

**Documentazione Progetto Ingegneria Del Software**

**Corso Di Laurea Di Informatica**

Vincenzo Oratore [N86005092]

Santolo Vito [N86005309]

**Versioning**

**Document History**

Di seguito si riportano le modifiche apportate al documento nel corso dell’intero progetto. Ogni record della tabella sottostante indica una modifica e consta dei seguenti attributi: versione del documento [L’ultimo record ne specifica la versione corrente] data della modifica, autore e descrizione della modifica.

|  |  |  |  |
| --- | --- | --- | --- |
| Versione | Data | Autore | Descrizione |
| 0.0 | 10-10-2025 | Vincenzo Oratore  Santolo Vito | Formalizzazione dei requisiti progettuali. |
| 0.1 | 11-10-2025 al  13-10-2025 | Vincenzo Oratore  Santolo Vito | Creazione struttura del documento e stesura dell’introduzione e finalità del software. |
| 0.2 | 14-10-2025 al  15-10-2025 | Vincenzo Oratore  Santolo Vito | Stesura dell’intera seconda sezione della documentazione. Curata la modellazione dei diagrammi e la tabella di Cockburn. Iniziata la progettazione dei primi mockup. |
| 0.3 | 16-10-25 al  18-10-25 | Vincenzo Oratore  Santolo Vito | Progettazione e inserimento dei mockup all’interno della documentazione e stesura del glossario. |

**Indice**

1. **Introduzione**
   1. Finalità del documento
   2. Riferimenti
   3. Glossario
2. **Documento dei requisiti**
   1. Modello dei casi d’uso
      1. Tabella dei casi d’uso
      2. Descrizione dettagliata dei casi d’uso
      3. Cosa non gestisce il software?
      4. Diagramma dei casi d’uso
   2. Descrizioni testuali strutturate
      1. Tabella Cockburn: Segnalazione Issue
   3. Prototipazione attraverso Mock-Up
      1. Scelte progettuali
      2. Flusso di lavoro
   4. Identificazione e caratterizzazione degli attori
      1. Target utenti
      2. Correlazioni
3. **Requisiti** 
   1. Requisiti non funzionali

**Repository di GitHub**

Link: <https://github.com/v-oratore/BugBoard26.git>

1. **Introduzione**
   1. **Finalità del documento**

Questo documento descrive i requisiti per una nuova piattaforma di nome “BugBoard26” di gestione bug destinata a SoftEngUniNa. L'obiettivo primario del sistema è ottimizzare la gestione collaborativa delle issue nei progetti software interni all'azienda. La Specifica dei Requisiti Software [SRS] che segue dettaglia l'insieme completo dei requisiti funzionali e non funzionali. Questo testo costituisce il riferimento ufficiale per le fasi di progettazione e sviluppo e garantisce che tutte le parti interessate abbiano una visione comune delle capacità e del funzionamento della piattaforma.

* 1. **Riferimenti**

Questo documento è stato strutturato seguendo lo standard IEEE Std. 830-1993, che rappresenta il riferimento per la stesura delle Specifiche dei Requisiti Software. L'utilizzo di questo standard garantisce che i requisiti siano chiari, completi e verificabili. In particolare, il modello IEEE 830-1993 favorisce un'organizzazione coerente del documento, una netta distinzione tra requisiti funzionali e non funzionali, e una migliore tracciabilità in tutte le fasi del progetto, facilitando così la comunicazione tra committenti e sviluppatori.

|  |  |
| --- | --- |
| **Termine** | **Spiegazione** |
| Attore | Ruolo ricoperto da un utente o da un sistema esterno che interagisce con il software per raggiungere un obiettivo. |
| Caso d’uso [Use Case] | Descrizione di una sequenza di azioni eseguite da un attore per ottenere un risultato specifico tramite il sistema. Definisce un requisito funzionale. |
| Issue | Segnalazione registrata nel sistema che può indicare un bug, una richiesta di feature o un task da completare. |
| Mockup | Prototipo visivo statico che mostra la struttura e la disposizione degli elementi di un’interfaccia, concentrandosi sul flusso di navigazione e non sugli aspetti grafici finali. |
| Requisito funzionale | Specifica ciò che il sistema deve eseguire, descrivendo una funzionalità, un comportamento o una qualunque interazione tra utente e sistema. |
| Requisito non funzionale | Specifica come il sistema deve operare, definendo attributi di qualità o vincoli come prestazioni, sicurezza o affidabilità. |
| Tabella di Cockburn | Formato testuale strutturato per la descrizione dettagliata di un caso d’uso, comprende attori, precondizioni, post-condizioni e sequenza di passi dello scenario principale o delle eventuali varianti. |
| Uml [Unified Modeling Language] | Linguaggio di modellazione visuale standardizzato usato per progettare e descrivere sistemi software. Il diagramma dei casi d’uso è un esempio di diagramma Uml. |

* 1. **Glossario**

1. **Documento dei requisiti**
   1. **Modello dei casi d’uso**
      1. **Tabella dei casi d’uso**

Il nostro processo di analisi è iniziato con l'identificazione sistematica degli attori che interagiranno con il sistema e delle funzionalità [Casi d'uso] a loro disposizione. L'obiettivo di questo approccio è definire in modo preciso e inequivocabile le interazioni tra utenti e software. Per garantire una rappresentazione visiva chiara e standardizzata, abbiamo utilizzato il linguaggio UML [Unified Modeling Language]. Come primo risultato di questa analisi, la tabella nella pagina seguente formalizza la mappatura tra ciascun attore e i casi d'uso di sua pertinenza. Nelle sezioni successive, invece, presenteremo il diagramma UML completo che illustra graficamente queste stesse relazioni.

|  |  |
| --- | --- |
| **Attore** | **Casi d’uso** |
| Utente non autenticato | * Effettuare l’autenticazione |
| Utente autenticato | * Effettuare il logout * Segnalare una issue * Visualizzare il riepilogo delle issue * Modificare lo stato di una issue assegnata a sé * Filtrare e ordinare il riepilogo delle issue |
| Amministratore | * Eredita tutti i casi d’uso dell’utente autenticato * Accesso e controllo completo su tutte le issue * Creare nuove utenze normali o “amministratore” * Assegnare issue a un utente * Impostare scadenze alle varie issue |

**2.1.2 Descrizione dettagliata dei casi d’uso**

* **Autenticazione**: Deve essere implementato un sistema di autenticazione semplice e sicuro, basato su e-mail e password. Le informazioni gestite dall’applicazione sono critiche per l’azienda, ed è fondamentale preservarne l’integrità e la segretezza. Il sistema viene fornito con un account da amministratore già attivo, con credenziali di default. Un amministratore può creare ulteriori utenze, specificando una e-mail, una password e indicando se quell’utenza sarà “normale” o “amministratore”.
* **Segnalazione issue:** Tutti gli utenti autenticati possono segnalare una issue indicando almeno un titolo e una descrizione. Alcuni utenti potrebbero voler specificare anche una priorità e sarebbe gradita la possibilità di allegare una immagine. Le issue possono essere di diverso tipo: question, bug, documentation e feature. Le issue create sono inizialmente nello stato “todo”.
* **Riepilogo:** Il sistema deve offrire una vista riepilogativa delle issue, con la possibilità di filtrare o ordinare i risultati in base a criteri come tipologia, stato, priorità o altri parametri rilevanti.
* **Modifica stato:** L’utente cui un bug viene assegnato può successivamente modificarne lo stato. Quando un bug viene risolto, il sistema dovrebbe notificare l’utente che lo ha segnalato.
* **Controllo da parte degli amministratori:** Gli utenti devono poter modificare solo i bugs a loro assegnati mentre gli amministratori devono avere accesso e controllo completo.
* **Scadenze da parte degli amministratori:** Gli amministratori possono impostare scadenze opzionali per la risoluzione di una certa issue.

**2.1.3 Cosa non gestisce il software?**

Sono escluse dallo scopo del progetto diverse funzionalità avanzate di gestione e collaborazione. Il sistema non implementerà l'assegnazione diretta dei bug con relative notifiche, né fornirà suggerimenti automatici sugli assegnatari basati sul carico di lavoro. Non saranno presenti funzionalità collaborative come la sezione commenti, la cronologia dettagliata delle modifiche o il supporto per etichette personalizzabili. Inoltre, sono omesse funzioni specifiche di workflow come la marcatura dei duplicati e l'archiviazione dei bug. Sul fronte dell'analisi e dell'esportazione, il software non includerà una dashboard per amministratori con dati aggregati, la generazione di report mensili, una funzione di ricerca per parola chiave, né la possibilità di esportare elenchi in formati come CSV, Excel o PDF. Infine, non è prevista la creazione di account esterni in modalità "read-only".

**2.1.4 Diagramma dei casi d’uso**

Per definire i confini del sistema BugBoard26 e le responsabilità dei suoi attori, presentiamo il seguente diagramma. Questo modello utilizza la notazione UML per mappare in modo visuale e formale le relazioni tra gli attori e i casi d'uso. Questa rappresentazione grafica integra l'analisi tabellare precedente, fornendo una visione d'insieme chiara e standardizzata delle funzionalità del sistema.

**Immagine che contiene diagramma, linea, design

Il contenuto generato dall'IA potrebbe non essere corretto.**

**Figura 1: Diagramma dei Casi d’Uso per il sistema BugBoard26.**

* 1. **Descrizioni testuali strutturate**

In questa sezione, analizziamo in dettaglio il caso d'uso "Segnalare una issue", scelto perché rappresenta una funzionalità essenziale e non banale, in linea con le richieste del progetto. Per documentare formalmente il flusso di interazione tra utente e software, abbiamo adottato lo stile proposto da Alistair Cockburn in "Writing Effective Use Cases". Per ragioni di sintesi, abbiamo utilizzato una versione semplificata del suo template tabellare, concentrandoci solo sugli elementi fondamentali. Questa analisi di interazione ha costituito la base di partenza e il riferimento diretto per la successiva fase di progettazione visuale, durante la quale sono stati realizzati i mock-up dell'interfaccia utente.

**2.2.1 Tabella Cockburn: Segnalazione Issue**

|  |  |  |  |
| --- | --- | --- | --- |
| **USE CASE #01** | **SEGNALAZIONE ISSUE** | | |
| Goal | Segnalazione Issue | | |
| Preconditions | Utente accede, trova la issue | | |
| Success End Condition | Segnalazione | | |
| Primary Actor | Utente | | |
| Descrizione | Step n. | Utente | Sistema |
| 1 | Nel mock-up “Home”, seleziona un progetto tra i disponibili. |  |
| 2 |  | Mostra riepilogo di tutte le issue relative al progetto specifico, con tutte le informazioni relative dei singoli issue. |
| 3 | Nel mock-up “Riepilogo Issue”, clicca sul pulsante “+” in alto a destra. |  |
| 4 |  | Mostra il modale dedicato, richiedendo il titolo, descrizione, tipologia, priorità e media in allegato. |
| 5 | Compila tutti i campi e clicca il pulsante “Save” in alto a destra. |  |
| 6 |  | Torna alla pagina precedente, mostra un messaggio di conferma, aggiorna il riepilogo delle issue e mostra la segnalazione appena avvenuta in cima alla lista. |
|  | | |
| Estensione | Step n. | Utente | Sistema |
| 5.a |  | Il sistema mostra la scritta “Please provide all relevant information before submitting the issue.”. |
| L’utente non inserisce un campo | 5.b | Inserisce i dati mancanti o errati e riprende dal step 5. |  |

**2.3 Prototipazione attraverso Mock-Up**

In questa sezione vengono illustrati i prototipi a bassa fedeltà realizzati per BugBoard26. Tale approccio è coerente con la fase progettuale iniziale [Homework 1], dove la priorità non è la definizione di un'interfaccia utente definitiva. L'obiettivo primario di questi Mock-up è, infatti, validare la struttura logica e il flusso di interazione del caso d'uso "Segnalare una issue". Questi prototipi costituiscono uno strumento di analisi che fornisce una rappresentazione concreta delle funzionalità, ponendo le basi per le successive fasi di progettazione visuale approfondita.

**2.3.1 Scelte progettuali**

Questi prototipi a bassa fedeltà servono a validare l'architettura dell'interazione di BugBoard26, che si fonda su due pilastri concettuali, pensati per un'utenza specializzata (sviluppatori, project manager). Il primo pilastro è la **massimizzazione dell'efficienza operativa**. Abbiamo decostruito il processo di segnalazione per ridurre al minimo lo sforzo mentale e il numero di clic. L'interfaccia è priva di qualsiasi elemento ridondante che possa distogliere dal compito. Questo è fondamentale per un utente che percepisce l'atto della segnalazione come un'attività secondaria e urgente. Il secondo è la **composizione visiva ordinata**. Il design è pulito e strutturato. L'uso deliberato di ampi spazi vuoti serve a separare logicamente i blocchi di contenuto, migliorando la leggibilità. Le azioni eseguibili sono differenziate visivamente in base alla loro importanza, creando un percorso di navigazione intuitivo.

**2.3.2 Flusso di lavoro**

Di seguito `e illustrato il flusso visuale che corrisponde agli step definiti nella tabella di Cockburn.

1. Il percorso dell'utente comincia dalla **Home**, dove gli viene richiesto di selezionare un progetto dalla lista di quelli disponibili.

Figura 2: Mockup della **Home**

**Immagine che contiene testo, schermata, design

Il contenuto generato dall'IA potrebbe non essere corretto.**

1. Una volta scelto il progetto, il sistema mostra il riepilogo contenente tutte le issue relative a quel progetto.
2. Nel riepilogo contenente tutte le issue, l’utente deve cliccare sul pulsante “+” in alto a destra per poter procedere alla segnalazione di una nuova issue.

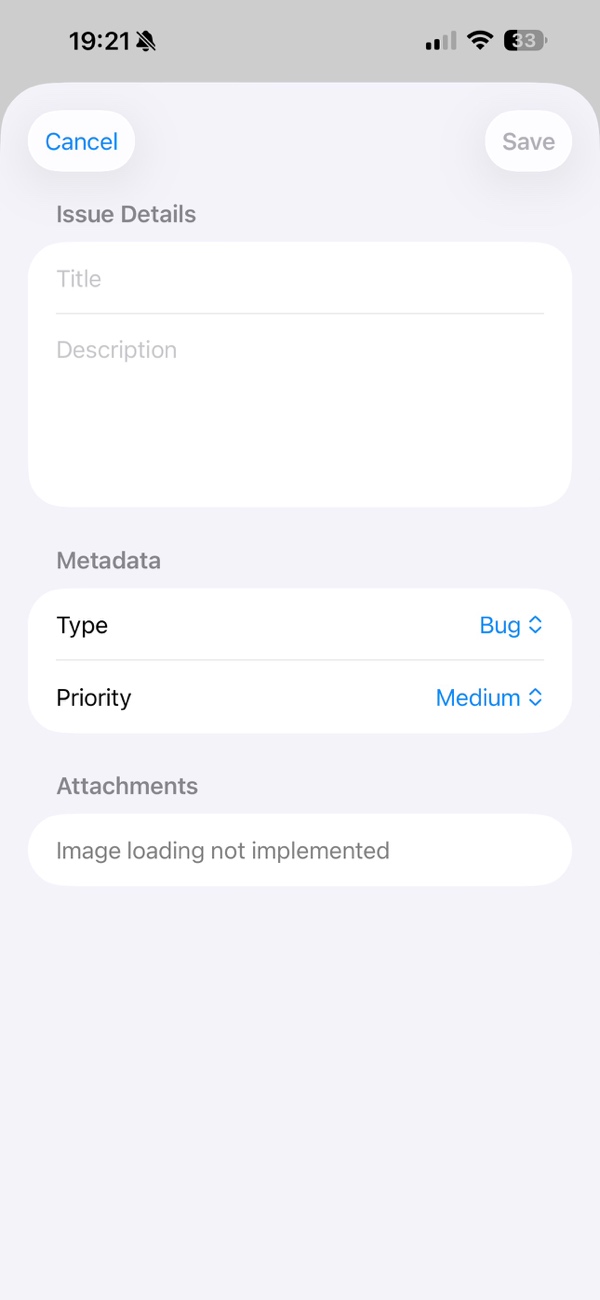
Figura 3: Mockup di “Project’s Issue”

**Immagine che contiene testo, schermata, Carattere, design

Il contenuto generato dall'IA potrebbe non essere corretto.**

1. Il sistema mostra il modale dedicato alla segnalazione di una nuova issue richiedendo tutte le informazioni relative come il titolo, descrizione, stato, tipologia, priorità ed eventualmente un media in allegato.
2. L’utente compila tutti i campi e clicca sul pulsante “Save” in alto a destra.
3. Il sistema re-directa in automatico al “Riepilogo Issue” [**Figura 3**], mostrando un messaggio di conferma e aggiornando il riepilogo delle issue mostrando la nuova segnalazione in cima alla lista.

Figura 4: Mockup di “New Issue”

****

5a. In caso di mancate o errate informazioni durante la compilazione dei dati per la segnalazione di una nuova issue, il sistema mostra il seguente messaggio di errore “Please provide all relevant information before submitting the issue.” sotto ai due campi del titolo e della descrizione.

5b. L’utente aggiunge o corregge i dati e riprende dallo step 5.

**2.4 Identificazione e caratterizzazione degli attori**

Per definire correttamente i confini e le funzionalità di BugBoard26, abbiamo identificato tre distinti ruoli utente, ognuno con specifici permessi e obiettivi:

* **Utente Non Autenticato**: Questo attore rappresenta chiunque tenti di accedere al sistema senza aver effettuato il login. Il suo unico scopo è autenticarsi, e di conseguenza, l'unica azione che può compiere è il login.
* **Utente Autenticato**: È il profilo standard per i membri del gruppo, come sviluppatori o tester. Questo utente è il cuore dell'operatività quotidiana: può segnalare nuove issue, consultare gli elenchi e modificare lo stato dei task a lui assegnati. Non ha però privilegi di gestione del sistema.
* **Amministratore**: Si tratta di un ruolo gestionale con privilegi elevati. L'Amministratore eredita tutte le capacità dell'Utente Autenticato, ma ha in aggiunta accesso a funzioni esclusive, come la creazione di nuovi account [Sia amministratore che utente normale] e l'impostazione delle scadenze per le issue.

**2.4.1 Target utenti**

Per progettare un software efficace, non è sufficiente elencare le funzionalità; è indispensabile comprendere per chi lo stiamo costruendo. Invece di progettare per un "utente" generico e astratto, abbiamo distillato le esigenze del nostro target in archetipi concreti, noti come **Personas**.

**Obiettivo**: Le Personas fungono da "bussola" per il design, costringendo il gruppo a prendere decisioni ponendosi domande specifiche: "Chiara capirebbe questo pulsante? Questo flusso è abbastanza veloce?". Questo approccio, centrato sull'utente, previene la creazione di funzionalità superflue e garantisce che il prodotto finale risolva problemi specifici e tangibili per le persone che lo useranno. L'obiettivo non è soddisfare tutti, ma soddisfare i nostri utenti chiave.

**Metodologia di Definizione**: Non potendo basarci su ricerche di mercato su larga scala, abbiamo adottato un approccio qualitativo interno. Abbiamo condotto brevi interviste informali con alcuni membri del gruppo [sia sviluppatori che responsabili di progetto] per comprendere come gestiscono attualmente le problematiche e quali sono i loro punti dolenti. Da questa analisi sono emersi due profili principali che incarnano la tensione tra "esecuzione" e "supervisione":

* **Analisi dei Punti Dolenti [Pain Points]:** Abbiamo mappato le frustrazioni ricorrenti.
* **Raggruppamento Comportamentale**: Abbiamo identificato pattern nel modo in cui i diversi ruoli cercano informazioni e gestiscono il loro tempo.
* **Sintesi in Archetipi**: Questi pattern sono stati sintetizzati nei due profili utente che seguono.

**Persona 1: Marco, Developer Mid-Level**

**Profilo**: Marco 28 anni, developer focalizzato. Il suo valore primario è la produttività e la capacità di rimanere concentrato sul codice.

**Obiettivi**: Documentare un problema tecnico in modo preciso [allegando log o screenshot] ma estremamente rapido. Vuole vedere solo i task che gli sono stati assegnati e filtrare tutto il resto.

**Frustrazioni**: Applicazioni lente. Flussi di lavoro che richiedono troppi clic per un'azione semplice. Non poter allegare rapidamente uno screenshot.

**Citazione**: "Se ci metto più tempo a segnalare il bug che a correggerlo, stiamo usando lo strumento sbagliato."

**Persona 2: Chiara, Team Lead**

**Profilo**: Chiara, 41 anni, gestisce un gruppo di 6 sviluppatori. Responsabile della qualità, del carico di lavoro e del rispetto delle scadenze.

**Obiettivi**: Avere una "mappa" dello stato dei lavori. Capire al volo chi è sovraccarico e quali sono i colli di bottiglia.

**Frustrazioni**: Dover chiedere aggiornamenti su Slack o via e-mail. Non avere una visione d'insieme aggregata.

**Citazione**: "Non posso gestire il gruppo se non so chi sta lavorando su cosa. Ho bisogno di una dashboard, non di un elenco."

**2.4.2 Correlazione tra Attori e Personas**

È fondamentale capire la differenza tra Attori e Personas:

* Gli **Attori** descrivono i permessi [Cosa un utente può fare].
* Le **Personas** descrivono il comportamento [Cosa un utente vuole fare e perché].

La progettazione di BugBoard26 nasce dalla loro intersezione:

* **Marco** (Developer) incarna l'Utente Autenticato. Le sue esigenze di velocità e affidabilità sono la causa diretta dei requisiti di performance e della progettazione di un form di segnalazione snello e rapido.
* **Chiara** (Team Lead) rappresenta l'Amministratore. Il suo bisogno di una "mappa" e di gestire il carico di lavoro giustifica la creazione di funzionalità di gestione e la necessità di una dashboard di supervisione.

**2.5 Requisiti Non-Funzionali**

I requisiti non funzionali stabiliscono gli standard di qualità e i vincoli operativi per il sistema BugBoard26. Questi requisiti sono fondamentali per assicurare che il software non si limiti a funzionare, ma sia anche efficiente, sicuro e pratico per l'ambiente di lavoro per cui è stato progettato.

* **Usabilità:** Il sistema è progettato per utenti tecnici che privilegiano l'efficienza. Di conseguenza, l'usabilità è misurata come segue: un nuovo utente, al suo primo tentativo e senza alcuna formazione specifica, deve poter completare con successo il caso d'uso "Segnalare una issue" in meno di 60 secondi.
* **Affidabilità:** Essendo uno strumento di lavoro primario, il sistema deve garantire un'elevata disponibilità. Si esige un **uptime** mensile minimo del 99.5%. Da questo calcolo sono escluse le finestre di manutenzione programmata, le quali devono essere notificate al gruppo con un preavviso di almeno 24 ore.
* **Performance:** Per non ostacolare la produttività degli sviluppatori, il sistema deve rispondere in modo quasi istantaneo. Nello specifico:
  + Il caricamento della dashboard principale deve completarsi in meno di 1.5 secondi.
  + L'invio del modulo per una nuova issue deve restituire un riscontro all'utente in meno di 1 secondo.
* **Sicurezza:** La gestione delle credenziali di accesso rende la sicurezza un aspetto prioritario e non negoziabile:
  + Tutte le comunicazioni tra il client e il server devono essere obbligatoriamente criptate tramite protocollo **HTTPS**.
  + Le password degli utenti non devono mai essere archiviate in chiaro. È obbligatorio l'uso di un **algoritmo di hashing robusto** per la loro memorizzazione nel database.
* **Manutenibilità:** Per assicurare la scalabilità e l'evoluzione futura del progetto, il codice sorgente deve rispettare rigorosamente gli standard di codifica definiti per la tecnologia in uso.