarco Midena	Amministratore	2015-03-02
1.2.0	*Aggiunta attività di verifica prese da* Piano di Qualifica v1.0.0 *(§3.2.1)*	Gianmarco Midena	Amministratore	2015-02-27
1.1.0	*Riorganizzazione struttura generale del documento per processi*	Gianmarco Midena	Amministratore	2015-02-27
1.0.0	*Approvazione documento*	Giacomo Manzoli	Responsabile	2014-12-09
0.5.1	*Verifica del documento*	Gianmarco Midena	Verificatore	2014-12-08
0.4.0	*Stesura ambiente di lavoro*	Andrea Ongaro	Amministratore	2014-12-02
0.3.0	*Stesura parte organizzativa*	Daniele Marin	Amministratore	2014-12-03
0.2.0	*Stesura parte relativa alla documentazione*	Giacomo Manzoli	Amministratore	2014-12-02
0.1.0	*Impostazione scheletro documento*	Giacomo Manzoli	Amministratore	2014-12-01
0.0.0	*Creazione documento*	Giacomo Manzoli	Amministratore	2014-12-01

Tabella 1: Diario delle modifiche.

Indice

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Questo documento definisce le norme che i membri di Pragma dovranno adottare durante lo svolgimento del *progetto*$_G$ Premi al fine di regolamentarne l'intero processo di sviluppo.
Tutti i membri del *team*$_G$ sono tenuti a visionare tale documento e ad applicare le norme in esso contenute, al fine di migliorare la qualità dei processi, uniformare il materiale prodotto e migliorare l'efficienza nel lavoro e nelle operazioni di verifica.
Le norme definite in pratica tratteranno:

- La definizione dei ruoli e l'identificazione delle relative mansioni;
- Le modalità di lavoro durante le diverse fasi del *progetto*$_G$;
- Le interazioni sia tra i soli membri del *team*$_G$ che con entità esterne ad esso;
- Le convenzioni tipografiche e le modalità di stesura dei documenti;
- Gli ambienti di sviluppo, il *repository*$_G$ ed il *ticketing*$_G$.

1.2 Scopo del prodotto

Lo scopo del prodotto è di permettere la creazione e l'esecuzione di presentazioni a partire da *mappe mentali*$_G$. L'utente sarà guidato nella creazione di una *mappa mentale*$_G$ e di uno o più *percorsi di presentazione*$_G$, utilizzando i nodi di tale mappa. L'utente potrà eseguire una presentazione seguendo un *percorso* creato oppure visitando qualsiasi nodo della *mappa* costruita; rompendo così la sequenzialità nella presentazione. Il prodotto sarà utilizzabile attraverso un *browser*$_G$.

1.3 Glossario

Al fine di evitare ogni ambiguità di linguaggio e massimizzare la comprensione dei documenti, i termini tecnici, di dominio, gli acronimi e le parole che necessitano di essere chiarite, sono riportate nel documento *Glossario v2.0.0* . Ogni occorrenza dei vocaboli presenti nel *Glossario* è marcata da una "G" maiuscola in pedice ed è scritta in corsivo (es: *Esempio*$_G$).

1.4 Riferimenti

1.4.1 Normativi

- **Verbale esterno:** *E1 v1.0.0* ;
- **Verbale esterno:** *E5 v1.0.0* ;
- **Verbale esterno:** *E6 v1.0.0* .

1.4.2 Informativi

- **Piano di *Progetto*$_G$**: *Piano di Progetto v3.0.0* ;
- ***Piano di Qualifica***: *Piano di Qualifica v2.10.0* ;
- ***ISO*$_G$ 31-0**: http://en.wikipedia.org/wiki/ISO_31-0;
- ***ISO*$_G$ 8601**: http://it.wikipedia.org/wiki/ISO_8601;
- **Jenkins**: http://en.wikipedia.org/wiki/Jenkins_(software);
- ***ECMAScript*$_G$**: http://en.wikipedia.org/wiki/ECMAScript;

- ***JavaScript*$_G$: The Good Parts**: http://it-ebooks.info/book/274/;
- **Specifiche *UTF-8*$_G$**: http://www.unicode.org/versions/Unicode6.1.0/ch03.pdf;
- **Regole d'indentazione**: http://en.wikipedia.org/wiki/Programming_style;
- ***AngularJS*$_G$ Style Guide**: https://github.com/johnpapa/angular-styleguide.

2 Processi primari

2.1 Processo di sviluppo

2.1.1 Attività

2.1.1.1 Analisi dei requisiti

2.1.1.1.1 Task - Studio di fattibilità

Alla pubblicazione dei capitolati d'appalto il *Responsabile di Progetto* avrà il compito di organizzare un adeguato numero di riunioni, affinché i membri del gruppo possano discutere e confrontarsi. A seguito di quanto emerso durante tali riunioni, gli *Analisti* dovranno procedere con la stesura del documento "*Studio di Fattibilità*", che dovrà essere stilato basandosi sui seguenti punti:

- **Dominio Tecnologico e Applicativo**: riflessione su tecnologie e tecniche richieste per lo sviluppo del capitolato, dominio tecnologico e conoscenze che il *team*$_G$ già possiede;
- **Rapporto Costi/Benefici**: analisi della quantità di requisiti obbligatori e del loro costo in termini di realizzazione in funzione del risultato atteso;
- **Individuazione dei Rischi**: comprensione dei punti critici nella realizzazione e individuazione di eventuali rischi.

2.1.1.1.2 Task - Analisi dei requisiti

Dopo aver concluso lo *Studio di Fattibilità*, gli *Analisti* dovranno procedere con la stesura del documento "*Analisi dei Requisiti*". Lo scopo principale di questa attività è quella di produrre dei requisiti semplici e di facile comprensione, a partire da tutte le informazioni recuperabili. Per automatizzare e velocizzare il più possibile questa attività, è stato creato dal *team*$_G$ il software *PragmaDB*, in cui andranno inseriti tutti i requisiti.

2.1.1.2 Progettazione

2.1.1.2.1 Task - Specifica tecnica

I *Progettisti* devono definire la struttura ad alto livello dell'architettura del sistema e dei singoli componenti, raccogliendo il tutto nella *Specifica Tecnica*. Devono, inoltre, essere definiti i test di integrazione tra le varie componenti, che verranno inseriti in appendice al *Piano di Qualifica*.
I prodotti di questo task saranno:

- **Diagrammi *UML*$_G$:**
  - Diagrammi dei package;
  - Diagrammi delle classi;
  - Diagrammi di sequenza;
  - Diagrammi di attività.
- **Design pattern:** i *Progettisti* devono fornire una descrizione dei *design pattern*$_G$ adottati nella definizione dell'architettura. Questa descrizione dovrà essere accompagnata da un diagramma *UML*$_G$, che ne esemplifichi il funzionamento, e dalle motivazioni che hanno portato all'adozione di tale pattern;
- **Tracciamento delle componenti:** ogni componente dovrà essere tracciato ed associato ad almeno un requisito. In tal modo sarà possibile avere la certezza che tutti i requisiti accettati siano soddisfatti e che ogni componente presente nell'architettura soddisfi almeno un requisito. Tale tracciamento dovrà essere effettuato tramite *PragmaDB*, che si occupa di generare in modo automatico le relative tabelle. Maggiori informazioni sono disponibili nella sezione 2.1.3.1.8;

- **Test d'integrazione:** i *Progettisti* devono definire delle strategie di verifica per poter dimostrare la corretta integrazione tra le varie componenti definite.

2.1.1.2.2 Task - Definizione di prodotto

I Progettisti, a partire dalla *Specifica Tecnica*, devono produrre la *Definizione di Prodotto* dove viene descritta la progettazione di dettaglio del sistema. Lo scopo di questo documento è quello di definire dettagliatamente ogni singola unità di cui è composto il sistema in modo da semplificare l'attività di codifica e allo stesso tempo di non fornire alcun grado di libertà al Programmatore. Parallelamente alla progettazione di dettaglio dei componenti software dovranno essere progettati i relativi test di unità che verranno descritti nel Piano di Qualifica. I prodotti di questo task saranno:

- **Diagrammi *UML*$_G$:**
  - Diagrammi dei package;
  - Diagrammi delle classi;
  - Diagrammi di sequenza.
- **Definizione delle classi:** ogni classe precedentemente progettata viene descritta più nel dettaglio, fornendo una descrizione più approfondita dello scopo, delle sue funzionalità e del suo funzionamento. Per ogni classe dovranno essere anche definiti i vari metodi e attributi che la caratterizzano;
- **Tracciamento delle classi:** ogni classe deve essere tracciata ed associata ad almeno un requisito, in questo modo è possibile avere la certezza che tutti i requisiti accettati siano soddisfatti e che ogni classe presente nell'architettura soddisfi almeno un requisito. Questo tracciamento dev'essere effettuato tramite *PragmaDB*, che si occupa di generare in modo automatico le tabelle di tracciamento. Maggiori informazioni sono disponibili nella sezione 2.1.3.1.8;
- **Test di unità:** i *Progettisti* devono definire le strategie di verifica delle varie classi in modo che durante l'attività di codifica sia possibile verificare che la classe si comporti in modo corretto.

2.1.2 Norme

2.1.2.1 Classificazione dei requisiti

Gli *Analisti* hanno il compito di elencare una lista di requisiti emersi durante lo studio e l'analisi del capitolato scelto ed eventualmente in seguito a degli incontri con il *Proponente*$_G$. I requisiti prodotti devono essere classificati per tipo e per importanza secondo la seguente codifica:

R[Tipo][Importanza][Codice]

- **Tipo**: può assumere solo uno tra i seguenti valori:
  - *F*: Funzionale;
  - *Q*: Qualità;
  - *P*: Prestazionale;
  - *V*: Vincolo.
- **Importanza**: può assumere solo uno tra i seguenti valori:
  - *O*: Obbligatorio;
  - *D*: Desiderabile;
  - *F*: Facoltativo.

- **Codice**: è il codice gerarchico, unico per ogni requisito, che viene espresso in forma numerica (esempio: 1.1.1).

Vengono inoltre specificati per ogni requisito:

- **Descrizione**: deve risultare sintetica, completa ed il meno ambigua possibile;
- **Fonte**: deve risultare una o più tra le seguenti:
  - *Capitolato*: il requisito deriva direttamente dal capitolato;
  - *Verbale*: il requisito è stato aggiunto in seguito ad un incontro con il *Proponente*$_G$ e verbalizzato;
  - *Interno*: il requisito è derivato da considerazioni interne derivanti dall'applicazione di standard di qualità;
  - *Caso d'uso*: il requisito è derivato da uno o più casi d'uso.

2.1.2.2 Classificazione dei casi d'uso

I casi d'uso prodotti devono essere descritti usando il seguente formato:

UC[Codice padre].[Codice di livello]

- **Codice del padre**: codice del caso d'uso padre, se non è possibile individuare un caso d'uso padre, è da omettere;
- **Codice di livello**: codice progressivo che identifica i casi d'uso figli.

Per ogni caso d'uso devono inoltre essere specificati:

- **Nome**: definizione del caso d'uso;
- **Attori**: attori coinvolti nel caso d'uso;
- **Descrizione**: chiara, completa e sintetica del caso d'uso;
- **Precondizione**: condizione che deve essere vera al momento dell'esecuzione del caso d'uso;
- **Postcondizione**: condizione che deve essere vera alla fine dell'esecuzione del caso d'uso;
- **Scenario principale**: spiegazione composta dal flusso dei casi d'uso figli;
- **Scenari Alternativi**: spiegazione composta dai casi d'uso non appartenenti al flusso principale;
- **Estensioni**: spiegazione di tutte le estensioni, se presenti;
- **Inclusioni**: spiegazione di tutte le inclusioni, se presenti;
- **Generalizzazioni**: spiegazione di tutte le generalizzazioni, se presenti.

2.1.2.3 Norme progettuali

2.1.2.3.1 Denominazione di entità e relazioni

Tutti i package, le classi, i metodi e gli attributi devono essere denominati in modo chiaro. Il nome deve avere una lunghezza che non ne pregiudichi chiarezza e leggibilità. Le abbreviazioni ai nomi degli elementi sono ammesse soltanto se risultano immediatamente comprensibili, contestualizzate e non devono causare ambiguità. Inoltre è preferibile utilizzare sostantivi per i nomi delle entità e verbi per le relazioni.

2.1.2.3.2 Notazioni per la progettazione architetturale

Poiché nel linguaggio *JavaScript*$_G$ le funzioni sono costrutti *high-order*$_G$ in aggiunta al formalismo *UML*$_G$ 2.0 è stata definita una notazione apposita per rappresentare il tipo di dato funzione: `function(<parameters>)`. Questa notazione rappresenta quindi il tipo funzione che richiede i parametri `<parameters>`.
Durante la descrizione delle componenti viene spesso utilizzata la notazione "*oggetto contenente le informazioni riguardo ... organizzate come coppie chiave/valore*" per indicare un oggetto *JavaScript*$_G$ che ha come campi dati le varie chiavi e ognuno di questi campi ha come valore il dato associato alla chiave.
Ad esempio "*un oggetto contenente le informazioni riguardanti i* percorsi$_G$ *presenti all'interno di un* progetto$_G$*, organizzate come coppie chiave/valore, usando come chiave l'id del* percorso$_G$ *e come valore il nome*" individua un oggetto che ha come campi dati tutti gli `id` dei vari *percorsi di presentazione*$_G$ e come valore del campo è assegnato il nome del *percorso*$_G$ di presentazione. Per facilitare la comprensione dei diagrammi *UML*$_G$ si è scelto di utilizzare il colore arancione per evidenziare le *librerie*$_G$ esterne inoltre, quando vengono presentati i diagrammi dei package, vengono mostrate le classi prive dei campi dati e dei metodi.
Questa scelta è stata fatta per evitare di creare diagrammi troppo grandi che sarebbero risultati illeggibili. I diagrammi completi di ogni classe saranno quindi presenti solo vicino alle relative descrizioni.

2.1.2.3.3 Notazioni per la progettazione di dettaglio

2.1.2.3.3.1 Notazioni per il *Front-End*$_G$

2.1.2.3.3.1.1 Notazioni derivate da AngularJS

####### 2.1.2.3.3.1.1.1 Directive

Nelle descrizioni degli *scope isolati* delle directive, la modalità con cui vengono passati i parametri utilizza la seguente notazione:

- `@`: indica che il parametro passato allo scope è una stringa e i cambiamenti effettuati su di essa dallo scope non sono visibili all'esterno;
- `=`: indica che il parametro passato allo scope è un riferimento ad un oggetto;
- `&`: indica che il parametro passato è una funzione, che può avere dei parametri, e che lo scope può essere invocato dalla directive.

Come nome per l'oggetto *scope* nella definizione di una directive si è scelto di utilizzare `scope` al posto di `$scope` in quanto rispecchia il nome del campo dati utilizzato nella definizione della directive.
Nei diagrammi delle classi relative alle directive sono stati esplicitati solamente i campi dati, definiti da *AngularJS*$_G$, necessari all'applicazione.

####### 2.1.2.3.3.1.1.2 Controller

Le funzioni che un *controller*$_G$ definisce nell'oggetto `$scope` sono state modellate come metodi pubblici del *controller*$_G$ stesso.

####### 2.1.2.3.3.1.1.3 Promise e $q

Tutti i metodi dei services che eseguono richieste al *server*$_G$ forniscono come valore di ritorno una promessa. Questo è stato modellato indicando come tipo di ritorno la classe `Promise`, maggiori informazioni riguardo l'interfaccia della classe e le *API*$_G$ offerte da `$q` possono essere trovate nella pagina: https://docs.angularjs.org/api/ng/service/$q

2.1.2.3.3.1.1.4 Tipi di dato

Per differenziare i vari tipi di dato usati forniti da *Angular*$_G$ si è scelto di utilizzare la seguente notazione:

- `Scope`: per i vari oggetti `$scope`;
- `Promise`: per le promesse;
- `DOMElement`: per gli oggetti passati jQuery o *jqLite*$_G$ passati alla funzione `link` di una directive;
- `$nomeDelService`: per il tipo dei vari service, es: `$http` indica il tipo del servizio `$http`.

Maggiori informazioni riguardo le interfacce pubbliche di questi oggetti sono disponibili nella documentazione ufficiale del *framework*$_G$.

2.1.2.3.3.2 Notazioni per il *Back-End*$_G$

Per convenzione i metodi definiti in questa componente non hanno mai tipo di ritorno. Infatti il tipo di ritorno in molti casi dipende dal comportamento *runtime*$_G$ del *back-end*$_G$, questo può ritornare un messaggio d'errore oppure un oggetto Response contenente la risposta del *server*$_G$.

2.1.2.3.3.2.1 Notazioni derivate da *framework*$_G$

2.1.2.3.3.2.2 Express

Nell'architettura definita dai *Progettisti* si è preferito mantenere la notazione `nomeRouter` che veniva utilizzato nella versione 3 di Express, al posto di `nomeRoutes` della versione 4 di Express perché è stato ritenuto più significativo. La motivazione di questa scelta deriva dal fatto che ogni modulo contiene tutte *routes* logicamente correlate tra loro, è quindi più facile pensare ad un modulo come *router* specifico anziché come un aggregato di *routes*. Di conseguenza la cartella contenente i vari moduli di *routes* si chiamerà `Routers` anziché `Routes`.

2.1.2.3.3.2.3 *MongoDB*$_G$ e *Mongoose*$_G$

Come identificativo per la ricerca di oggetti nelle `collections` di *MongoDB*$_G$ vengono usati parametri di tipo `ObjectId`$_G$. A livello di implementazione ai metodi con parametri di tipo `ObjectId`$_G$ verranno passate delle stringhe. La conversione viene effettuata automaticamente da *Mongoose*$_G$.

2.1.2.4 Codifica e convenzioni nei file

- Tutti i file devono essere conformi alla codifica *UTF-8*$_G$ senza *BOM*$_G$ (poiché potrebbe causare errori durante la procedura di verifica);
- Per andare a capo viene usato il carattere LF (U+000A);
- Modifiche alle convenzioni stabilite sono possibili solamente dopo una decisione del *Responsabile di Progetto*.

2.1.2.5 Nomi e norme stilistiche nel codice

- I nomi delle funzioni, delle variabili, dei metodi e delle classi devono essere scritti in lingua inglese;
- I nomi delle classi devono avere la prima lettera maiuscola;
- I nomi delle variabili, delle funzioni e dei metodi devono avere la prima lettera minuscola e le eventuali parole che ne compongono il nome devono avere la prima lettera maiuscola e non devono contenere underscore (es.: *myFirstVar*, *testFunction()*);

- Una funzione può avere al più 5 parametri;
- Tutti i file *JavaScript*$_G$ devono essere salvati e consegnati come file con estensione .js;
- Il codice *JavaScript*$_G$ non deve essere hard-coded all'interno del codice *html*$_G$ a meno che non sia inevitabile e quindi necessario;
- Questo tipo di tag devono essere posizionati il più avanti possibile nel codice *html*$_G$ in modo da ridurre gli effetti di ritardo imposti dal caricamento dello script all'interno della pagina;
- Non c'è bisogno di utilizzare gli attributi `language` oppure `type`, infatti per lo standard *HTML5*$_G$ *JavaScript*$_G$ è riconosciuto come linguaggio di scripting di default;
- Indentazione: l'unità di indentazione è 4 spazi, l'uso di tab deve essere evitato in quanto non esiste uno standard a riguardo, sarà quindi necessario utilizzare spazi anche se possono produrre file più pesanti, la differenza sarà praticamente nulla quando la versione del file da caricare verrà minificata;
- Tutto il codice che deve essere eseguito sia sul *client*$_G$ sia sul *server*$_G$ sarà minificato;
- Ogni linea di codice può essere lunga al più 80 caratteri, spezzarla nei punti dove si trovano gli operatori, o preferibilmente le virgole, per renderla più leggibile;
- Tutte le variabili dovrebbero essere dichiarate prima di essere usate, permette di rendere il codice più leggibile e meno prono ad errori;
- È vietato usare variabili globali implicite. In generale l'uso di variabili globali deve essere minimizzato il più possibile;
- L'uso di funzioni globali deve essere per quanto possibile minimizzato;
- Se si utilizzano variabili il primo statement nel corpo di una funzione deve essere `var`. In generale definire le variabili nella prima parte del corpo di una funzione;
- Non devono esserci spazi tra nome della funzione e le parentesi () della lista dei suoi parametri;
- Se si utilizzano funzioni anonime si deve inserire uno spazio tra la keyword function e le parentesi () della lista dei suoi parametri;
- Il simbolo "{" deve essere allineato verticalmente rispetto al simbolo "}" relativo;
- I costruttori devono essere scritti con il prefisso create, esempio createNomeCostruttore;
- I nomi non possono contenere:
  - Il carattere $;
  - Il carattere backslash;
- Le variabili globali vanno scritte a lettere maiuscole;
- È vietato usare lo statement "continue";
- In *JavaScript*$_G$ i blocchi non hanno uno scope. Solo le funzioni ed alcuni statement hanno uno scope. Non usare blocchi eccetto quando richiesto dal linguaggio.

2.1.2.5.1 AngularJS

Per l'utilizzo di *AngularJS*$_G$ si è scelto di adottare le convezioni, suggerite dalla comunità di sviluppatori, reperibili al seguente indirizzo:

`https://github.com/johnpapa/angular-styleguide`

Dovranno poi essere adottate le seguenti convenzioni:

- I costruttori dei `controllers` non sono soggetti a limitazioni sul numero di parametri;
- I nomi delle `directives` dovranno avere il prefisso `premi`;
- Le `directives` dovranno avere sempre `scope` isolato, tranne in situazioni particolari, come nel caso di utilizzo di *librerie*$_G$ esterne. Scelte discostanti dalla normale prassi dovranno essere giustificate e sottoposte ad approvazione.

2.1.2.5.2 *MongoDB*$_G$ e *Mongoose*$_G$

Per l'utilizzo di *MongoDB*$_G$ e *Mongoose*$_G$ si è scelto di adottare le seguenti convenzioni:

- Al posto dei costruttori vengono utilizzate funzioni statiche per avere un maggior controllo degli errori;
- Come identificativo per cercare oggetti nelle `collections` di *MongoDB*$_G$ vengono usati parametri di tipo `ObjectId`$_G$, anche se in realtà ai metodi vengono passate delle stringhe. La conversione viene effettuata automaticamente da *Mongoose*$_G$.

2.1.2.5.3 Commenti

- L'utilizzo di commenti all'interno dei file contenenti codice è obbligatorio. Ogni file deve iniziare con un'intestazione nella forma:

```
/**
*
* @class Nome del file
* @author: Autore del file (Indirizzo e-mail dell'autore)
* Data: data in cui è stato creato il file
* Requisiti: lista dei requisiti che il codice sorgente soddisfa
* @classdesc breve descrizione del file
*
*/
```

- Se nel file vi sono più funzioni o metodi va aggiunto un commento che specifichi una breve descrizione prima di ognuno di essi;
- I commenti saranno scritti in lingua italiana;
- I commenti presenti all'interno dei file di codice devono seguire JSDoc, le cui regole sono descritte all'indirizzo `http://en.wikipedia.org/wiki/JSDoc`.

2.1.2.6 Ricorsione

La ricorsione va evitata quando possibile. Per ogni funzione ricorsiva sarà necessario fornire una prova di terminazione e sarà necessario valutare il costo in termini di occupazione della memoria. Nel caso l'utilizzo di memoria risulti troppo elevato la ricorsione verrà rimossa.

2.1.3 Strumenti

2.1.3.1 *PragmaDB*

PragmaDB è un'applicazione *web*$_G$ sviluppata dal *team*$_G$ per la gestione di casi d'uso, attori, fonti, requisiti e termini del *Glossario*. Il suo scopo è velocizzare e automatizzare la gestione dei dati, e semplificare i tracciamenti. Ogni membro del *team*$_G$ può accedere all'applicazione via *browser*$_G$ previa autenticazione, inoltre, più persone possono lavorare simultaneamente sugli stessi dati poiché è stata gestita la concorrenza. L'applicazione si occupa di mantenere ordinata la gerarchia dei requisiti e degli UC in modo automatico durante tutto il suo ciclo di vita (creazione, modifica, eliminazione).
Vengono eseguiti diversi controlli su ciascun campo dati, e tale approccio garantisce *fault-tolerance*$_G$ del sistema nei confronti di qualsiasi input possibile. I membri del gruppo possono visualizzare, inserire, modificare ed eliminare elementi in modo semplice. *PragmaDB* consente di esportare l'elenco degli attori, delle fonti, del *Glossario*, dei casi d'uso, dei requisiti e delle loro relazioni come codice LaTeX, quindi facilmente inseribile all'interno dei documenti durante la stesura. Infine consente di visualizzare, attraverso un menù laterale, l'insieme dei link utili relativi al gruppo (link al *repository*$_G$, link al foglio dei comandi LaTeX, link a Redmine, link alla *mailing list*$_G$ Yahoo).

2.1.3.1.1 Attori

La sezione degli attori è pensata per mantenere traccia di tutti gli attori del sistema. Un attore è caratterizzato da un nome identificativo e da una descrizione. È possibile inserire, modificare o eliminare attori, ed esportare in LaTeX una tabella contenente tutte le informazioni su di essi. Cliccando sul nome di un attore è possibile vederne il dettaglio dello stesso, che mostra in particolare, una lista dei casi d'uso correlati, per facilitarne il processo di verifica. È consentita l'eliminazione di un attore solamente se non esiste alcun caso d'uso ad essa riferito.

2.1.3.1.2 Fonti

La sezione delle fonti è pensata per mantenere traccia di tutte le fonti (capitolati, verbali, ecc...) che hanno determinato l'individuazione di un requisito. Una fonte è caratterizzata da un identificativo, un nome e una descrizione. È possibile inserire, modificare o eliminare fonti, ed esportare in LaTeX una tabella contenente tutte le informazioni su di esse. Cliccando sull'identificativo di una fonte è possibile vederne il dettaglio dello stesso, che mostra in particolare, una lista dei requisiti correlati, per facilitarne il processo di verifica. È consentita l'eliminazione di una fonte solamente se non esiste alcun requisito derivato da essa.

2.1.3.1.3 Requisiti

La sezione dei requisiti è pensata per mantenere traccia di tutti i requisiti individuati. Un requisito è caratterizzato da un identificativo, un tipo (funzionale, di vincolo, di qualità, prestazionale), un livello di importanza (obbligatorio, desiderabile, facoltativo), un requisito padre (se non è radice), uno stato (accettato/non accettato + soddisfatto/non soddisfatto + implementato/non implementato), una o più fonti di riferimento e una descrizione. È possibile inserire, modificare o eliminare requisiti, ed esportare in LaTeX una tabella contenente tutte le informazioni su di essi. Cliccando sull'identificativo di un requisito è possibile vederne il dettaglio dello stesso, che mostra in particolare, una lista dei requisiti figli e dei casi d'uso correlati, per facilitarne il processo di verifica. È inoltre presente una funzionalità che permette di vedere quali requisiti non sono correlati ad alcun caso d'uso.

2.1.3.1.4 Casi d'uso

La sezione dei casi d'uso è pensata per mantenere traccia di tutti i casi d'uso individuati dagli *Analisti*. Un caso d'uso è caratterizzato da un identificativo, un nome, un diagramma, una o più precondizioni, una o più postcondizioni, un caso d'uso padre (se non è radice), uno scenario principale, una o più inclusioni (facoltativo), una o più estensioni (facoltativo), uno o più scenari alternativi (facoltativo) e una

descrizione. È possibile inserire, modificare o eliminare casi d'uso, ed esportare in LaTeX una tabella contenente tutte le informazioni su di essi. Cliccando sull'identificativo di un caso d'uso è possibile vederne il dettaglio dello stesso, che mostra in particolare, una lista dei casi d'uso figli e dei requisiti correlati, per facilitarne il processo di verifica. È inoltre presente una funzionalità che permette di vedere quali casi d'uso non sono correlati ad alcun requisito.

2.1.3.1.5 *Glossario*

Nell'area dedicata alla gestione del *Glossario* è possibile inserire, modificare o eliminare termini, ed esportare in LaTeX l'intero *Glossario* sotto forma di lista di `newglossaryentry`, macro fornita dal package `glossaries` di LaTeX. Una voce di *Glossario* è caratterizzata da un *identificativo*, un *nome* singolare che verrà mostrato nel *Glossario*, una descrizione, un *plurale* (facoltativo) per indicare la sua forma plurale, una forma *estesa* (facoltativo) per indicare la forma singolare (più rara) che il termine può assumere alla sua prima occorrenza all'interno dei documenti, una forma *estesa plurale* (facoltativo) per indicare la forma plurale (più rara) che il termine può assumere alla sua prima occorrenza nei documenti e un *sinonimo* (facoltativo).

2.1.3.1.6 Package e Classi

Per ciascun componente (package o classe) è possibile specificare:

- Un diagramma UML_G;
- Una descrizione testuale del componente;
- Una descrizione del contesto d'utilizzo;
- Le relazioni con gli altri componenti;
- I requisiti che il componente va a soddisfare.

Per le classi è inoltre possibile specificare gli attributi e i metodi che le caratterizzano e l'eventuale gerarchia di classi a cui appartengono. Alcuni screenshot esemplificativi di queste funzionalità sono stati inseriti nell'appendice C.

2.1.3.1.7 Test

In questa sezione è possibile inserire i test definiti dai *Progettisti*. Ogni test è caratterizzato da un tipo e una descrizione. I tipi di test sono: validazione, sistema, integrazione oppure unità; e in base a tale tipo, un test può essere associato a un requisito, a un componente oppure a un metodo di una classe. Ogni test è caratterizzato da un id univoco, calcolato automaticamente da *PragmaDB* e da una serie di parametri che specificano se il test è stato implementato, eseguito e superato.

2.1.3.1.8 Tracciamento

PragmaDB consente di esportare direttamente in LaTeX molte parti dei documenti del $progetto_G$, automatizzandone il processo di stesura. Grazie ad uno script appositamente creato è possibile in ogni momento scaricare tutti i file LaTeX, inserendoli nelle cartelle dei rispettivi documenti. *PragmaDB* consente il tracciamento delle seguenti coppie di elementi:

- Requisiti - Fonti;
- Fonti - Requisiti;
- Componenti - Requisiti;
- Requisiti - Componenti;
- Classi - Requisiti;

- Requisiti - Classi;
- Elemento - Test;
- Test - Elemento.

2.1.3.1.9 Funzionalità di supporto

Per semplificare la verifica del tracciamento tra gli elementi memorizzati in *PragmaDB*, sono state aggiunte funzionalità che permettono di ottenere una lista degli elementi non relazionati con nessun altro elemento.
In particolare *PragmaDB* fornisce la lista di:

- Requisiti non sono derivati da casi d'uso;
- Casi d'uso che non generano requisiti;
- Package che non soddisfano requisiti;
- Classi che non soddisfano requisiti;
- Test che non verificano classi o requisiti.

2.1.3.2 WebStorm

L'ambiente di sviluppo integrato (IDE) utilizzato è ***WebStorm***$_G$. Sono stati testati anche altri *IDE*$_G$, ma nessuno si è dimostrato all'altezza di ***WebStorm***$_G$. Esso presenta le seguenti funzionalità:

- Autocompletamento del codice *JavaScript*$_G$, *HTML*$_G$ 5 e CSS3;
- Autocompletamento per metodi, funzioni e *framework*$_G$ esterni, utili per il *progetto*$_G$;
- Debugger *JavaScript*$_G$;
- Consente di effettuare unit test per *JavaScript*$_G$ mediante il *framework*$_G$ ***Karma***$_G$;
- Compila automaticamente i file *Sass*$_G$ in *CSS*$_G$;
- Tiene traccia dei cambiamenti effettuati sui file, consentendo di visualizzare lo storico delle modifiche locali e ritornare a versioni precedenti in caso di modifiche o perdite accidentali.

3 Processi di supporto

3.1 Processo di documentazione

In questa sezione vengono descritte tutte le convenzioni riguardanti la stesura di documenti scelte e adottate da Pragma.

3.1.1 Procedure

3.1.1.1 Formalizzazione dei documenti

Un documento può trovarsi in tre stadi diversi:

- **In lavorazione:** quando necessita di modifiche, da appena creato e per tutto il periodo necessario alla sua realizzazione;
- **Da verificare:** dopo il termine della sua realizzazione, quando viene preso in consegna dai *Verificatori*, che avranno il compito di rilevare e correggere eventuali errori o imprecisioni;
- **Approvato:** dopo l'approvazione del *Responsabile di Progetto*, che avviene dopo il termine della fase di verifica.

Ogni documento ripeterà questo suo ciclo interno ad ogni avanzamento di fase, cioè ogni volta che avverrà un aumento dell'indice di versione più significativo. Tale procedura di formalizzazione viene riassunta nel diagramma di attività indicato in figura 1.

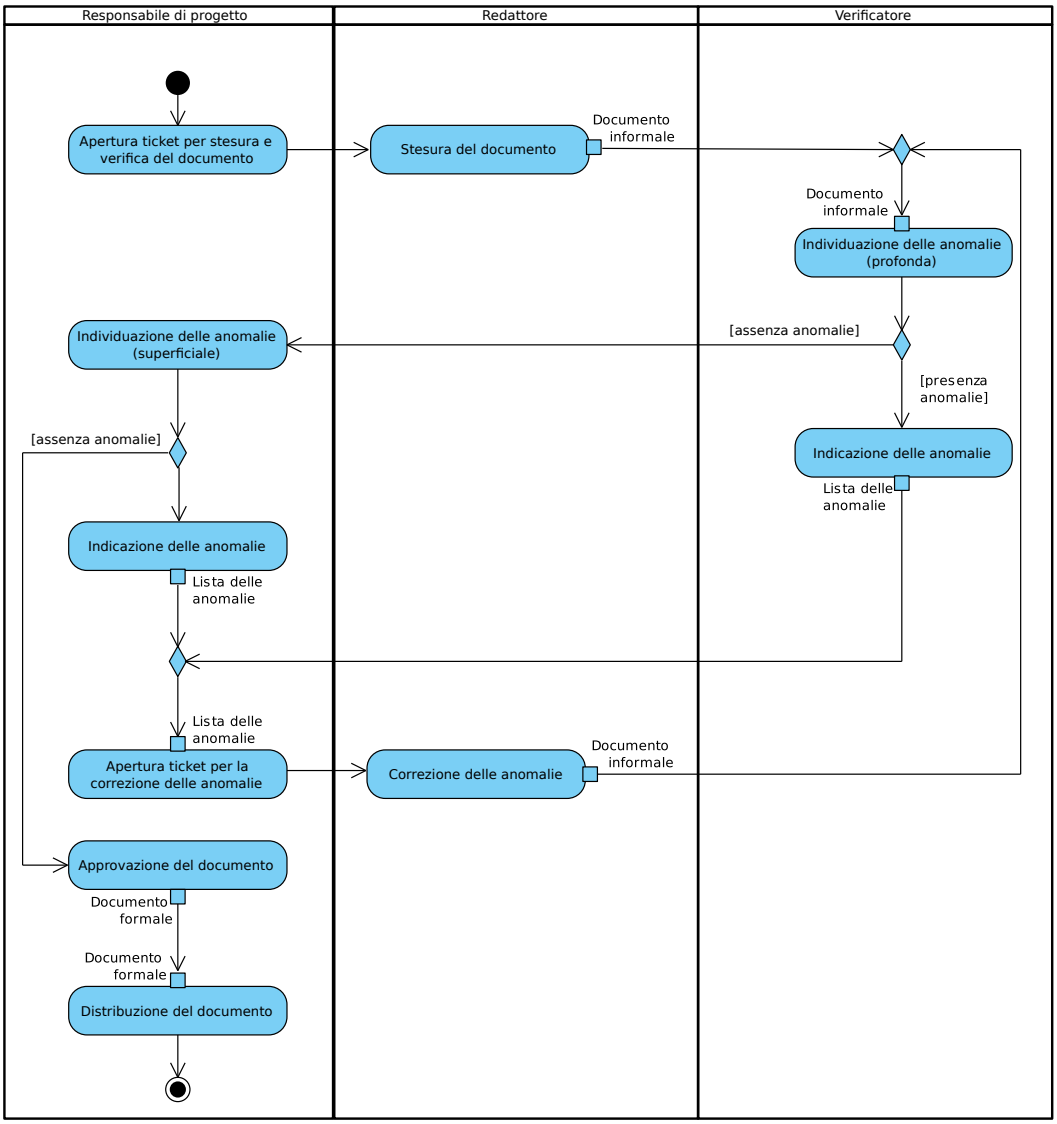


Figura 1: Diagramma di attività - formalizzazione di un documento

3.1.2 Norme

3.1.2.1 Template

Per agevolare e uniformare la redazione dei documenti è stato creato un *template*$_G$ LaTeX contenente tutte le impostazioni stilistiche e i comandi citati in questo documento.
Il *template*$_G$ è disponibile nel *repository*$_G$ `pragmaDocs` all'interno della cartella `template`.

3.1.2.2 Norme tipografiche

Questa sezione racchiude le convenzioni adottate da Pragma per scrivere i documenti in modo uniforme.

3.1.2.2.1 Norme riguardo nomi

Ogniqualvolta si debbano elencare i nomi completi dei componenti del gruppo, tale elenco dovrà essere ordinato lessicograficamente per cognome, e deve sempre essere indicato prima il nome e poi il cognome.

3.1.2.2.2 Stile del testo

- **Corsivo:** deve essere usato quando ci si riferisce al nome di un documento o a un ruolo, quando si scrivono delle citazioni o abbreviazioni e in tutti gli altri casi in cui si ritiene sia utile mettere in rilievo del testo;
- **Grassetto:** può essere usato per evidenziare delle parole chiave oppure per evidenziare il concetto sviluppato da una voce in un elenco puntato;
- ***Monospace*$_G$:** deve essere utilizzato quando si riportano parti di codice, comandi o *percorsi*$_G$ di file;
- **Maiuscolo:** l'utilizzo del maiuscolo è riservato solamente per le macro;
- **LaTeX:** per riferirsi a LaTeX è obbligatorio usare l'apposito comando \LaTeX.

3.1.2.2.3 Punteggiatura

- **Punteggiatura:** un carattere di punteggiatura non deve mai essere preceduto da un carattere di spaziatura;
- **Parentesi:** il testo racchiuso tra parentesi non deve mai iniziare o terminare con un carattere di spaziatura o di punteggiatura, inoltre all'interno del testo racchiuso tra parentesi non deve esserci un altro gruppo di parentesi;
- **Lettere maiuscole:** le lettere maiuscole vanno usate per indicare il nome del *team*$_G$, del *progetto*$_G$, dei documenti, dei ruoli, delle fasi di lavoro, nelle parole *Proponente*$_G$ e *Committente*$_G$, all'inizio di un punto di un elenco puntato e in tutte le occasioni in cui ne è previsto l'uso dalla lingua italiana;
- **Ritorno a capo:** la decisione riguardo l'uso del ritorno a capo è lasciata a chi scrive il documento, questo perché l'andare a capo dipende dal contesto.

Queste convenzioni possono essere trascurate solamente quando si inserisce del codice sorgente all'interno del documento.

3.1.2.2.4 Composizione del testo

- **Elenchi puntati:** ogni punto dell'elenco deve terminare con un punto e virgola, fatta eccezione per l'ultimo elemento che deve terminare con un punto. La prima parola deve iniziare con una lettera maiuscola, salvo casi particolari in cui è richiesto l'uso della lettera minuscola (es: nome di un file);
- **Note a pié pagina:** ogni nota deve cominciare con l'iniziale della prima parola maiuscola e non deve essere preceduta da alcun carattere di spaziatura. Ogni nota deve inoltre terminare con un punto;
- **Sigle:** l'uso delle sigle è consentito solamente nei casi in cui sia necessario risparmiare spazio come per esempio nelle tabelle o diagrammi. Le sigle che si prevedono di utilizzare sono:
  - **AdR:** *Analisi dei Requisiti*;
  - **GL:** *Glossario*;
  - **NdP:** *Norme di Progetto*;

- **PdP:** *Piano di Progetto*;
- **PdQ:** *Piano di Qualifica*;
- **SdF:** *Studio di Fattibilità*;
- **ST:** *Specifica Tecnica*;
- **RR: Revisione dei Requisiti**;
- **RP: Revisione di Progettazione**;
- **RQ: Revisione di Qualifica**;
- **RA: Revisione di Accettazione**;
- **Re:** *Responsabile di Progetto*;
- **Am:** *Amministratore*;
- **An:** *Analista*;
- **Pt:** *Progettista*;
- **Ve:** *Verificatore*;
- **Pr:** *Programmatore*.

3.1.2.2.5 Formati ricorrenti

- ***Percorsi***$_G$**:** per gli indirizzi email e *web*$_G$ deve essere utilizzato il comando \url, mentre per gli indirizzi relativi va usato il comando LaTeX \texttt che usa il formato *monospace*$_G$;
- **Date:** le date dovranno seguire il formato

  AAAA-MM-GG

  dove:

  - AAAA: rappresenta l'anno scritto utilizzando 4 cifre;
  - MM: rappresenta il mese scritto utilizzando sempre 2 cifre;
  - GG: rappresenta il giorni scritto utilizzando sempre 2 cifre.
- **Numeri:** i numeri saranno formattati secondo lo standard [SI/ISO 31-0];
- **Nome dei file:** per riferirsi ad un file usandone solo il nome è necessario utilizzare il formato *monospace*$_G$;
- **Nome dei documenti:** per garantire la scrittura uniforme del nome dei documenti sono stati inseriti dei comandi distinti dalle iniziali maiuscole del nome del documento, ad esempio \NP che stampa *Norme di Progetto*.
  Nel caso sia necessario fare riferimento alla versione più aggiornata del documento sono stati predisposti dei comandi LaTeX \nomeDelDocumento che stampano in modo corretto il nome del documento e l'ultima versione approvata, ad esempio *Norme di Progetto v3.0.0* ;
- **Ruoli di *progetto***$_G$**:** per garantire la scrittura uniforme dei ruoli di *progetto*$_G$ sono stati inseriti dei comandi distinti dalle iniziali maiuscole del nome del ruolo e che hanno come prefisso una "r", ad esempio \rRP stampo *Responsabile di Progetto*;
- **Revisioni:** per garantire la scrittura uniforme delle revisioni sono stati inseriti dei comandi caratterizzati dalle iniziali maiuscole dei nomi delle revisioni, ad esempio \RR stampa **Revisione dei Requisiti**;
- **Fasi del *progetto***$_G$**:** per garantire la scrittura uniforme del nome delle fasi sono stati inseriti dei comandi caratterizzati dalle iniziali maiuscole del nome delle fasi preceduti da un "f", ad esempio \fAD stampa **Consolidamento dei requisiti**;

- **Nomi dei componenti:** per riferirsi ai componenti del *team*$_G$ sono state definite le seguenti macro:
  - **ao**: Andrea Ongaro;
  - **fv**: Fabio Vedovato;
  - **sm**: Stefano Munari;
  - **mb**: Massimiliano Baruffato;
  - **dm**: Daniele Marin;
  - **gmi**: Gianmarco Midena;
  - **gma**: Giacomo Manzoli.
- **Nome del gruppo:** ci si riferirà al gruppo solamente con il nome Pragma, per scrivere in modo corretto il nome è stata definita la macro \gruppo;
- **Nome del *Proponente*$_G$:** ci si riferirà al *Proponente*$_G$ come "Zucchetti S.p.A." o con "Proponente". Per la corretta scrittura è stata definita la macro \proponente;
- **Nome del referente del *Proponente*$_G$:** ci si riferirà al referente del *Proponente*$_G$ come "Gregorio Piccoli" o con "Referente Zucchetti S.p.A.". Per la corretta scrittura è stata definita la macro \referenteProponente;
- **Nome del *Committente*$_G$:** ci si riferirà al *Committente*$_G$ come "Prof. Vardanega Tullio" o con "Committente". Per la corretta scrittura è stata definita la macro \committente;
- **Nome del *progetto*$_G$:** ci si riferirà al *progetto*$_G$ solo come "Premi" . Per la corretta scrittura è stata definita la macro \progetto.

Un file con i comandi appena descritti è reperibile al seguente link:
https://docs.google.com/document/d/1dRy2r-Ewp7Ye8iP-YKJz3T5XkkRIbCOZEPJ-aZocWK4/edit

3.1.2.3 Componenti grafiche

3.1.2.3.1 Tabelle

Ad ogni tabella presente all'interno dei documenti deve essere associata una didascalia e un numero identificativo incrementale al fine di renderla tracciabile all'interno del documento.

3.1.2.3.2 Immagini

Il formato preferibile per le immagini è *PDF*$_G$, ma qualora non fosse disponibile è desiderabile l'uso de formato *PNG*$_G$.

3.1.2.4 Struttura dei documenti

3.1.2.4.1 Frontespizio

La prima pagina di ogni documento contiene, nell'ordine, le seguenti informazioni:

- Nome del *progetto*$_G$;
- Logo e nome del gruppo;
- Titolo del documento;
- Versione del documento
- Nome e cognome dei redattori del documento;
- Nome e cognome dei *Verificatori* del documento;

- Nome e cognome del *Responsabile di Progetto*, che dovrà approvare il documento;
- Uso del documento;
- Lista di distribuzione del documento;
- Descrizione del documento;
- Anno accademico;
- Mail del *team*$_G$.

3.1.2.4.2 Diario delle modifiche

La seconda pagina di ogni documento contiene il diario delle modifiche, cioè una tabella contenente le seguenti informazioni:

- Data della modifica;
- Descrizione delle modifiche effettuate; specificandone, quando possibile, le sezioni interessate, e segnalando eventuali riferimenti a sezioni di documenti esterni coinvolti (per riferirsi a decisioni prese in verbali ufficiali si veda la sezione 3.1.2.5.2.1);
- Nome e cognome dell'autore;
- Ruolo ricoperto all'interno del gruppo dall'autore della modifica;
- Versione del documento dopo la modifica.

Le righe della tabella saranno ordinate per data decrescente, in modo che la prima riga della tabella corrisponda all'ultima modifica effettuata.

3.1.2.4.3 Indici

Dopo il diario delle modifiche è presente l'indice delle sezioni, e a seguire, solo nel caso siano presenti figure o tabelle, gli indici delle figure e delle tabelle.

3.1.2.4.4 Struttura generale di una pagina

L'intestazione di ogni pagina contiene:

- Il nome del gruppo;
- La sezione corrente del documento.

Il pié di pagina contiene:

- Il nome del documento;
- Il numero di pagina corrente espresso nella forma *Pagina: X / Y*, dove X è il numero di pagina corrente e Y è il numero di pagine totali.

3.1.2.5 Tipi di documenti

3.1.2.5.1 Documenti interni

Rappresentano documenti redatti per un utilizzo interno a Pragma, che non devono essere distribuiti all'esterno e che non necessitano di *versionamento*$_G$. Questa tipologia di documentazione verrà archiviata su *Google Drive*$_G$. A questa categoria di documenti appartengono le bozze di documento e i verbali interni informali.

3.1.2.5.1.1 Verbali interni informali

Un verbale interno informale è un documento che descrive gli argomenti discussi durante una riunione tra soli membri del *team*$_G$, che dovrà restare a loro esclusiva disposizione. Verrà redatto da un membro del *team*$_G$, condiviso mediante *Google Drive*$_G$, e inviato a tutti i suoi componenti tramite posta elettronica.

3.1.2.5.1.2 Bozze di documenti ufficiali

Per velocizzare la stesura dei documenti informali è possibile iniziare a scrivere un bozza, in modo che anche i membri del gruppo che non sono familiari con LaTeX possano iniziare a produrre materiale fin da subito. Quando viene creata una bozza è necessario comunicarlo in *mailing list*$_G$ usando come oggetto:

```
[Bozza] Nome del documento
```

In ogni caso è necessario che la bozza sia promossa a documento informale il prima possibile.

3.1.2.5.2 Documenti ufficiali

3.1.2.5.2.1 Verbali ufficiali

Un verbale ufficiale è un documento che descrive gli argomenti discussi durante una riunione con il *Proponente*$_G$ (verbale interno) o con il *Committente*$_G$ (verbale esterno), ed ha quindi valore normativo. Ogni sezione del documento dovrà essere dedicata ad un particolare problema trattato all'incontro, che dovrà essere presentato e ne dovrà essere descritta la relativa decisione presa con il *Committente*$_G$.
Il documento dovrà essere consultabile dal *team*$_G$ e dalle parti esterne, quindi verrà allegato ad un messaggio di risposta alla mail di convocazione della riunione esterna e inviato alla *mailing list*$_G$ del *team*$_G$. Per agevolarne l'identificazione, i verbali interni verranno denominati con un codice univoco *TN*, dove:

- *T* rappresenta il tipo del verbale: *E*, per i verbali esterni relativi a riunioni tenute con il *Committente*$_G$, e *I*, per i verbali interni relativi a riunioni con il *Proponente*$_G$;
- *N* è un numero intero che parte da *1* e viene incrementato per ogni nuovo verbale di tipo *T* di verbale.

Per riferirsi ad una precisa coppia problema-decisione di un verbale ufficiale[1], è sufficiente indicare il codice univoco del verbale seguito da un trattino e dal numero della coppia problema-decisione interessata. Per fare ciò si deve utilizzare la notazione *TN-S*, dove *TN* è il codice dell'*N*-esimo verbale ufficiale e *S* è l'*S*-esima coppia problema-decisione.

3.1.2.5.2.2 Documenti informali

Un documento è ritenuto informale finché non viene approvato dal *Responsabile di Progetto*. Appartengono a questa categoria tutti i documenti che dovranno essere consegnati al *Committente*$_G$ o al *Proponente*$_G$, questi documenti sono memorizzati nel *repository*$_G$ `pragmaDocs` e devono attenersi alle *Norme di Progetto*.

3.1.2.5.2.3 Documenti formali

Un documento diventa formale quando viene approvato dal *Responsabile di Progetto*. Per raggiungere questo stato è necessario che il documento sia conforme alle *Norme di Progetto* e che abbia seguito il *percorso*$_G$ di verifica e validazione in esse descritto.

[1] I verbali dovranno comparire anche tra i riferimenti normativi del documento.

3.1.2.5.2.4 *Glossario*

Il *Glossario* conterrà le *parole* dei documenti che possono far parte del contesto dell'applicazione e i *termini* che possono generare ambiguità d'interpretazione. Tali termini saranno disposti in ordine alfabetico ed ognuno di essi avrà una definizione, che dovrà essere sintetica e precisa, per non creare equivoci o ambiguità. Ogni membro del $team_G$ è invitato a inserire nel *Glossario* le parole da esso individuate, delle quali non esista ancora una definizione. L'inserimento dei termini nel *Glossario* avverrà parallelamente alla stesura dei documenti sfruttando le funzionalità offerte da *PragmaDB*.

3.1.2.6 Versionamento

L'avanzamento di versione da parte dei documenti sarà espresso nella seguente forma:

vX.Y.Z

dove:

- **X**: indica il numero di uscite formali del documento e viene incrementato in corrispondenza con l'ultima approvazione del *Responsabile di Progetto* prima del rilascio. L'incremento di **X** comporta l'azzeramento sia di **Y** che di **Z**;
- **Y**: indica il numero di modifiche e correzioni effettuate al documento. L'incremento **Y** comporta l'azzeramento di **Z**;
- **Z**: quando vale
  - ➔ 0: indica che le ultime modifiche (successive all'ultima verifica) non sono state verificate;
  - ➔ 1: indica che le ultime modifiche (successive all'ultima verifica) sono state verificate;
  - ➔ 2: indica che l'ultima verifica è stata approvata.

Quando si fa riferimento al contenuto di una specifica versione di un documento, ne è richiesta la sua precisazione usando la seguente sintassi:

NomeDocumento vX.Y.Z

Quando saranno creati i file, il loro nome dovrà seguire lo schema:

`nomeDocumento_vX.Y.Z.pdf`

3.1.3 Strumenti

3.1.3.1 LaTeX e Texmaker

LaTeX è il sistema scelto dal gruppo per la stesura della documentazione poiché consente di separare facilmente contenuto e formattazione, e permette di creare le varie sezioni in file separati, facilitando la stesura di documenti in parallelo. LaTeX è estendibile attraverso comandi e funzioni definibili dall'utente, e dispone inoltre di svariati pacchetti che ne estendono ulteriormente le funzionalità.

L'editor scelto dal gruppo per la redazione di tali documenti è **Texmaker**, poiché è stato scelto per la cura dell'interfaccia utente e per la completezza delle funzioni che mette a disposizione. Ad esempio è possibile abilitare il controllo ortografico durante la scrittura.

3.1.3.2 Editor UML

Per la modellazione dei diagrammi UML_G è stato scelto l'editor $multipiattaforma_G$ **Visual Paradigm** 2.0, che fornisce supporto completo e presenta un'interfaccia semplice ed esaustiva. Inoltre consente l'importazione e l'esportazione di diagrammi in formato XML_G, che è versionabile.
Per la realizzazione dei diagrammi dei package e delle classi è stato scelto **Astah**, poiché alcuni formalismi grafici di Visual Paradigm non rispettano la notazione UML_G 2.0. Anche **Astah** è $multipiattaforma_G$ ma è inoltre disponibile gratuitamente per studenti[2].

[2]Gli studenti possono richiedere gratuitamente la licenza Professional.

3.1.3.3 Script

Per semplificare la stesura dei documenti sono stati creati alcuni script, presenti nelle cartelle del *repository*$_G$ contenente la documentazione, che consentono di generare i documenti nel formato *pdf*$_G$:

- ***pragmaDocs***: si trova nella cartella principale ed è utilizzabile mediante il comando `./pragmaDocs`. Esso consente di generare tutti i documenti presenti nel *repository*$_G$, ne calcola l'indice Gulpease e marca tutti i termini presenti nel *Glossario*. L'indice Gulpease è calcolato con la seguente formula:

$$89 + \frac{300(numero\ delle\ frasi) - 10(numero\ delle\ lettere)}{numero\ delle\ parole}$$

- **Script per la compilazione dei documenti singolarmente**: all'interno della cartella principale di ogni documento è presente uno script, utilizzabile mediante il comando `./nomeDellaCartella`, che genera il file *pdf*$_G$ del documento corrispondente. Ad esempio nella cartella `PianoDiProgetto` si dovrà utilizzare il comando `./pianoDiProgetto`;
- ***Glossario***: nella cartella del *Glossario* è disponibile lo script `./Glossario` che prende i termini memorizzati nel database *PragmaDB* e genera il corrispettivo documento.

3.1.3.4 JSDoc 3

Per la documentazione di *namespace*, *classi*, *metodi* e *funzioni*, che costituiscono le *API*$_G$ dell'applicazione *JavaScript*$_G$, è stato scelto di usare **JSDoc 3** poiché consente di *uniformare*, *semplificare* e *velocizzare* notevolmente la stesura della documentazione, grazie alla creazione della stessa in formato *HTML*$_G$, attraverso l'inserimento di commenti direttamente all'interno del codice.

3.2 Processo di verifica

3.2.1 Attività

3.2.1.1 Analisi

3.2.1.1.1 Analisi statica

L'analisi statica è una tecnica di verifica applicabile ai documenti e al codice che verrà impiegata durante tutto lo sviluppo del sistema e che sarà automatizzata il più possibile, mediante gli strumenti descritti in seguito. L'analisi statica applicata al software non necessita che i programmi vengano eseguiti, bensì mira a trovare anomalie ed errori di sintassi, e a fare predizioni sulla qualità e la manutenibilità del codice prodotto. In seguito vengono riportate le metodologie di applicazione dell'analisi statica.

3.2.1.1.1.1 Walkthrough

Questa tecnica di analisi statica consiste in una lettura del documento o del codice, ricercando anomalie ed errori a largo spettro, ovvero senza una conoscenza precisa dei tipi di errori riscontrabili. Il *walkthrough*$_G$ verrà applicato nelle prime fasi dello sviluppo, poiché in tale fase non si possiede ancora una concezione degli errori possibili e più frequenti. Utilizzando questa tecnica, i *Verificatori* avranno il compito di stilare una lista di controllo contenente gli errori rilevati più spesso. Quando tale lista sarà sufficientemente completa, dovrà essere allegata in appendice a questo documento e da quel momento sarà possibile passare all'utilizzo della tecnica di *inspection*$_G$.

3.2.1.1.1.2 Inspection

Questa tecnica di analisi statica consiste nella lettura mirata dei documenti o del codice, mediante l'utilizzo di una lista di controllo contenente gli errori più frequenti[3]. Poiché tale lista verrà ampliata con l'acquisizione di esperienza nella verifica, tale tecnica diverrà sempre più efficace.

[3] La lista degli errori più frequenti si trova in in appendice al documento.

3.2.1.1.2 Analisi dinamica

L'analisi dinamica viene applicata solamente al software prodotto e alle sue componenti, e viene svolta mediante test che verificano il funzionamento di tali componenti e che ne identificano eventuali errori. Per ogni test è necessario definire:

- **Ambiente**: sistema *hardware*$_G$ e software sul quale è pianificata l'esecuzione del test. Inoltre, è necessario specificare uno stato iniziale di partenza per il test;
- **Specifica**: insieme degli input e dei corrispondenti output attesi;
- **Procedure**: specifica di istruzioni su come eseguire il test e come i risultati debbano essere interpretati e analizzati.

Affinché si possano ottenere risultati attendibili, è necessario che i test siano ripetibili, ovvero dato un certo input la loro esecuzione nello stesso ambiente dovrà produrre in output sempre gli stessi risultati.

3.2.1.2 Test

3.2.1.2.1 Test di unità

Per unità di prodotto software si intende la più piccola quantità di software che risulta conveniente verificare singolarmente, tipicamente quella prodotta da un singolo *Programmatore*. Ad esempio solitamente un modulo è parte dell'unità ed il componente invece integra più unità. I test di unità verificano che ogni unità funzioni correttamente, evidenziando errori di implementazione. Questi test verranno effettuati sui moduli base che compongono il software. I test di unità possono essere identificati grazie alla seguente sintassi:

TU[Codice Test]

3.2.1.2.2 Test di integrazione

Questo tipo di test verifica che due o più unità, tipicamente moduli, precedentemente verificati, una volta assemblati funzionino correttamente. I test di integrazione possono aiutare a rilevare errori residui sui moduli e verificano anche che l'eventuale cooperazione di essi con componenti esterni, quali *framework*$_G$ e *librerie*$_G$, non produca anomalie. I test di unità possono essere identificati grazie alla seguente sintassi:

TI[Codice Test]

3.2.1.2.3 Test di sistema

Un test di sistema viene eseguito su un prodotto che si ritenga essere giunto ad una versione definitiva. Viene perciò verificato che il prodotto soddisfi tutti i requisiti imposti. Questo test è quindi la validazione dei prodotti software. I test di sistema possono essere identificati grazie alla seguente sintassi:

TV[Tipo Requisito][Importanza Requisito][Codice Requisito]

Tipo, importanza e codice si riferiscono al requisito di cui verrà testato il soddisfacimento.

3.2.1.2.4 Test di regressione

Questo test consiste nella riesecuzione di tutti i test su un componente che ha subito una modifica. In questo modo si vuole verificare che il resto dei moduli continui a funzionare correttamente. Inoltre, eseguire dei test di regressione, consente di capire quali test sono a rischio di inesattezza in caso di modifiche al codice dei prodotti. I test di regressione possono essere identificati grazie alla seguente sintassi:

TR[Codice Test]

3.2.1.2.5 Test di validazione

Il test di validazione coincide con il collaudo del software in presenza del *Proponente*$_G$, e in caso di esito positivo, questo test determina un grado di maturità del prodotto tale da consentirne il rilascio. I test di validazione possono essere identificati grazie alla seguente sintassi:

TV[Tipo Requisito][Importanza Requisito][Codice Requisito]

Tipo, importanza e codice si riferiscono al requisito di cui verrà testato il soddisfacimento.

3.2.2 Procedure

3.2.2.1 Approvazione di un ticket

La procedura di approvazione di un ticket segue il diagramma di attività riportato in figura 2. L'attività di chiusura di ticket è delegata al *Verificatore*, che:

1. Riceve la notifica di un ticket con stato "Resolved";
2. Effettua una ricerca approfondita per trovare eventuali anomalie ancora presenti, che dà esito negativo;
3. Imposta lo stato del ticket ad "Approved";
4. Imposta lo stato del ticket di verifica a "Resolved".

3.2.2.2 Rigetto di un ticket

La procedura di rigetto di un ticket da parte del *Verificatore* segue il diagramma di attività riportato in figura 2. Il *Verificatore*:

1. Riceve la notifica di un ticket con stato "Resolved";
2. Effettua una ricerca approfondita per trovare eventuali anomalie ancora presenti, che dà esito positivo;
3. Imposta lo stato del ticket a "Rejected";
4. Crea una lista con le anomalie trovate;
5. Invia la lista con le anomalie al *Responsabile di Progetto*;
6. Imposta lo stato del ticket di verifica a "Resolved".

3.2.2.3 Gestione delle anomalie

Al termine di un ticket, il *Verificatore* dovrà procedere alla verifica del lavoro completato. Qualora individuasse delle anomalie, dovrà registrarle nella "*lista delle anomalie*", per consentirne la correzione tramite l'uso di una tecnica di ispezione mirata. Al termine dell'attività di verifica, se sono state individuate anomalie, il *Responsabile di Progetto* dovrà aprire dei ticket per consentirne la risoluzione e la successiva verifica.

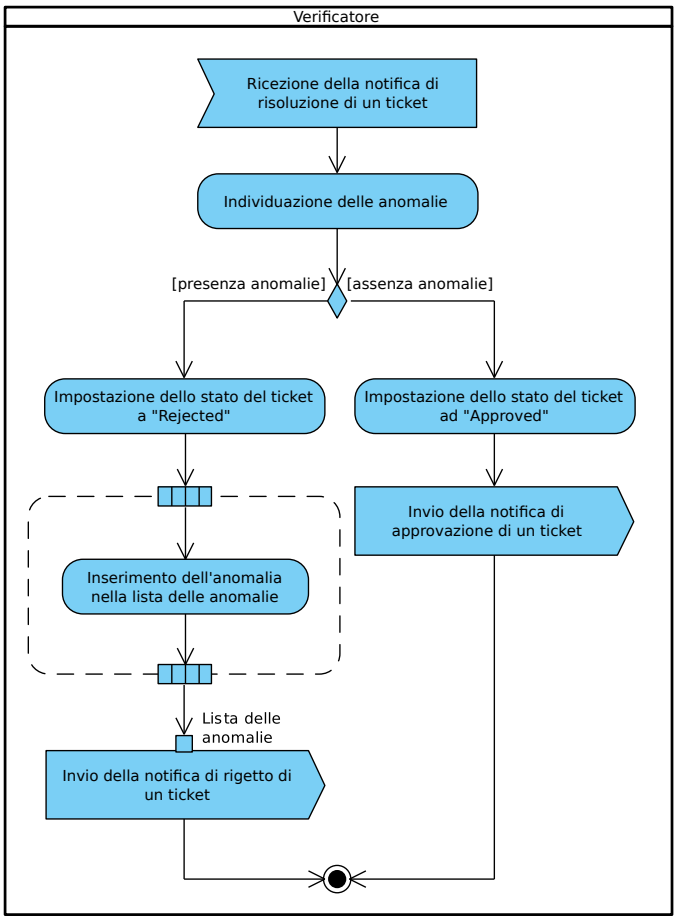


Figura 2: Diagramma di attività - rigetto/approvazione di un ticket

3.2.3 Strumenti

3.2.3.1 Documentazione

3.2.3.1.1 Texmaker

Configurando il controllo ortografico di **Texmaker** vengono mostrati eventuali errori durante la stesura del documento.

3.2.3.1.2 Aspell

Aspell è stato adottato per ottenere ulteriore supporto alla correzione ortografica. Può essere avviato da riga di comando, ma per renderne più semplice l'utilizzo, è stato reso accessibile mediante un *Makefile*.

3.2.3.1.3 Script

Gli script citati nella sezione 3.1.3.3 sono stati sviluppati per automatizzare il controllo dei documenti, ad esempio calcolando l'indice Gulpease e marcando opportunamente i termini presenti nel *Glossario*.

3.2.3.1.4 *PragmaDB*

Sistema di gestione dei requisiti, dei casi d'uso, dei termini del *Glossario*, delle fonti e degli attori creato da alcuni membri gruppo. I requisiti riportati nei documenti verranno direttamente estrapolati dal database *PragmaDB*. Ulteriori dettagli sono disponibili nella sezione 2.1.3.1.

3.2.3.2 Codice

3.2.3.2.1 Strumenti di analisi statica

3.2.3.2.1.1 *JSHint*$_G$

Strumento per la rilevazione di errori e problemi nel codice *JavaScript*$_G$, attenendosi a regole di codifica definite. Nel caso specifico di questo *progetto*$_G$, ***JSHint***$_G$ verrà utilizzato da riga di comando e installato come modulo per Node.js;

3.2.3.2.1.2 *CSSLint*$_G$

Strumento simile a ***JSHint***$_G$, verrà impiegato però per l'analisi di codice *CSS*$_G$. Anch'esso verrà installato come modulo per Node.js ed eseguito da riga di comando;

3.2.3.2.1.3 *W3C*$_G$ Markup Validator Service

Validatore *W3C*$_G$ che segnala eventuali errori di sintassi nel codice *HTML*$_G$. L'indirizzo *web*$_G$ di riferimento è il seguente: `validator.w3.org`;

3.2.3.2.1.4 complexity-report

Applicazione, disponibile come modulo per Node.js, che misura metriche riguardanti codice *JavaScript*$_G$, in particolare:

- *Complessità ciclomatica*: misura la complessità di funzioni, metodi o classi di un programma;
- *Rapporto linee di commento su linee di codice*: misura il rapporto tra linee di codice e linee di commento;
- *Dipendenze*: il numero di dipendenze interne o esterne con altre classi o moduli;
- *Chiamate annidate di metodi e funzioni*: il numero di chiamate innestate di funzioni e metodi all'interno di altre funzioni;
- *Indice di manutenibilità*: valore che indica quanto il codice prodotto è mantenibile.

3.2.3.2.2 Strumenti di analisi dinamica

3.2.3.2.2.1 *Google Chrome*$_G$ DevTools

Gli strumenti per gli sviluppatori forniti da *Google Chrome*$_G$ consentono di effettuare il profiling del software, e monitorare quindi l'utilizzo della CPU e della memoria da parte di oggetti e funzioni *JavaScript*$_G$ utilizzati dall'applicazione *web*$_G$.

3.2.3.2.2.2 *Karma*$_G$

Si tratta di uno strumento per eseguire test d'unità, che verrà configurato per eseguire test specifici riguardanti gli script *JavaScript*$_G$. Poiché si tratta di un modulo per Node.js, è eseguibile da riga di comando ed è integrabile direttamente in ***WebStorm***$_G$.

4 Processi organizzativi

4.1 Processo di gestione

4.1.1 Attività

4.1.1.1 Comunicazioni

4.1.1.1.1 Comunicazioni interne

Per le comunicazioni riguardanti un esigua cerchia di componenti del gruppo è consigliato, ma non vincolato, l'uso di Google *Hangouts*$_G$. Ad ogni modo i diretti interessati potranno usare il mezzo di comunicazione che riterranno più idoneo alle loro esigenze. L'utilizzo di telefonate è sconsigliato in quanto troppo invasivo. Resta comunque utilizzabile nel caso in cui sia necessaria una risposta immediata o nel caso in cui i due membri si siano precedentemente accordati. Se durante questo tipo di discussioni emergono informazioni riguardanti l'avanzamento del *progetto*$_G$, che interessano tutti i componenti del gruppo, è necessario scrivere un verbale interno da inserire nell'apposita cartella presente in *Google Drive*$_G$.

4.1.1.1.1.1 Comunicazioni via mailing list

L'utilizzo della *mailing list*$_G$ `sweteam201415@yahoogroups.com` è riservato per le comunicazioni che riguardano tutti i componenti del gruppo e che devono essere archiviate in modo da potervi accedere efficacemente.

4.1.1.1.1.2 Comunicazioni via *chat*$_G$

Per le comunicazioni che riguardano tutto il gruppo o solo alcuni suoi elementi, si è scelto di utilizzare Google *Hangouts*$_G$, il quale fornisce sia un servizio di *instant messaging*$_G$, che un servizio di conferenze e videoconferenze. La *chat*$_G$ di gruppo è da utilizzare solamente per discutere di argomenti inerenti al *progetto*$_G$, che richiedano una scelta di gruppo da prendere a breve termine. Poiché recuperare informazioni precise dalla *chat*$_G$ è un operazione troppo costosa e disordinata, al termine di ogni discussione un volontario scriverà un verbale interno, che sarà salvato nella cartella `Verbali` di *Google Drive*$_G$.

4.1.1.1.2 Comunicazioni esterne

Per le comunicazioni esterne e per le registrazioni ai servizi utili per il gruppo è stata creata una casella di posta elettronica:

`pragma.swe@gmail.com`

Questo indirizzo dev'essere l'unico canale di comunicazione esistente tra il gruppo di lavoro e l'esterno. Solo il *Responsabile di Progetto* può accedere ed inviare mail da questa casella di posta. Al fine di mantenere informati tutti i membri del gruppo le mail in arrivo verranno automaticamente pubblicate nella *mailing list*$_G$.

4.1.1.2 Riunioni

4.1.1.2.1 Riunioni interne

Il *Responsabile di Progetto* ha il compito di convocare le riunioni che si svolgono tra i soli membri del gruppo. Egli deve stilare l'ordine del giorno e individuare la data di svolgimento dell'incontro, segnalandola al *team*$_G$ tramite *mailing list*$_G$, con un preavviso di almeno 2 giorni. Ogni membro del gruppo dovrà dare conferma tempestiva della propria disponibilità a partecipare all'incontro. Potrà eventualmente manifestare l'impossibilità di parteciparvi, fornendo le adeguate motivazioni, entro e non

oltre le ore 12 del giorno successivo alla ricezione dell'invito, in modo da consentire al *Responsabile di Progetto* un'eventuale spostamento di data della riunione. Valgono le medesime regole anche per gli incontri destinati a un insieme ristretto di componenti del $team_G$.
La mail di convocazione delle riunioni interne deve contenere:

- **Oggetto**: Convocazione della riunione interna n. xxx (dove xxx rappresenta il numero crescente della riunione);
- **Corpo**:
  - Data e ora previste;
  - Luogo di svolgimento della riunione;
  - Ordine del giorno;
  - Durata prevista.

Alla fine di ogni riunione, un componente del $team_G$, a discrezione del *Responsabile di Progetto*, avrà il compito di redigerne il verbale.

4.1.1.2.2 Riunioni esterne

Gli incontri esterni, con il $Proponente_G$ o con il $Committente_G$, vengono concordati dal *Responsabile di Progetto* che, prima di prendere accordi, dovrà assicurarsi della presenza dei componenti del gruppo interessati all'incontro. Ogni membro del $team_G$ può richiedere al *Responsabile di Progetto* un incontro esterno, e tale richiesta dovrà essere motivata e approvata prima che possa essere fissato un incontro. Le informazioni quali *data*, *ora* e *luogo* dovranno essere comunicate al gruppo. Gli assenti all'incontro potranno essere scelti, con alta probabilità, per redigere il verbale o eseguirne la verifica. Almeno una fase tra *stesura*, *verifica* e *approvazione* del documento dovrà essere compito di membri presenti all'incontro. I verbale relativi a riunioni esterne dovrà sempre essere resi disponibili a tutto il $team_G$, e inviati al $Committente_G$ e ad altre eventuali entità esterne partecipanti.

4.1.1.3 Ticketing

Per consentire un'agevole assegnazione dei compiti ai vari componenti del $team_G$ e un monitoraggio istantaneo del loro stato di avanzamento, il team ha deciso di appoggiarsi ad un sistema di $ticketing_G$ (per i dettagli si veda 4.1.4.5).

4.1.2 Procedure

4.1.2.1 Apertura di un ticket

La procedura di apertura di un ticket segue il diagramma di attività riportato in figura 3, e la sua applicazione è delegata al *Responsabile di Progetto*, che:

1. Seleziona la tipologia: *task*, *feature*, bug_G o *step*;
2. Seleziona la categoria specifica: *documentazione* o *verifica*;
3. Definisce l'*oggetto*;
4. Definisce una *descrizione*;
5. Imposta lo *stato* a "*New*";
6. Seleziona il livello di *priorità*: *low*, *normal*, *high*, *urgent* o *immediate*;
7. Indica una *stima del tempo* (in ore) necessario per portarlo a termine;
8. Definisce una *lista di attività* da compiere (facoltativo);
9. Stabilisce l'*attività principale* da eseguire (facoltativo);

10. Definisce la data di *inizio* (facoltativo);
11. Definisce la data di *scadenza* (facoltativo);
12. Sceglie l'*assegnatario* (facoltativo);
13. Indica degli *osservatori* (facoltativo).

L'assegnatario e gli osservatori del ticket ricevono una notifica via mail che li informa della sua apertura.
N.B.: qualora non venga scelto alcun assegnatario, dovrà essere indicato almeno un osservatore.

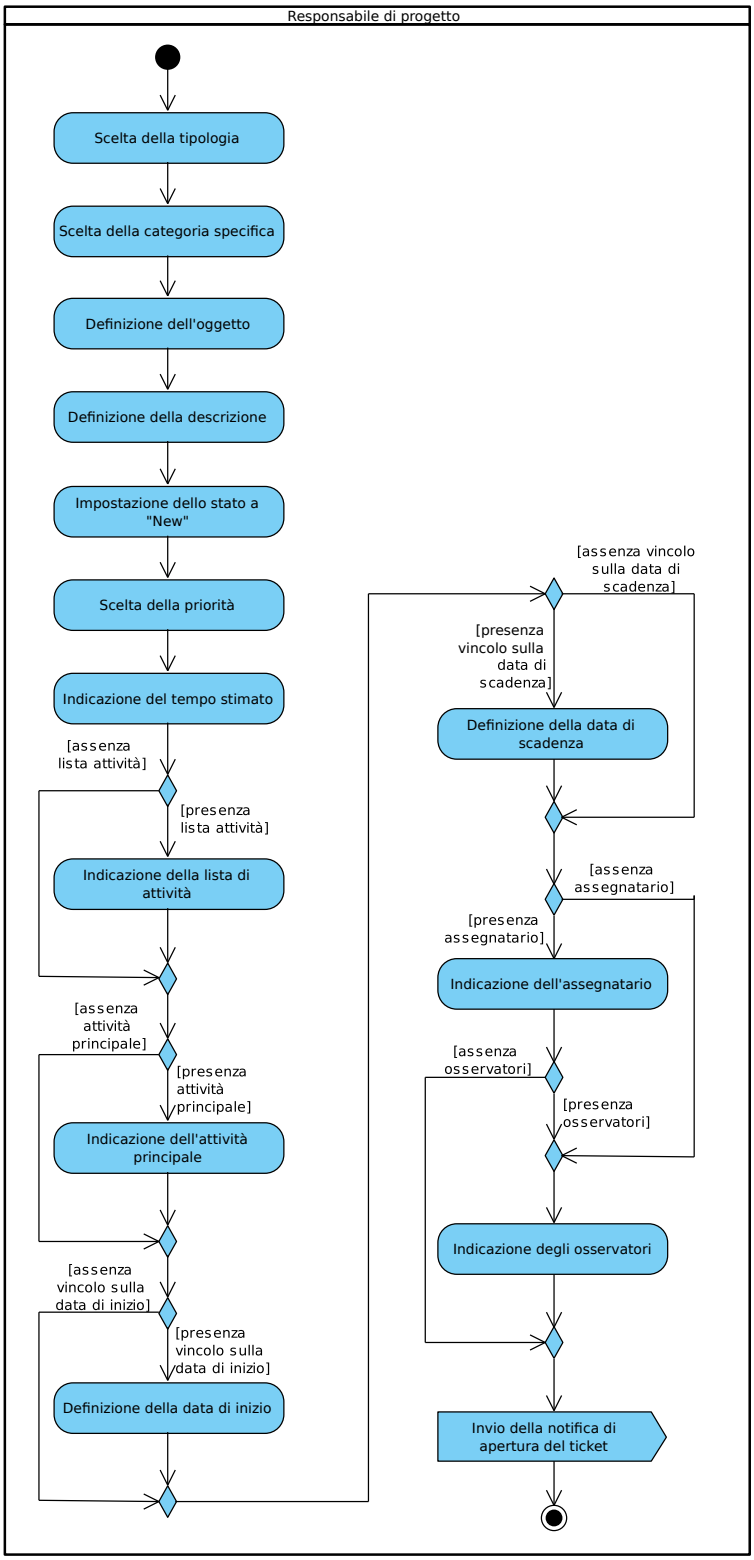


Figura 3: Diagramma di attività - apertura di un ticket

4.1.2.2 Rigetto di un ticket

La procedura di rigetto di un ticket da parte del *Responsabile di Progetto* segue il diagramma di attività riportato in figura 4. Il *Responsabile di Progetto*:

1. Riceve la notifica di un ticket con stato “Approved”;
2. Effettua una rapida ricerca per trovare eventuali anomalie non catturate dal *Verificatore*, che dà esito positivo;
3. Crea una lista con le anomalie trovate;
4. Imposta lo stato del ticket a “Rejected” e lo commenta con il link alla lista delle anomalie;
5. Apre i ticket per la correzione delle anomalie;
6. Apre un ticket per la verifica della correzione delle anomalie.

4.1.2.3 Chiusura di ticket

La procedura di chiusura di ticket segue il diagramma di attività riportato in figura 4. L’attività di chiusura di ticket è delegata al *Responsabile di Progetto*, che:

1. Riceve la notifica di un ticket con stato “Approved”;
2. Effettua una rapida ricerca per trovare eventuali anomalie non catturate dal *Verificatore*, che dà esito negativo.

oppure:

1. Riceve la notifica di un ticket con stato “Rejected”, accompagnata dalla lista delle anomalie trovate dal verificatore;
2. Apre i ticket per la correzione delle anomalie;
3. Apre un ticket per la verifica della correzione delle anomalie.

Infine, il *Responsabile di Progetto* imposta lo stato del ticket a “Closed”.

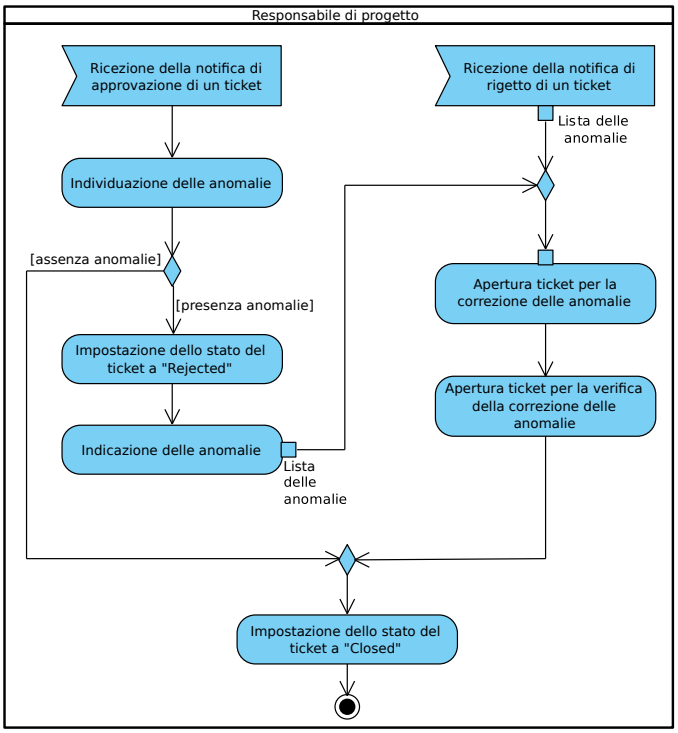


Figura 4: Diagramma di attività - rigetto/chiusura di ticket

4.1.2.4 Rilevazione dei rischi

In ciascuna fase del *progetto*$_G$, il *Responsabile di Progetto* avrà il compito di individuare i rischi indicati nel *Piano di Progetto*. Nel caso si verifichino problematiche non previste, il *Responsabile di Progetto* dovrà includerle nell'analisi dei rischi. Complessivamente la procedura di rilevazione dei rischi prevede i seguenti passi:

1. Rilevazione dei problemi non calcolati;
2. Analisi e classificazione dei nuovi rischi individuati;
3. Pianificazione di controllo dei nuovi rischi individuati;
4. Definizione di contromisure per i nuovi rischi individuati.

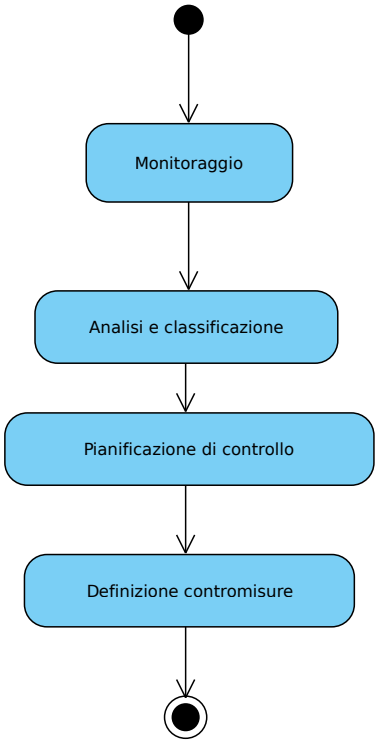


Figura 5: Diagramma di attività - rilevazione dei rischi

4.1.3 Norme

4.1.3.1 Regole generali di ticketing

Ogni ticket possiede una struttura standard e deve essere conforme alle seguenti regole:

- L'oggetto del ticket è un insieme di parole chiave che devono fornire un'idea generale della mansione da completare e devono semplificarne la ricerca;
- La descrizione dev'essere un elenco esaustivo dei compiti da svolgere;
- Nel caso non si tratti di un ticket di verifica, il *Responsabile di Progetto* dovrà preoccuparsi di creare un ulteriore ticket per la verifica di tale compito, da assegnare ad un differente membro del $team_G$;
- Tra ticket con la medesima priorità, dev'essere data maggiore importanza a ticket con scadenza più vicina alla data attuale.

Da parte dei membri del $team_G$ è richiesta una prassi di utilizzo riguardo ai ticket, che aiuti il *Responsabile di Progetto* a supervisionare le attività correnti. Vengono quindi elencate le norme relative allo stato del ticket:

- I ticket presi in carico saranno caratterizzati dallo stato "Assigned";
- I ticket interrotti per ricezione di ticket prioritari saranno caratterizzati dallo stato "Blocked";
- I ticket completati e in attesa di essere verificati saranno caratterizzati dallo stato "Resolved";
- I ticket rigettati dal *Verificatore* o dal *Responsabile di Progetto* saranno caratterizzati dallo stato "Rejected";
- I ticket verificati dal *Verificatore* saranno caratterizzati dallo stato "Approved";

- I ticket approvati dal *Responsabile di Progetto* saranno caratterizzati dallo stato “Closed”.

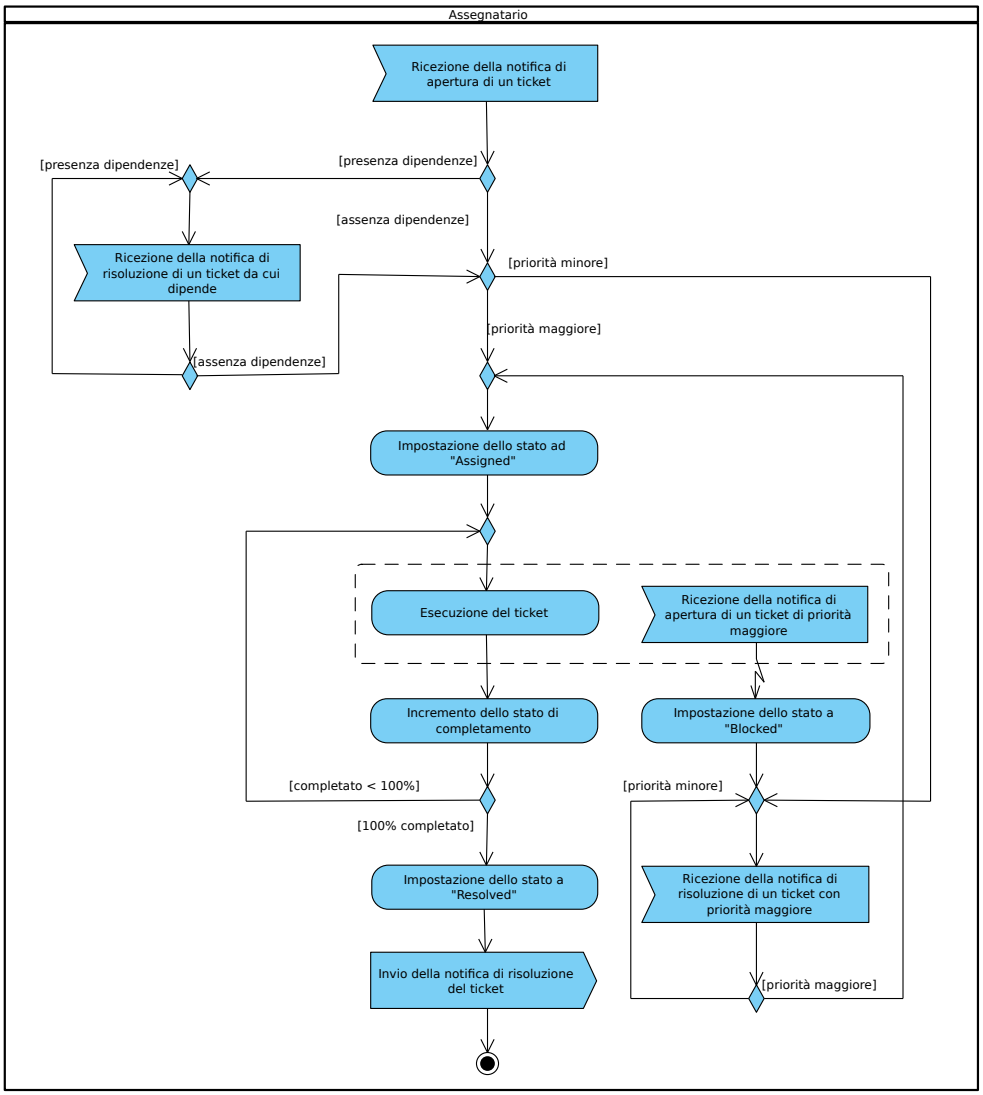


Figura 6: Diagramma di attività - lavorazione di un ticket

4.1.3.2 Protocollo di utilizzo dei ticket

La procedura di lavorazione di un ticket da parte dell’assegnatario segue il diagramma di attività riportato in figura 6.

1. Ogni membro, ricevuto almeno un ticket attivo, ne sceglierà uno prediligendo quelli a priorità maggiore e con scadenza più imminente;
2. Ogni utente non potrà avere più ticket in lavorazione contemporaneamente;
3. Nel caso in cui un utente ricevesse un ticket a scadenza più imminente rispetto a quello attualmente in lavorazione, dovrà sospendere il ticket corrente e iniziare a lavorare al ticket prioritario;
4. Una volta completato un ticket e dopo la sua verifica, se il *Verificatore* non ha individuato alcuna anomalia, allora potrà impostare lo stato del ticket ad “Approved”. In caso contrario, seguirà la prassi descritta nella sezione 3.2.2.3, che è dedicata alle anomalie.

Fasi	Giacomo Manzoli	Andrea Ongaro	Massimiliano Baruffato	Daniele Marin	Gianmarco Midena	Stefano Munari	Fabio Vedovato
Ammissione al progetto	Responsabile Amministratore Verificatore	Amministratore Analista	Amministratore Analista	Responsabile Amministratore Analista Verificatore	Analista Verificatore	Analista Verificatore	Responsabile Analista Verificatore
Consolidamento dei requisiti	Analista Verificatore	Responsabile	Verificatore	Analista	Amministratore Verificatore	Analista Verificatore	Analista Verificatore
Progettazione dell'architettura	Progettista Verificatore	Progettista Verificatore	Progettista	Progettista Verificatore	Progettista Verificatore	Responsabile Amministratore Progettista Verificatore	Analista Verificatore
Consolidamento dell'architettura	Amministratore Analista Verificatore	Progettista Verificatore	Responsabile Progettista Verificatore	Progettista Verificatore	Progettista	Progettista Verificatore	Progettista
Realizzazione del prodotto	Progettista Programmatore Verificatore	Programmatore Verificatore	Analista Progettista Programmatore	Programmatore Verificatore	Responsabile Analista Verificatore	Progettista Programmatore Verificatore	Progettista Verificatore
Collaudo finale	Progettista Verificatore	Analista Verificatore	Progettista Verificatore	Responsabile Verificatore	Programmatore Verificatore	Amministratore Verificatore	Programmatore Verificatore

Tabella 2: Rotazione ruoli di progetto.

4.1.3.3 Ruoli di progetto

I ruoli ricoperti dai vari componenti del gruppo varieranno ad ogni fase in modo tale che, alla fine del $progetto_G$, ogni componente abbia rivestito ogni ruolo almeno una volta. Tali ruoli sono stati stabiliti dal *Responsabile di Progetto* e saranno turnati secondo quanto previsto dalla seguente tabella. Deve valere sempre la seguente condizione: *un componente che abbia preso come incarico la stesura di uno specifico documento o la scrittura di uno particolare file di codice sorgente non potrà, in alcun modo, risultare come* Verificatore *dello stesso*.
Questa norma garantisce l'assenza di conflitto di interessi nello svolgimento delle successive attività di verifica e di approvazione all'interno del gruppo.

4.1.3.3.1 *Responsabile di Progetto*

Il *Responsabile di Progetto* ha il compito di mantenere i contatti con gli enti esterni al gruppo, e detiene il potere decisionale, quindi ha responsabilità su:

- Pianificazione, coordinamento e controllo delle attività;
- Gestione delle risorse;
- Analisi e gestione dei rischi;
- Approvazione dei documenti;
- Approvazione dell'offerta economica;
- Redazione del *Piano di Progetto* e collabora alla stesura del *Piano di Qualifica*, in particolare nella sezione relativa alla pianificazione.

Si assicura che le attività di verifica vengano svolte sistematicamente seguendo le *Norme di Progetto*, vengano rispettati i ruoli e le competenze assegnate nel *Piano di Progetto*, non vi siano conflitti di interesse tra redattori e *Verificatori*. È l'unico a poter decidere l'approvazione di un documento e a sancirne la distribuzione. Ha inoltre l'incarico di gestire la creazione e l'assegnazione dei ticket delle macro-fasi e di assegnare ad un membro del gruppo il ruolo di responsabile di quest'ultima.

4.1.3.3.2 *Amministratore*

L'*Amministratore* è il responsabile dell'ambiente di lavoro e le sue principali mansioni sono:

- Ricerca di strumenti che possano automatizzare qualsiasi compito automatizzabile;
- Risoluzione dei problemi legati alle difficoltà di gestione e controllo dei processi e delle risorse, poiché la loro risoluzione richiede l'adozione di strumenti software adatti;
- Controllo delle versioni e delle configurazioni del prodotto;
- Gestione dell'archiviazione e del *versionamento*$_G$ della documentazione di *progetto*$_G$, attraverso l'uso di database;
- Scelta di procedure e strumenti per il monitoraggio e la segnalazione per il controllo qualità;
- Redazione delle *Norme di Progetto*, dove spiega e norma l'utilizzo degli strumenti, redige la sezione del *Piano di Qualifica* dove vengono descritti strumenti e metodi di verifica.

4.1.3.3.3 *Analista*

L'*Analista* è il responsabile delle attività di analisi e le sue responsabilità principali sono:

- Produzione di una specifica di *progetto*$_G$ comprensibile e motivata in ogni suo punto;
- Comprensione esaustiva della natura e della complessità del problema;
- Redazione dello *Studio di Fattibilità*, dell'*Analisi dei Requisiti* e parte del *Piano di Qualifica*. Partecipa alla redazione del *Piano di Qualifica* in quanto conosce l'ambito del *progetto*$_G$ ed ha chiari i livelli di qualità richiesta e le procedure da applicare per ottenerla.

4.1.3.3.4 *Progettista*

Il *Progettista* è il responsabile delle attività di progettazione e le sue responsabilità sono:

- Produzione di una soluzione attuabile che sia comprensibile, robusta e semplice;
- Scelta dei *design pattern*$_G$ da impiegare;
- Scelta degli aspetti progettuali e tecnologici che rendano il prodotto facilmente manutenibile;
- Scelta degli aspetti progettuali e tecnologici che rendano il prodotto modulare entro i limiti del possibile;
- Redazione della *Specifica Tecnica*, della *Definizione di Prodotto* e delle sezioni inerenti le metriche di verifica della programmazione del *Piano di Qualifica*.

4.1.3.3.5 *Programmatore*

Il *Programmatore* è il responsabile delle attività di codifica e delle componenti di ausilio necessarie per l'esecuzione delle prove di verifica e validazione. Le sue responsabilità sono:

- Implementazione rigorosa delle soluzioni descritte dal *Progettista*, da cui seguirà la realizzazione del prodotto;
- Scrittura di codice documentato, versionato, manutenibile e che rispetti gli standard stabiliti per la scrittura del codice;
- Implementazione dei test sul codice scritto, necessari per le prove di verifica e validazione;
- Redazione del *Manuale Utente*.

4.1.3.3.6 *Verificatore*

Il *Verificatore* il è responsabile delle attività di verifica. Effettua la verifica dei documenti utilizzando gli strumenti e i metodi proposti dal *Piano di Qualifica* e attenendosi a quanto descritto nelle *Norme di Progetto*. Le responsabilità di tale ruolo sono:

- Controllo di conformità dell'attuazione delle attività con le norme stabilite;
- Controllo di conformità di ogni stadio del ciclo di vita del prodotto;
- Redazione della sezione del *Piano di Qualifica*, che illustra l'esito e la completezza delle verifiche e delle prove effettuate.

4.1.4 Strumenti

4.1.4.1 Google Drive

Google Drive$_G$ è il servizio di *webstorage*$_G$ usato dal gruppo per condividere documenti con le seguenti caratteristiche:

- Non necessitano di controllo di versione;
- Contengono informazioni utili allo sviluppo del *progetto*$_G$, ma non fanno parte dei documenti di *progetto*$_G$;
- Possono essere acceduti mediante il semplice utilizzo del *web*$_G$ *browser*$_G$.

Google Drive$_G$ va inteso come strumento di supporto allo sviluppo del *progetto*$_G$, sia per la documentazione che per il software. Consente, inoltre, ai membri del gruppo di lavorare in modo collaborativo sui documenti.

4.1.4.2 Google Calendar

Google Calendar$_G$ viene utilizzato come calendario condiviso dal gruppo, al fine di gestire le risorse umane. È fondamentale per la segnalazione dei giorni di reperibilità dei membri del *team*$_G$, per semplificare la scelta delle date delle riunioni, e per memorizzare e notificare le date degli appuntamenti del gruppo.

4.1.4.3 Git

Il software di *versionamento*$_G$ scelto è ***Git***$_G$. Il gruppo ha preso in considerazione come altra alternativa **SVN**, ma ha ritenuto la scelta di ***Git***$_G$ più efficace per i seguenti motivi:

- **Flessibilità**: essendo un *repository*$_G$ distribuito, ***Git***$_G$ consente di lavorare in locale ed avere *commit* e *revert* locali;
- **Velocità delle operazioni**: quasi tutte le operazioni sono effettuate sulla copia locale, eliminando problemi legati alla *latenza*$_G$ della rete;
- **Esperienza del gruppo**: tutti i componenti del gruppo hanno utilizzato in passato ***Git***$_G$.

4.1.4.4 Bitbucket

Per lo svolgimento del *progetto*$_G$ sono stati creati due *repository*$_G$ *privati* nello spazio di *hosting*$_G$ offerto da ***Bitbucket***$_G$. Affinché ogni membro del gruppo possa sincronizzarsi con i *repository*$_G$ privati è necessario che ognuno abbia un account ***Bitbucket***$_G$. L'*Amministratore* si occupa di attribuire permessi di lettura e scrittura ad ogni componente del gruppo. I due *repository*$_G$ creati sono i seguenti:

- **pragmadocs.git**: contiene tutti gli elementi necessari alla stesura della documentazione, quindi i sorgenti LaTeX e gli script utilizzati per la loro corretta compilazione. Il *repository*$_G$ è disponibile agli indirizzi:

https://nome_utente@bitbucket.org/gmidena/pragmadocs.git
ssh://git@bitbucket.org:gmidena/pragmadocs.git

- **pragmasrc.git**: conterrà i file dell'applicazione. Il *repository*$_G$ è disponibile agli indirizzi:

https://nome_utente@bitbucket.org/gmidena/pragmasrc.git
ssh://git@bitbucket.org:gmidena/pragmasrc.git

Al termine della stesura di un documento, i *Verificatori* lavoreranno e apporteranno modifiche in parallelo al resto dei componenti del gruppo, utilizzando un opportuno branch creato per la verifica. Al termine della verifica, verrà effettuato il merge delle modifiche.
Ad ogni revisione, la versione dei documenti verrà identificata mediante la creazione di un tag.
Le suddivisioni del *repository*$_G$ dedicato all'applicazione, **pragmasrc.git**, verranno trattate durante la fase di **Progettazione dell'architettura**.

4.1.4.5 Redmine

Il software di gestione del *progetto*$_G$ scelto dal gruppo è **Redmine**, che fornisce i seguenti servizi:

- Traccia il *diagramma di Gantt*$_G$ delle attività;
- Calendario per l'organizzazione di attività e compiti;
- Possibilità di associare i *repository*$_G$ e quindi visualizzarne la struttura ed i contenuti direttamente dall'interfaccia *web*$_G$;
- Interfaccia *web*$_G$ semplice e curata;
- Gestione dei ticket personalizzabile e molto flessibile;
- Possibilità di osservare la differenza tra il tempo stimato e quello effettivo per il completamento di una attività.

Le piattaforme alternative prese in visione sono:

- **YouTrack**;
- **Gitorious**.

Entrambe però si sono mostrate meno complete, personalizzabili e curate di **Redmine**.

4.1.4.6 Microsoft Project

Come software di supporto alla pianificazione del *progetto*$_G$ si è scelto di utilizzare Microsoft Project 2010, questo perché in grado di fornire tutte le funzionalità necessarie in modo intuitivo e completo. Per l'utilizzo di Microsoft Project è possibile utilizzare la licenza gratuita offerta agli studenti tramite l'Università degli Studi di Padova[4].
Alternativamente a Microsoft Project è possibile utilizzare ProjectLibre, software che offre funzionalità analoghe ma *multipiattaforma*$_G$. La scelta del software da utilizzare è a discrezione del *Responsabile di Progetto*, dal momento che è possibile convertire un *progetto*$_G$ di Microsoft Project ad un *progetto*$_G$ di ProjectLibre e viceversa mediante l'utilizzo di file *XML*$_G$.

4.1.4.7 Jenkins

Il software utilizzato per l'integrazione continua è **Jenkins CI**. Configurato per allacciarsi ai *repository*$_G$ del *progetto*$_G$, consente di visualizzare lo stato del codice prodotto, pianificare compilazioni ed eseguire script per i test. Nel caso specifico di questo *progetto*$_G$, è stato configurato per eseguire la compilazione dei documenti LaTeX e nelle fasi successive verrà aggiunta la possibilità di effettuare test specifici sul codice prodotto.

[4] http://msdnaa.studenti.math.unipd.it/2011/.

Appendice A Lista degli errori frequenti

- Norme stilistiche:
  - Mancato rispetto delle norme relative a elenchi puntati e numerati;
  - Mancato utilizzo delle macro;
  - Mancato rispetto delle norme relative alla redazione del diario;
  - Mancata precisazione della versione nei riferimenti a documenti esterni.
- Linguistica:
  - Mancato rispetto degli accenti: uso dell'accento acuto quando è richiesto quello grave, in particolare "è" confusa con "é";
  - "HTML", "URL", "URI" sono acronimi, vanno con tutte le lettere maiuscole;
  - Periodi eccessivamente lunghi, che complicano la lettura;
  - Errato uso delle doppie, in particolare raddoppio della lettera z davanti a parole che finiscono in "-ione";
  - Mancato rispetto della terza persona singolare del congiuntivo presente;
  - Mancato uso dei pronomi, per evitare la ripetizione di una componente della frase;
  - Non si scrive "Lo scopo è quello di definire . . . " ma "Lo scopo è definire . . . ";
  - Uso errato delle persone dei verbi, in particolare persone singolari usate al posto di quelle plurali, a causa di un'errata individuazione del relativo soggetto all'interno delle frasi;
  - Uso errato dei pronomi relativi, in particolare "il/di/in/a cui" usati al posto di "i/dei/nei/ai quali" e "le/delle/nelle/alle quali".
- LaTeX:
  - Per inserire caratteri speciali, quali "$" e "&", è necessario inserire il prefisso "\", ad esempio "\$" e "\&";
  - La maiuscola della lettera "è" si scrive "\`E" oppure "\`{E}".

Appendice B Screenshot Redmine

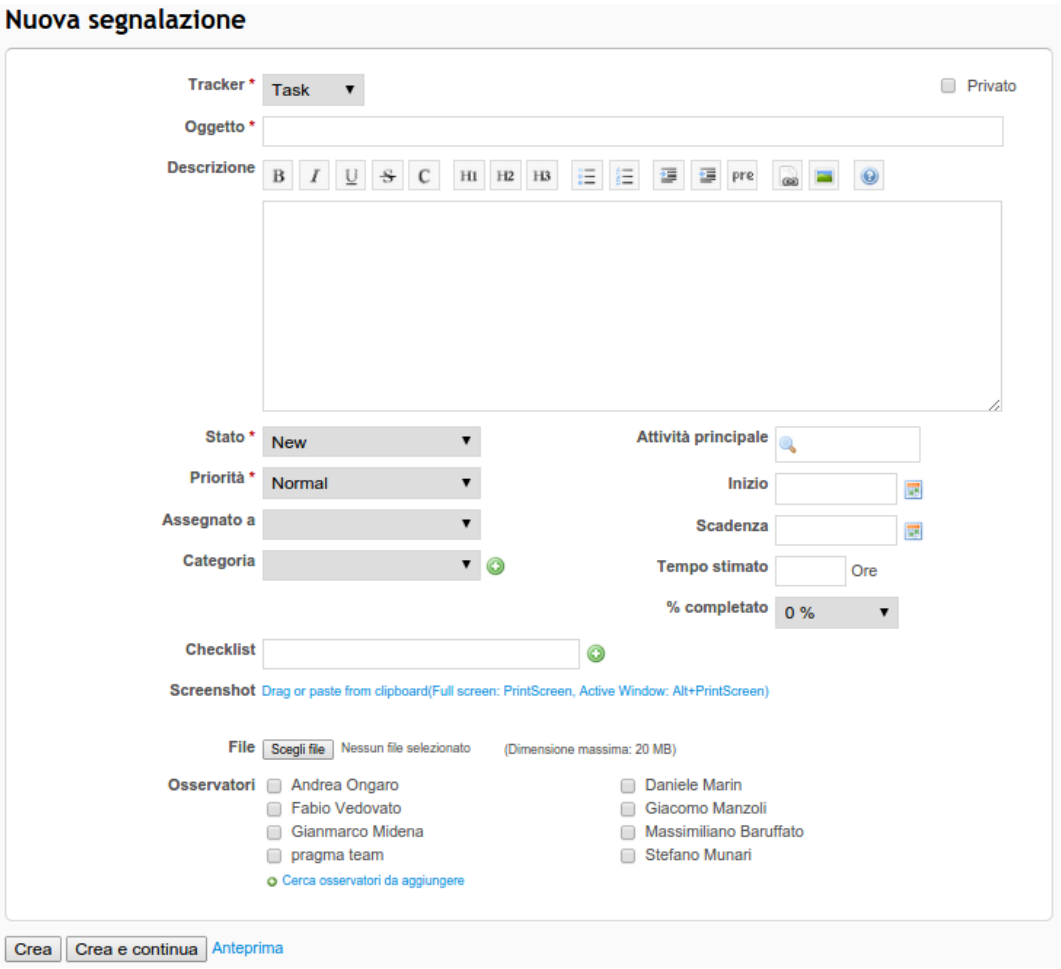


Figura 7: Form di creazione di un ticket

Appendice C Screenshot *PragmaDB*

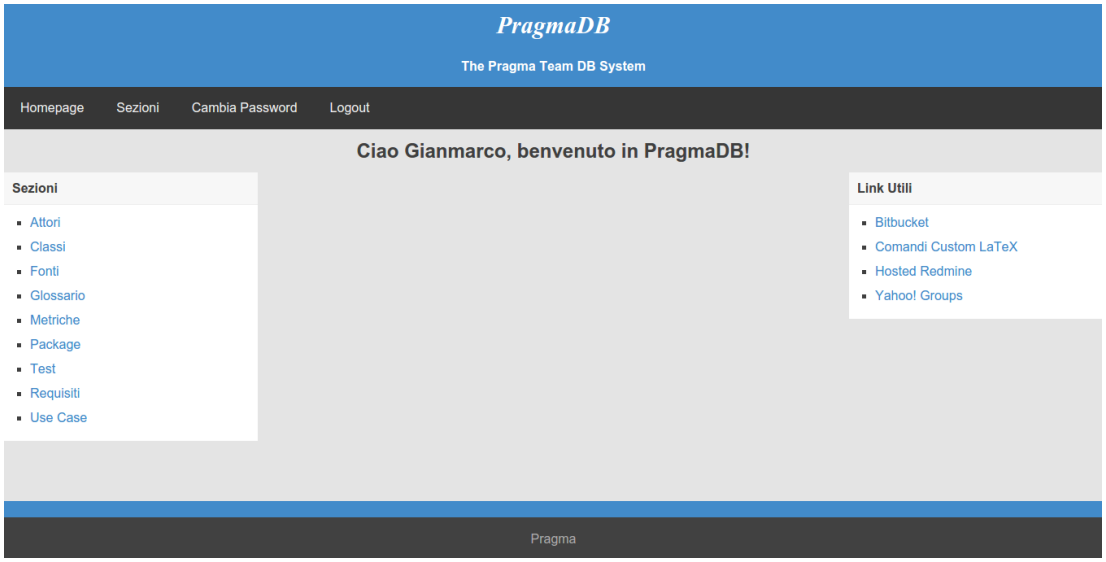


Figura 8: *PragmaDB* - pagina principale

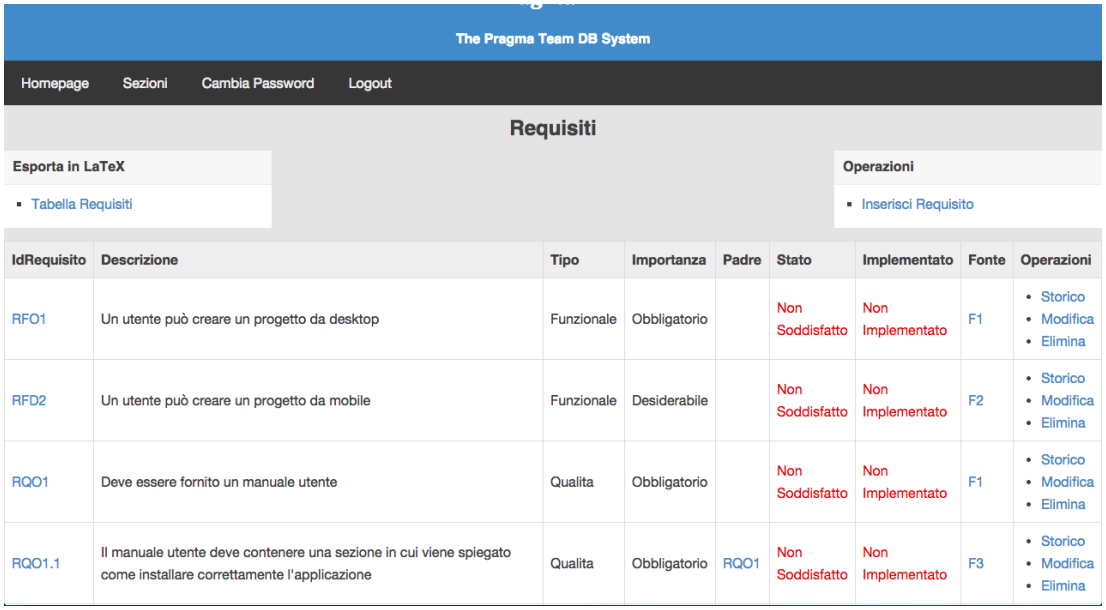


Figura 9: *PragmaDB* - pagina dei requisiti

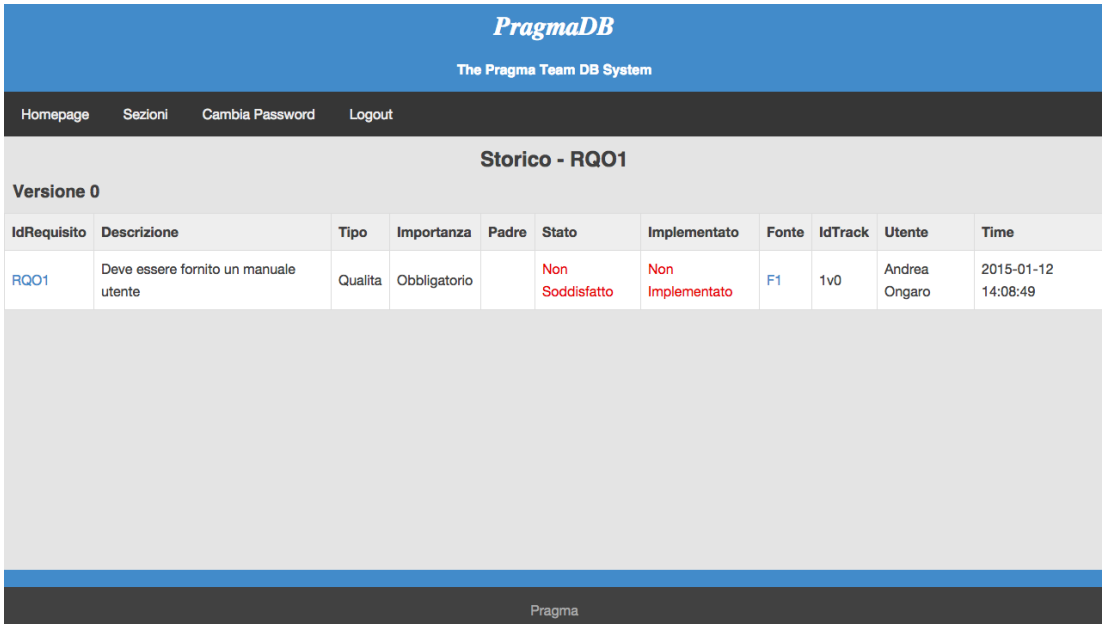


Figura 10: *PragmaDB* - pagina dello storico di uno specifico requisito

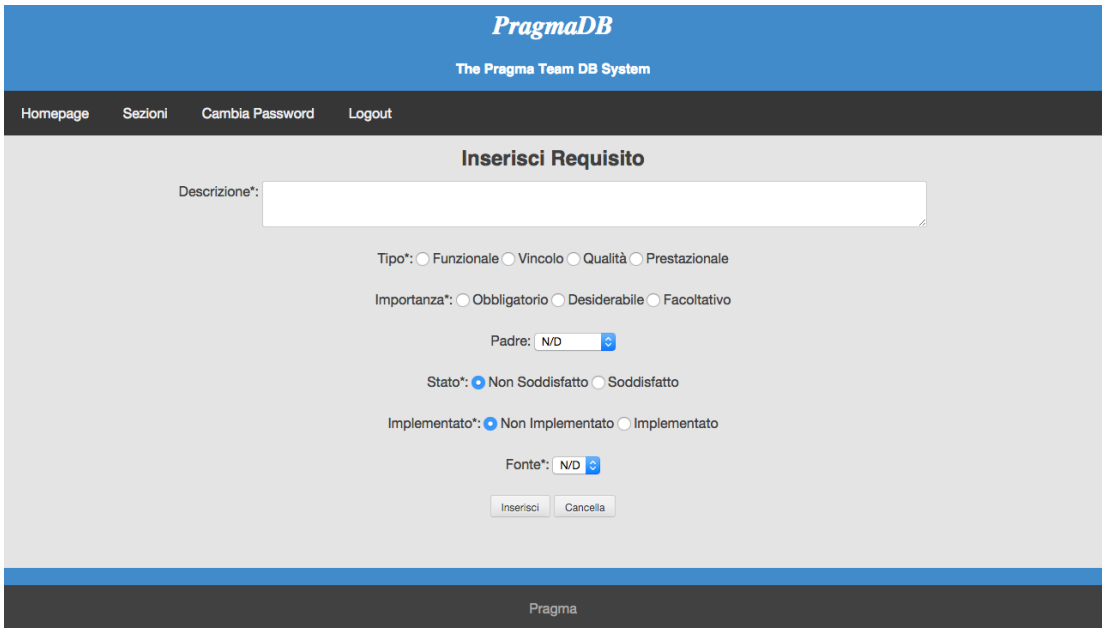


Figura 11: *PragmaDB* - pagina di inserimento di un requisito

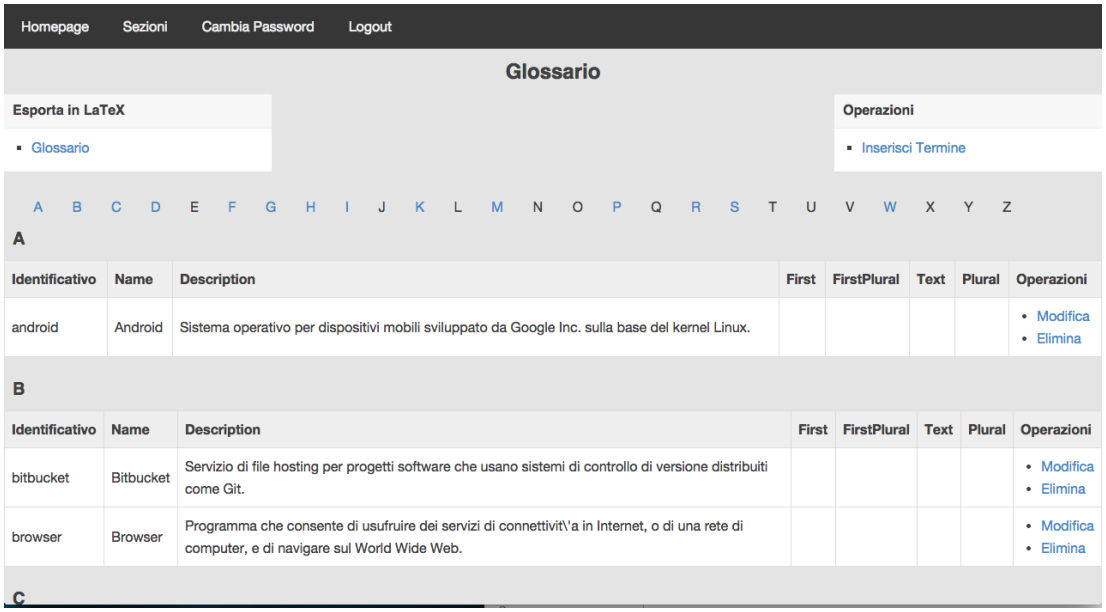


Figura 12: *PragmaDB* - pagina del *Glossario*

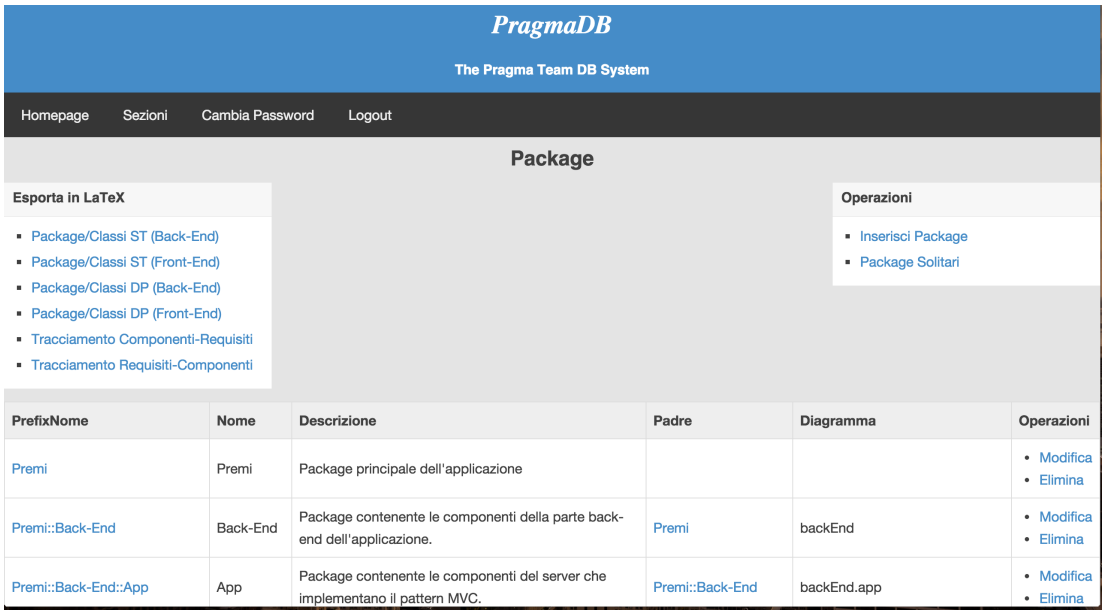


Figura 13: *PragmaDB* - pagina dei package

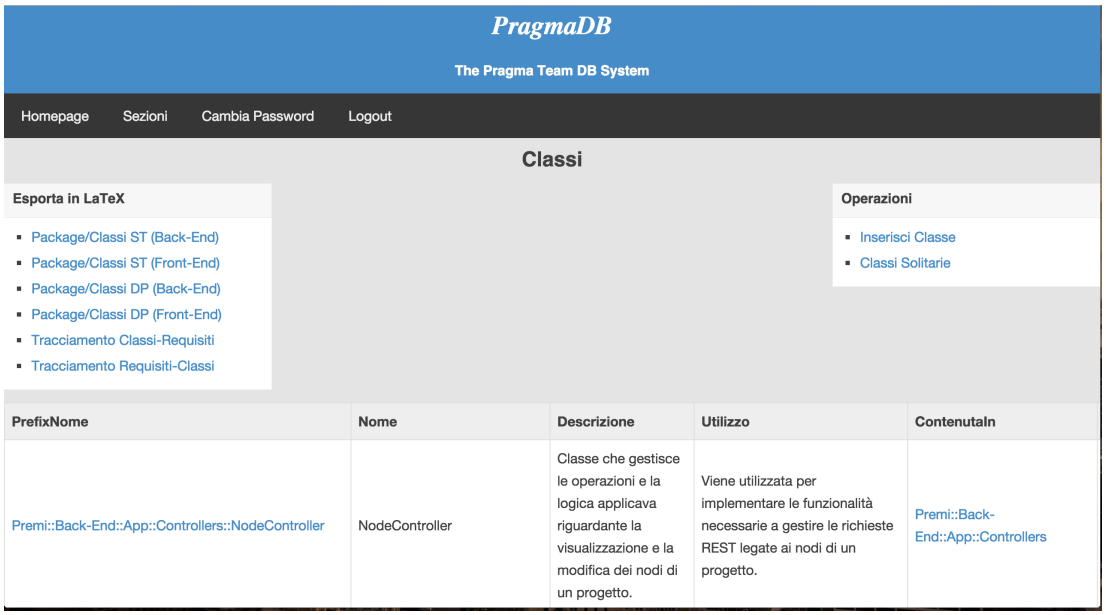


Figura 14: *PragmaDB* - pagina delle classi

Figura 15: *PragmaDB* - pagina dei test