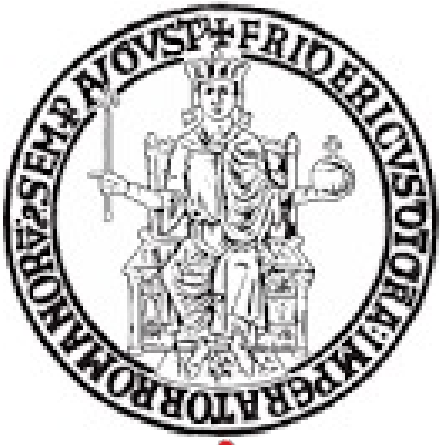

Progetto di Ingegneria del Software

A.A. 2025/2026

Software Requirements Specification

per il sistema *BugBoard26*

(Soggetto a successive modifiche)

Autori:

Salvatore Savino (N86004099)
Francesco Sorrentino (N86004817)

Indice

1 Introduzione

Il progetto **BugBoard26** riguarda la specifica e la progettazione di un sistema di *issue tracking*, concepito per supportare i team di sviluppo software nella gestione del ciclo di vita delle anomalie (bug). L'obiettivo primario del sistema è ottimizzare e semplificare il processo di segnalazione, tracciamento e risoluzione delle issue, favorendo la collaborazione e l'efficienza operativa.

1.1 Struttura della Documentazione

Il presente documento costituisce la prima consegna progettuale (Homework 1, successivamente verrà unificato con gli altri Homework) e si concentra sulla stesura del seguente elaborato:

- **Documento dei Requisiti del Software:** in questa sezione fondamentale, vengono analizzate in dettaglio le specifiche della traccia progettuale. Si procede con l'elaborazione rigorosa di tutti i requisiti funzionali, non-funzionali e dei vincoli che definiscono il sistema da realizzare. Questa fase analitica rappresenta le fondamenta su cui si baserà l'intero ciclo di sviluppo del software.

2 Documento dei Requisiti

Al fine di garantire il conseguimento degli obiettivi progettuali per il sistema **BugBoard26**, risulta inevitabile una definizione coerente e completa dei suoi requisiti. Il processo di analisi condotto si è articolato sui seguenti punti cardine:

Chi lo userà? Per prima cosa, sono stati definiti i ruoli(gli **Attori**) e i permessi di ciascuno, per avere chiaro fin da subito chi può fare cosa.

Cosa deve fare essattamente? Sono state individuate le funzionalità chiave(i **Casi d'Uso**), come la segnalazione di una issue o la creazione di un nuovo utente da parte di un amministratore.

Per chi lo stiamo progettando davvero? Per non creare uno strumento astratto, sono stati immaginati dei profili utente realistici (le **Personas**), in modo da avere sempre in mente le esigenze di persone vere durante le scelte di progettazione.

Come apparirà (più o meno)? Sono state realizzate (molto approssimativamente) delle bozze dell'interfaccia (i **Mock-Up**) per assicurare fin da subito che il flusso di navigazione sia logico e intuitivo, prima ancora di scrivere una riga di codice.

2.1 Modello dei Casi d'uso

2.1.1 Tabella dei casi d'uso

Il modello dei casi d'uso ha lo scopo primario di definire in modo esigente e non ambiguo le interazioni tra gli attori e le funzionalità esposte dal sistema. L'utilizzo di UML (*Unified Modeling Language*) garantisce una rappresentazione visuale chiara e completamente comprensibile di tali interazioni.

Il processo di analisi ha avuto inizio con l'identificazione sistematica degli attori che operano con il sistema e dei casi d'uso a loro associati. Tale mappatura iniziale è stata formalizzata in una tabella che correla ciascun attore con le funzionalità di sua pertinenza, come illustrato nella pagina che segue. Nelle sezioni successive, invece, viene presentato il diagramma dei casi d'uso completo che visualizza queste relazioni.

Tabella 1: Tabella degli Attori e dei rispettivi Casi d'Uso per BugBoard26.

Attore	**Casi d'uso**
Utente Non Autenticato	• Autenticarsi nel sistema
Utente Autenticato	• Segnalare una issue • Consultare elenco issue • Modificare stato issue • Cercare issue per parola chiave • Filtrare e ordinare l'elenco
Amministratore	• Creare nuova utenza • Impostare scadenza issue • *(Eredita tutti i casi d'uso dell'Utente Autenticato)*

2.1.2 Descrizione dettagliata dei casi d'uso:

Le seguenti descrizioni specificano i requisiti funzionali del sistema, precedentemente identificati. Ciascun punto definisce il comportamento che il sistema deve esibire in risposta a una specifica interazione con un attore.

Funzionalità per l'Utente Non Autenticato:

Autenticarsi nel sistema: Il sistema deve fornire a un `Utente Non Autenticato` la possibilità di accedere tramite credenziali (email e password). In caso di successo, il sistema deve validare le credenziali, concedere l'accesso e assegnare all'attore il ruolo di `Utente Autenticato`. In caso di fallimento, deve essere notificato un errore all'utente.

Segnalare una issue: Il sistema deve permettere a un `Utente Autenticato` di creare una nuova segnalazione (*issue*). La creazione deve richiedere obbligatoriamente un titolo e una descrizione del problema. Deve essere data all'utente la possibilità di associare opzionalmente una priorità (es. Bassa, Media, Alta). A seguito della sottomissione, il sistema deve validare i dati, registrare la nuova issue con un identificativo univoco e impostarne lo stato iniziale su "Aperta".

Consultare elenco issue: Il sistema deve presentare all'`Utente Autenticato` l'elenco completo delle issue registrate. L'elenco deve essere visualizzato in formato tabellare

e deve includere, per ciascuna issue, le informazioni salienti: ID, titolo, stato corrente, priorità e data di creazione. Ciascun elemento della lista deve essere selezionabile per accedere alla vista di dettaglio.

Modificare stato issue: Un `Utente Autenticato` deve avere la facoltà di modificare lo stato di una issue esistente. Il sistema deve rendere disponibili solo le transizioni di stato valide in base allo stato corrente della issue (es. da "Aperta" a "In Lavorazione", da "In Lavorazione" a "In Verifica"). Ogni cambiamento di stato deve essere registrato e tracciato nello storico della segnalazione.

Filtrare e cercare issue: Il sistema deve fornire all'`Utente Autenticato` strumenti per la ricerca e il filtraggio dell'elenco delle issue. Deve essere presente una funzionalità di ricerca testuale per parola chiave su titolo e descrizione. Inoltre, il sistema deve offrire filtri per restringere i risultati in base a criteri specifici, quali stato o priorità.

Funzionalità ad uso esclusivo dell'Amministratore:

Creare nuova utenza: La creazione di nuove utenze deve essere una funzionalità ad uso esclusivo dell'`Amministratore`. Il sistema deve fornire un'interfaccia dedicata attraverso la quale l'amministratore deve poter registrare un nuovo utente nel sistema, specificandone le informazioni anagrafiche (nome, cognome) e le credenziali di accesso iniziali (email e password temporanea).

Impostare scadenza issue: L'`Amministratore` deve avere la facoltà esclusiva di associare una data di scadenza a una specifica issue. Tale informazione deve essere visibile nel dettaglio della issue e deve poter essere utilizzata come parametro per ordinare o filtrare l'elenco, al fine di prioritizzare le attività del team di sviluppo.

2.1.3 Diagramma dei Casi d'Uso

A complemento della descrizione tabellare, il diagramma che segue fornisce una rappresentazione visuale e formale, secondo la notazione UML, delle interazioni tra gli attori e i casi d'uso del sistema *BugBoard26*. Questo modello è fondamentale per definire i confini del sistema e le responsabilità di ciascun attore.

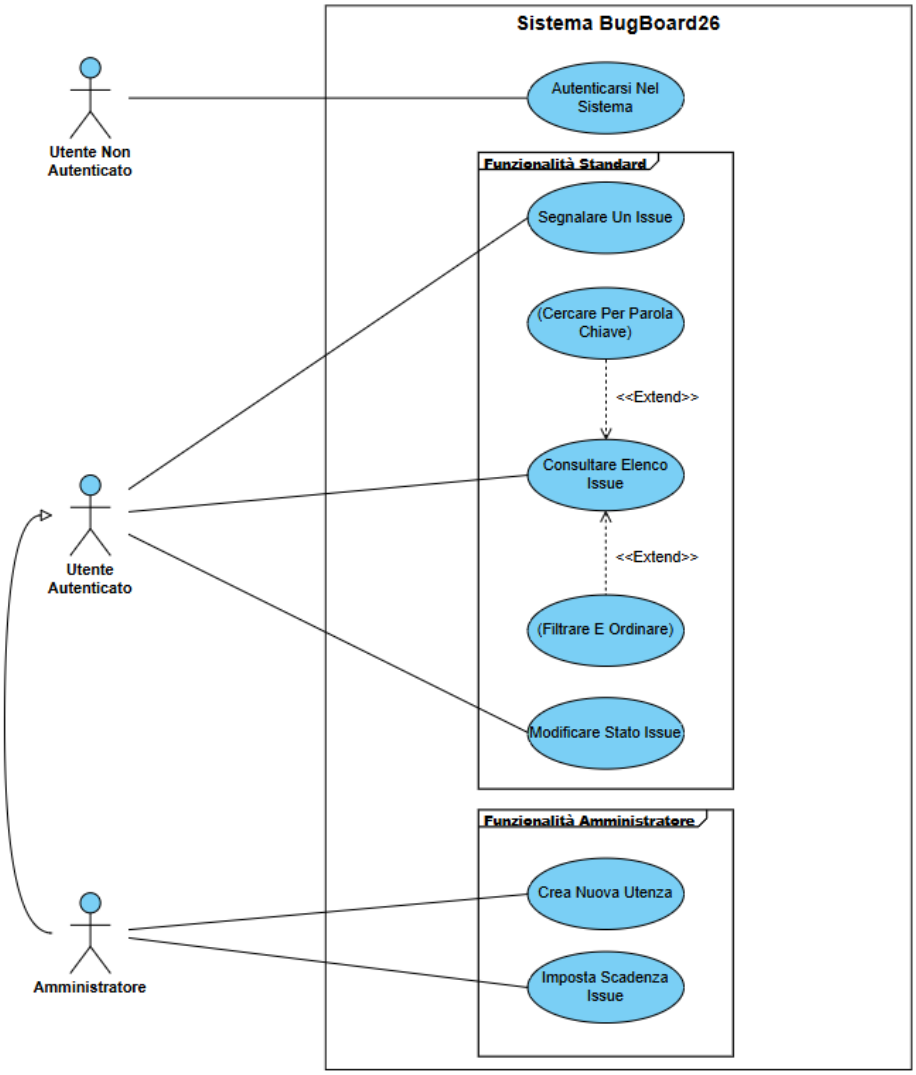

Figura 1: Diagramma dei Casi d'Uso per il sistema BugBoard26.

2.2 Descrizioni Testuali Strutturate

In questa sezione viene condotta un'analisi dettagliata di un caso d'uso rappresentativo del sistema, esprimendo in modo formale il flusso di interazioni tra l'utente e il software.

Lo stile di documentazione adottato è quello proposto da Alistair Cockburn nel suo testo *"Writing Effective Use Cases"*. Per ragioni di sintesi e pertinenza, è stata utilizzata una versione semplificata del template tabellare originale, focalizzandosi sugli elementi essenziali del caso d'uso.

In conformità alle richieste della traccia progettuale, che prevedeva l'analisi di un caso d'uso non banale, è stato selezionato **"Segnalare una issue"** quale funzionalità essenziale del sistema. Per definire il modello di interazione utente-sistema, la composizione di questa tabella ha rappresentato la base su cui, in una fase successiva, è stata costruita la progettazione visuale dell'interfaccia utente attraverso i mock-up.

2.2.1 Tabella Cockburn: Segnalare una issue

USE CASE #01: SEGNALARE UNA ISSUE			
Goal	L'utente vuole segnalare una issue		
Preconditions	L'utente ha effettuato l'accesso		
Success End Condition	L'utente ha segnalato la issue ed è consultabile da tutti gli altri utenti		
DESCRIPTION	**Step n.**	**Utente**	**Sistema**
	1	Preme il pulsante "Segnala una issue" nel mockup "Home".	
	2		Mostra il mockup "Segnalazione" che richiede di inserire: Oggetto, Descrizione e eventuali allegati.
	3	Inserisce tutti i dati e clicca il pulsante "Conferma segnalazione".	
	4		Torna alla home, mostra un messaggio di conferma e aggiorna la lista delle issue, mostrando la nuova segnalazione in cima.
EXTENSIONS	**Step n.**	**Utente**	**Sistema**
L'utente non inserisce oggetto o descrizione	3.a		Il Sistema mostra la scritta "Inserisci tutte le informazioni utili per segnalare la issue".
	3.b	Inserisce i dati mancanti e riprende dal step 3.	

2.3 Prototipazione visuale attraverso Mock-up

Questa sezione presenta i mock-up realizzati per il sistema *BugBoard26*. Coerentemente con la natura preliminare di questa prima fase progettuale (Homework 1), si è optato per la realizzazione di prototipi a **bassa fedeltà**.

Tale scelta è intenzionale: l'obiettivo attuale non è sviluppare un'estetica o un'interfaccia utente definitiva, ma piuttosto verificare la struttura e il processo di interazione del caso *"Segnalare una issue"*. Pertanto, questi prototipi fungono quindi da strumento di analisi per fornire una rappresentazione concreta e tangibile delle funzionalità menzionate prima di intraprendere scelte di design più approfondite.

2.3.1 Scelte Progettuali

Nonostante la scarsa affidabilità dei prototipi, la filosofia di design adottata per *BugBoard26* si basa già su principi di minimalismo, efficienza e chiarezza, progettato per soddisfare le esigenze di utenti tecnici (sviluppatori, project manager) che apprezzano la rapidità operativa e l'assenza di complicazioni inutili.

Efficienza del Flusso e Minimalismo: L'interfaccia è stata progettata in modo da ridurre il numero di passaggi e il carico cognitivo necessario per completare le operazioni essenziali. Ogni componente non necessario è stato eliminato per non distrarre l'utente dal suo compito principale. Tale metodo soddisfa le esigenze della Persona *"Luca, lo Sviluppatore Junior"*, che vede la segnalazione di un bug come una perdita di concentrazione e desidera tornare all'attività di sviluppo il più rapidamente possibile.

Chiarezza Visiva e Gerarchia: È stato adottato un layout pulito, con un uso strategico dello spazio bianco per migliorare la leggibilità e ridurre l'affaticamento visivo. Il contrasto cromatico tra lo sfondo neutro e gli elementi interattivi primari (es. i pulsanti) stabilisce una chiara gerarchia visiva, guidando l'attenzione dell'utente verso le azioni più importanti.

Feedback Immediato e Contestuale: In accordo con le euristiche di usabilità di Nielsen, ogni azione intrapresa dall'utente riceve un feedback sistematico e immediato. I messaggi di errore vengono presentati in modo contestuale, indicando con precisione la natura del problema (es. campi obbligatori non compilati), mentre i messaggi di successo confermano l'esito positivo delle operazioni senza interrompere il flusso di lavoro.

2.3.2 Flusso di Lavoro: Segnalare una Issue

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

Step 1: Avvio della Segnalazione dalla Home Page Il punto di partenza per l'utente è la Home Page del sistema. Questa schermata fornisce una panoramica immediata delle issue correnti. L'azione principale, **"Segnala Una Issue"**, è rappresentata da un pulsante ben visibile nella barra di navigazione superiore, garantendo che sia sempre accessibile senza dover navigare attraverso menù complessi.

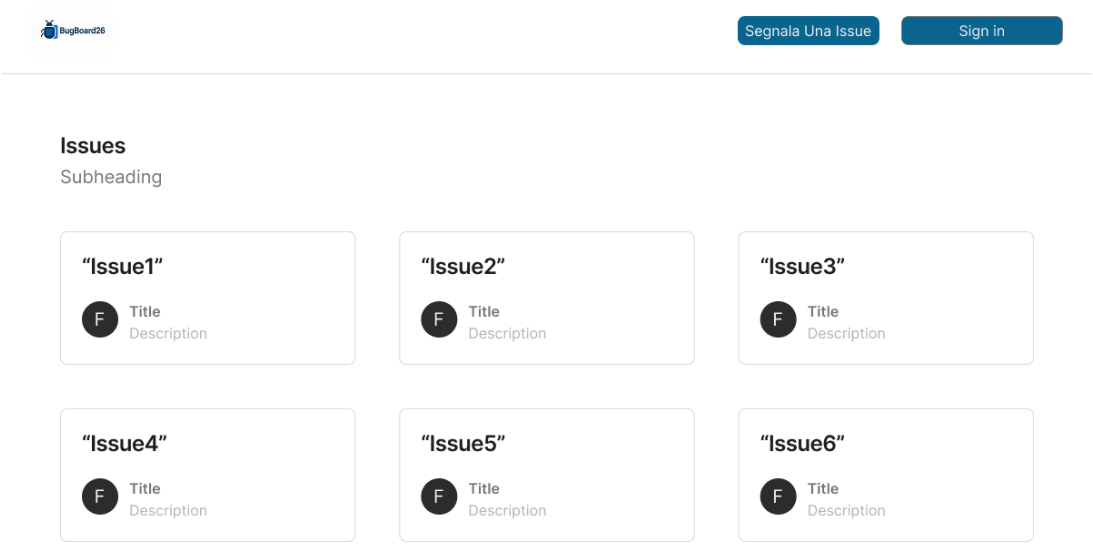

Figura 2: Mock-up della Home Page (Step 1).

Step 2: Presentazione del Form di Inserimento Al click sul pulsante di segnalazione, l'utente viene indirizzato a una pagina dedicata. Qui, il sistema presenta un form di inserimento dati progettato per essere il più semplice possibile. I campi richiesti sono limitati all'essenziale ("Oggetto", "Descrizione") per velocizzare la compilazione. È inclusa anche un'area per l'upload di file, funzionalità utile per allegare screenshot o log di errore.

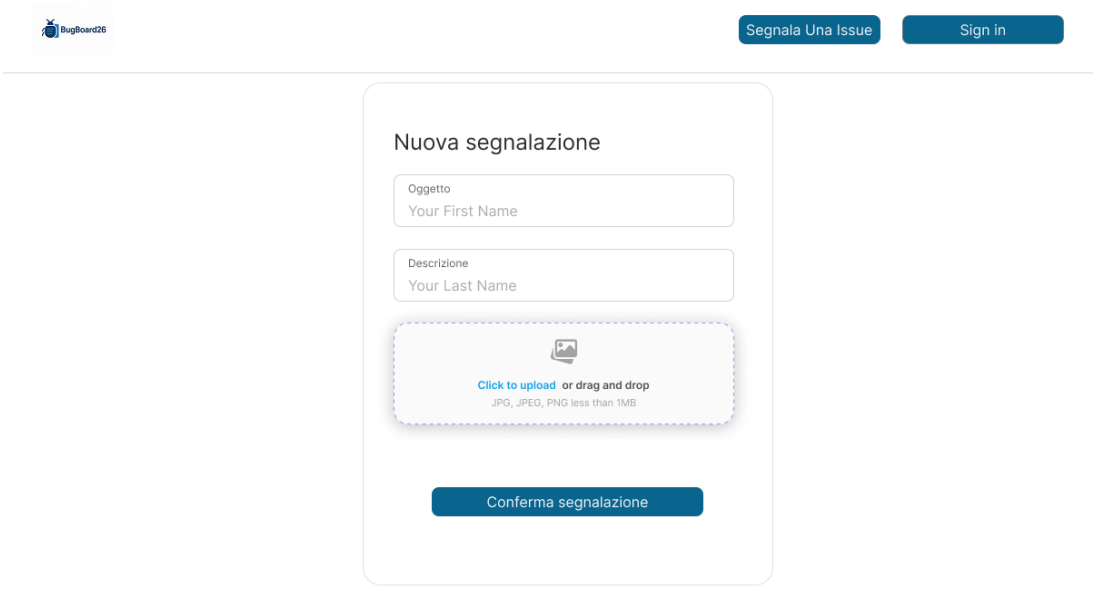

Figura 3: Mock-up del form di segnalazione vuoto, presentato all'utente (Step 2).

Estensione 3.a: Gestione dell'Errore in Compilazione Nel caso in cui l'utente tenti di inviare il form senza aver compilato i campi obbligatori, il sistema previene l'invio e fornisce un feedback di errore chiaro e contestuale. Invece di un pop-up generico, un messaggio di aiuto compare direttamente all'interno dei campi vuoti. Questo approccio, noto come "inline validation", aiuta l'utente a identificare e correggere rapidamente il problema senza interrompere il flusso.

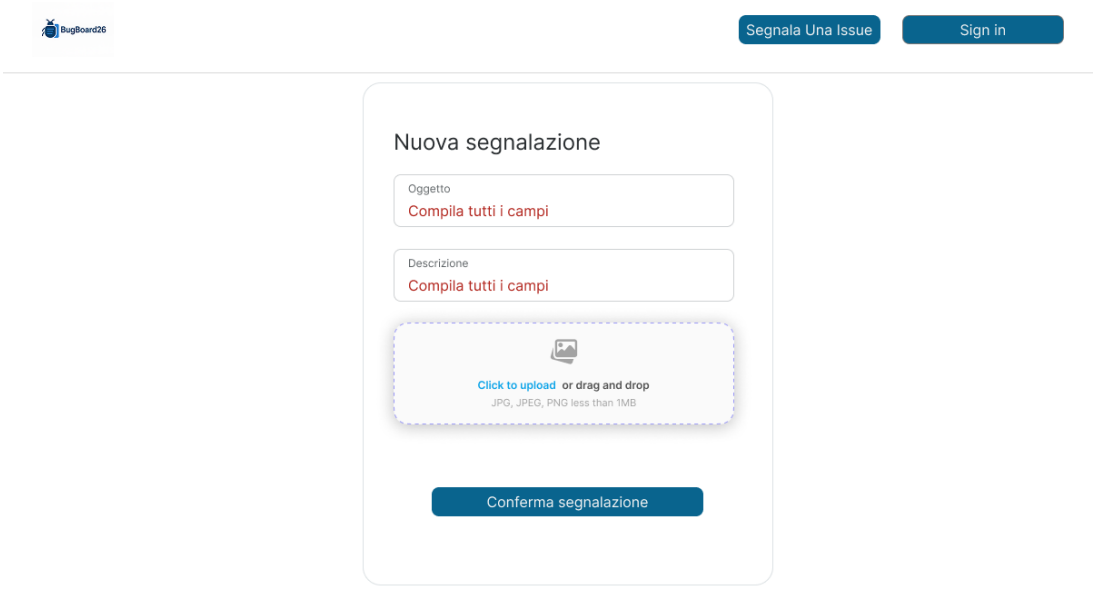

Figura 4: Mock-up del form che mostra un messaggio di errore per campi non compilati (Estensione 3.a).

Step 3 e 3.b: Compilazione Corretta e Invio L'utente compila i campi richiesti. L'interfaccia mostra i dati inseriti, pronti per l'invio. Questa schermata rappresenta lo stato del form poco prima che l'utente prema "Conferma segnalazione", completando con successo l'inserimento dei dati come previsto dallo Step 3 dello scenario principale e dallo Step 3.b dell'estensione.

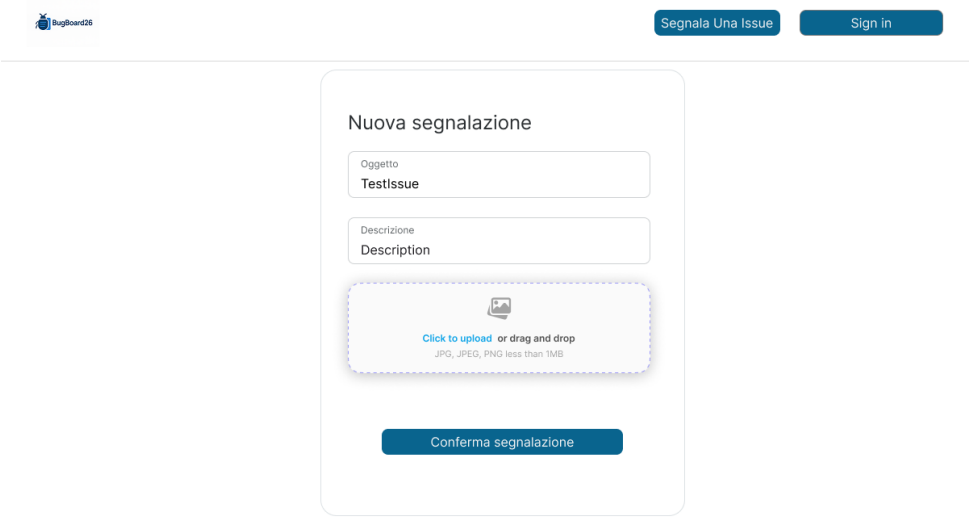

Figura 5: Mock-up del form compilato correttamente dall'utente, pronto per l'invio (Step 3).

Step 4: Conferma di Successo e aggiornamento lista issue Dopo aver premuto "Conferma segnalazione", l'utente viene immediatamente reindirizzato alla Home Page. Il sistema fornisce un feedback di successo visibile ma non invasivo, tramite un banner che appare brevemente in cima alla pagina. Contemporaneamente, la lista delle issue viene aggiornata e la nuova segnalazione ("TestIssue") appare in cima, confermando visivamente all'utente che la sua azione ha avuto l'esito desiderato.

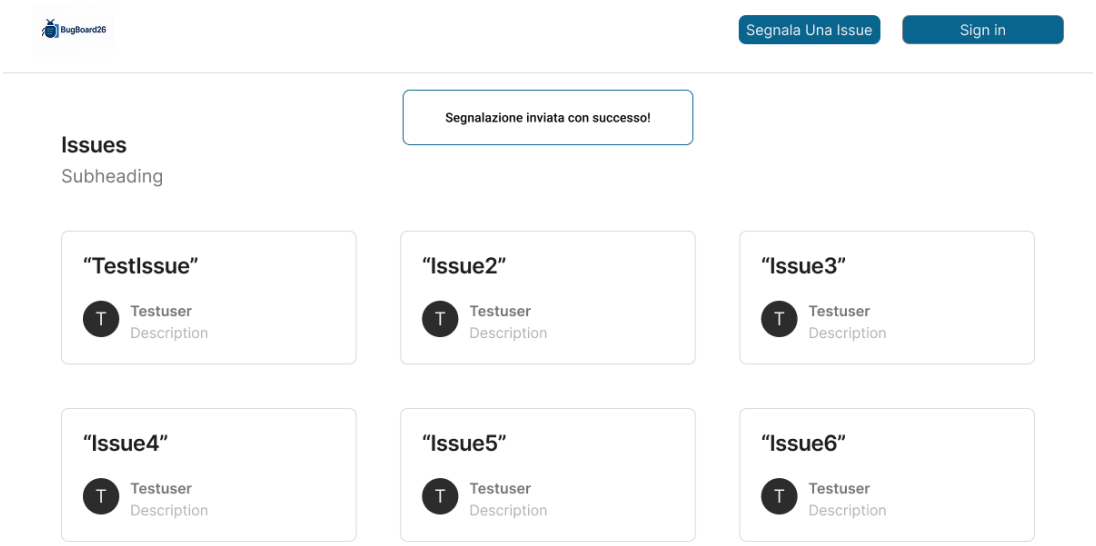

Figura 6: Mock-up della Home Page aggiornata con il banner di successo e la nuova issue (Step 4).

2.4 Identificazione e Caratterizzazione degli Attori

Un passo fondamentale nell'analisi dei requisiti è stata l'identificazione e la caratterizzazione di tutti gli attori che interagiranno con il sistema *BugBoard26*. Questo processo è essenziale per definire il perimetro del sistema e per progettare funzionalità che rispondano a esigenze reali.

L'analisi ha portato all'individuazione di tre attori distinti, i cui ruoli, obiettivi e permessi sono caratterizzati come segue.

Utente Non Autenticato Rappresenta qualsiasi individuo che non ha ancora effettuato l'accesso al sistema. Il suo unico obiettivo è ottenere l'accesso e, di conseguenza, l'unica interazione che può avviare è il caso d'uso *Autenticarsi nel sistema*.

Utente Autenticato È l'attore principale del sistema e rappresenta un membro generico del team (es. sviluppatore, tester). L'obiettivo primario di questo attore è l'operatività quotidiana sul tracciamento dei bug. Può eseguire le funzionalità chiave del sistema, come *Segnalare una issue*, *Consultare l'elenco issue* e *Modificare lo stato issue*, ma non dispone di privilegi amministrativi.

Amministratore È un utente con privilegi elevati, tipicamente con un ruolo gestionale. L'Amministratore **eredita tutte le funzionalità e i permessi dell'Utente Autenticato** e, in aggiunta, ha accesso a funzionalità esclusive per la gestione del sistema, come *Creare nuova utenza* e *Impostare scadenza issue*.

2.4.1 Target Utenti: le Personas

Nello sviluppo del software *BugBoard26*, è essenziale avere un'idea concreta del profilo utente a cui ci si rivolge. Questo approccio aiuta il team a concentrarsi sulle esigenze di categorie ben definite di utenti, sintetizzandone obiettivi, ruoli e necessità in archetipi chiamati **Personas**. Ogni Persona riassume comportamenti, scopi e caratteristiche rilevanti di utenti reali che utilizzeranno il software.

Obiettivo Nello specifico, le Personas permettono ai progettisti di concentrarsi sui bisogni reali dei futuri utilizzatori, evitando generalizzazioni inefficaci. Permettono di focalizzare l'attenzione su un gruppo ristretto di persone con caratteristiche specifiche, come sviluppatori e project manager, evitando di creare un prodotto che, nel tentativo di soddisfare un pubblico troppo vasto, finirebbe per non soddisfare nessuno.

Metodologia di Creazione Per la creazione delle Personas di *BugBoard26*, non potendo condurre ricerche, osservazioni e studi sperimentali diretti, si è adottato un approccio analitico basato sul dominio di riferimento. Il processo si è articolato come segue:

1. **Analisi di Sistemi Esistenti:** Sono state analizzate le funzionalità e i flussi di lavoro di piattaforme di bug tracking e project management di successo (come Jira, Trello e GitHub Issues) per identificare i pattern di interazione più comuni.

2. **Identificazione delle Variabili Comportamentali:** Dall'analisi sono state estratte le variabili chiave per l'utenza target, come la frequenza di segnalazione, la necessità di avere una visione d'insieme, le competenze tecniche e le frustrazioni ricorrenti (es. interfacce lente o complesse).

Di seguito riportiamo le Personas che sono emerse da questo processo di analisi e inferenza.

Luca, lo Sviluppatore Junior

Profilo

24 anni — *Sviluppatore da 1 anno*

Passa la maggior parte della sua giornata a scrivere codice. Per lui, ogni attività che non sia programmazione è un'interruzione che spezza la sua concentrazione.

Obiettivi

Segnalare un bug nel minor tempo possibile

Tornare a programmare rapidamente

Tracciare lo stato delle sue issue senza difficoltà

Frustrazioni

Odia i sistemi macchinosi che richiedono la compilazione di troppi campi per una semplice segnalazione.

Citazione

"Voglio solo segnalare il bug in 30 secondi e tornare al mio codice, non compilare un modulo delle tasse."

Giulia, la Project Manager

Profilo **35 anni** — *Responsabile del team*

Ha bisogno di una visione d'insieme chiara e costantemente aggiornata per pianificare i task e rispettare le scadenze.

Obiettivi

Avere una dashboard chiara dello stato del progetto

Gestire gli accessi del team in autonomia

Prioritizzare il lavoro impostando scadenze

Frustrazioni

Perde tempo se deve chiedere aggiornamenti a voce perché il tool non fornisce una visione aggregata delle informazioni.

Citazione

"Ho bisogno di sapere a colpo d'occhio dove sono i problemi. Se devo aprire 20 ticket per capirlo, il tool è inutile."

2.4.2 Correlazione tra Attori e Personas

Le Personas create non sono entità astratte, ma rappresentano la concretizzazione degli attori precedentemente identificati:

- **Luca, lo Sviluppatore Junior**, rappresenta l'attore **Utente Autenticato**. Le sue necessità di rapidità ed efficienza giustificano le scelte di design minimalista per l'interfaccia di segnalazione.
- **Giulia, la Project Manager**, rappresenta l'attore **Amministratore**. I suoi bisogni di supervisione e gestione giustificano la presenza di funzionalità dedicate come la creazione di utenze e l'impostazione delle scadenze.

Questa duplice analisi assicura che il sistema *BugBoard26* sia progettato non solo per eseguire delle funzioni (come descritto dagli attori), ma per risolvere i problemi reali di persone reali (come descritto dalle Personas).

2.5 Requisiti Non-Funzionali

I requisiti non funzionali definiscono gli attributi di qualità e i vincoli operativi a cui il sistema *BugBoard26* deve attenersi. Essi sono cruciali per garantire che il software sia non solo funzionante, ma anche efficiente, sicuro e realmente utilizzabile nel contesto lavorativo a cui è destinato.

Usabilità: Il sistema è rivolto a un'utenza tecnica (la Persona "Luca") che predilige l'efficienza. Pertanto, il requisito di usabilità è definito come segue: un nuovo utente, senza alcuna formazione specifica, deve essere in grado di completare con successo il caso d'uso "Segnalare una issue" in **meno di 60 secondi** al suo primo tentativo.

Affidabilità: Il sistema deve essere uno strumento di lavoro costantemente disponibile. Si richiede un **uptime minimo del 99.5% su base mensile**. Questo calcolo esclude le finestre di manutenzione programmata, che devono essere comunicate al team con almeno 24 ore di anticipo.

Performance: Per non interrompere il flusso di lavoro degli sviluppatori, le interazioni devono essere percepite come istantanee. Nello specifico:

- Il caricamento della dashboard principale con l'elenco delle issue deve avvenire in **meno di 1.5 secondi**.
- La sottomissione del form per una nuova issue deve fornire un feedback all'utente in **meno di 1 secondo**.

Sicurezza: Dato che il sistema gestisce credenziali di accesso, la sicurezza è un requisito critico:

- Tutta la comunicazione tra il client e il server deve avvenire obbligatoriamente tramite protocollo **HTTPS** per criptare i dati in transito.
- Le password degli utenti non devono mai essere memorizzate in chiaro. Devono essere salvate nel database utilizzando un algoritmo di **hashing robusto con salt** (es. `bcrypt`).

Manutenibilità: Per garantire l'evoluzione futura del progetto, il codice sorgente dovrà aderire a standard di codifica definiti per il linguaggio scelto. Inoltre, l'architettura del software dovrà seguire il principio di **separazione delle responsabilità** (*Separation of Concerns*), mantenendo la logica di presentazione, di business e di accesso ai dati in moduli distinti e testabili.

2.6 Glossario

Questo glossario è stato compilato per fornire definizioni chiare e non ambigue dei termini tecnici, metodologici e specifici del dominio utilizzati in questo documento, garantendo un linguaggio comune per tutti gli stakeholder del progetto.

Attore Un ruolo ricoperto da un utente (o un altro sistema) che interagisce con il software per raggiungere un obiettivo. Non rappresenta una persona specifica, ma una categoria di utenti (es. Amministratore).

Caso d'uso (Use Case) Una descrizione di una sequenza di azioni che un attore esegue utilizzando il sistema per raggiungere un fine specifico. Definisce un requisito funzionale.

Hashing Una tecnica crittografica utilizzata per memorizzare le password in modo sicuro. La password viene trasformata in una stringa di caratteri non reversibile (hash) attraverso un algoritmo, rendendola illeggibile anche in caso di accesso al database.

Issue Termine generico che indica una qualsiasi segnalazione registrata nel sistema. Può rappresentare un bug, una richiesta di nuova funzionalità (feature request) o un task generico da completare.

Mock-up a Bassa Fedeltà Un prototipo visuale statico (non interattivo) che si concentra sulla struttura, sulla disposizione degli elementi e sul flusso di navigazione di un'interfaccia, tralasciando deliberatamente i dettagli estetici come colori e font definitivi.

Personas Un archetipo di utente fittizio, creato sulla base di dati e analisi, che rappresenta un gruppo significativo di utenti reali. Serve a guidare le scelte di progettazione rendendo i bisogni degli utenti concreti e tangibili (es. "Luca" e "Giulia").

Requisito Funzionale Una specifica di ciò che il sistema deve *fare*. Descrive una funzionalità, un comportamento o un'interazione utente-sistema (es. "Il sistema deve permettere all'utente di creare una nuova issue").

Requisito Non-Funzionale Una specifica di *come* il sistema deve funzionare. Definisce un attributo di qualità o un vincolo, come performance, sicurezza, usabilità o affidabilità (es. "Il sistema deve caricare la dashboard in meno di 1.5 secondi").

SRS (Software Requirements Specification) Acronimo di "Specifica dei Requisiti del Software". È il documento formale (come questo) che descrive in modo completo i requisiti funzionali e non-funzionali di un sistema software.

Tabella di Cockburn Un formato testuale strutturato, proposto da Alistair Cockburn, per descrivere un caso d'uso in modo dettagliato. Include elementi come attori, precondizioni, postcondizioni e la sequenza di passi dello scenario principale e delle estensioni (scenari alternativi o di errore).

UML (Unified Modeling Language) Un linguaggio di modellazione visuale standardizzato, utilizzato in ingegneria del software per progettare, specificare e documentare sistemi software. Il Diagramma dei Casi d'Uso è un esempio di diagramma UML.