

UNIVERSITÀ DEGLI STUDI FEDERICO II

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE



CORSO DI LAUREA IN INFORMATICA

INSEGNAMENTO DI INGEGNERIA DEL SOFTWARE

ANNO ACCADEMICO 2025/2026

**Progettazione e sviluppo della piattaforma  
BugBoard26**

## Indice

<b>1</b>	<b>INTRODUZIONE</b>	<b>2</b>
1.1	Chi siamo . . . . .	2
<b>2</b>	<b>INGEGNERIA DEI REQUISITI</b>	<b>3</b>
2.1	Casi d'uso . . . . .	3
2.2	Individuazione delle personas . . . . .	4
2.3	Requisiti non funzionali e di dominio . . . . .	5
2.3.1	Requisiti non funzionali . . . . .	5
2.3.2	Requisiti di dominio . . . . .	5
2.4	Formalizzazione di un requisito . . . . .	6
<b>3</b>	<b>SYSTEM DESIGN</b>	<b>8</b>
3.1	Architettura del sistema . . . . .	8
3.2	Decomposizione del sistema . . . . .	8
3.3	Scelta delle tecnologie . . . . .	8
<b>4</b>	<b>SOFTWARE DESIGN</b>	<b>9</b>
4.1	Persistenza dei dati . . . . .	9
4.2	Strumenti di versioning . . . . .	10
4.3	Qualità del codice . . . . .	10

# 1 INTRODUZIONE

## 1.1 Chi siamo

Benvenuto su **BugBoard26**!

BugBoard26 è una piattaforma di *issues handling* che fornisce una soluzione unica per:

- Dividere in modo facile developer in progetti.
- Segnalare e gestire intuitivamente issue di vario tipo.
- Gestire in modo efficiente tutte le persone coinvolte in un progetto (anche non sviluppatori) tramite una gerarchia di utenze.

## Glossario

**issue** il "problema" identificato all'interno di un progetto che concerne l'utente.

**piattaforma** Vedi sistema.

**sistema** BugBoard26.

## 2 INGEGNERIA DEI REQUISITI

### 2.1 Casi d'uso

In questa sezione ci interesseremo all'individuazione dei casi d'uso. Come si può evincere dallo use case diagram riportato qui di seguito:

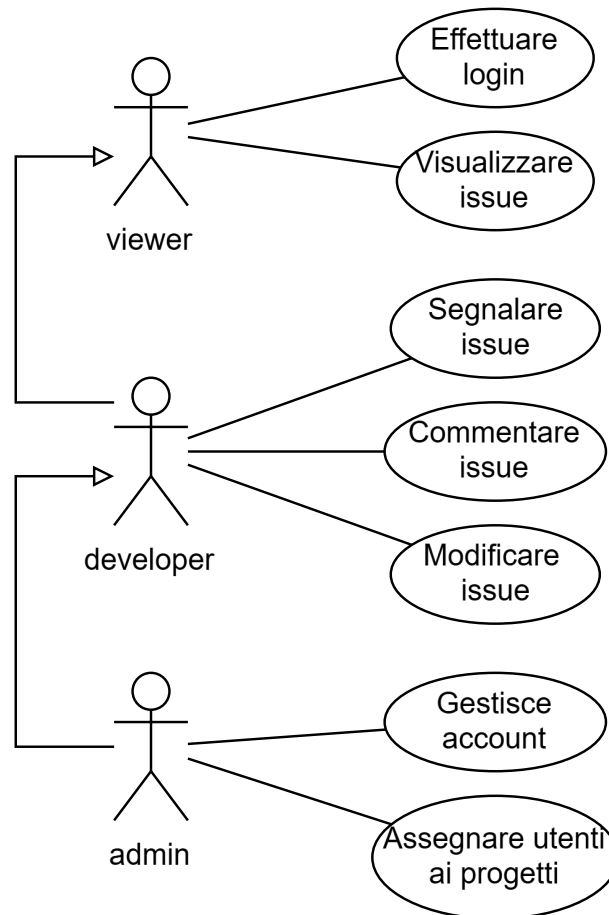


Figura 1: Use Case Diagram

Tutti gli utilizzatori della piattaforma possono essere divisi in tre grandi macrocategorie:

- **Viewer:** individuati anche nella figura di uno stakeholder. Sono utilizzatori, non necessariamente del settore, che hanno comunque interesse a visualizzare le issue legate al progetto senza poterle però aggiungere o modificare.
- **Developer:** rappresentano la stragrande maggioranza di utilizzatori della piattaforma. I developer sono coloro che contribuiscono attivamente all'individuazione e risoluzione delle issue.
- **Admin:** rappresentano l'estensione di un developer con permessi di creazione e gestione di altre utenze.

## 2.2 Individuazione delle personas

Ora esamineremo alcune personas che rispecchiano alcuni dei tipi di utilizzatori della nostra piattaforma.

Nome: Mark Party Età: 52 anni Posizione: Product Owner
<b>Obiettivi:</b> <ul style="list-style-type: none"><li>• Sarebbe molto utile poter vedere l'andazzo del team così da sapere in che direzione indirizzarlo e come gestirlo.</li></ul>
<b>Bio:</b> <p>Sono un economista italo-americano di Boston. Nell'arco della mia carriera mi sono ritrovato a gestire diverse start-up e gruppi di lavoro, nonostante non capisca molto di queste diavolerie informatiche, mi ritengo molto più capace a gestire e portare avanti prodotti</p>

Nome: Aleksander Lilia Età: 24 anni Posizione: Developer
<b>Obiettivi:</b> <ul style="list-style-type: none"><li>• Per lavorare in modo efficiente devo sapere quali problemi devo sistemare e magari avere del feedback dai miei colleghi.</li><li>• Nel caso dovessi trovare dei problemi, vorrei avere un modo comodo per segnalarli in modo dettagliato.</li><li>• Una volta risolti tali problemi vorrei poter segnalarlo al mio team.</li></ul>
<b>Bio:</b> <p>Sono un developer di Izdebki, dopo essermi laureato all'università di Cracovia mi sono trasferito a Napoli per lavoro e per amore. Sono grande amatore della filosofia "work smarter not harder" che cerco di applicare in ogni modo possibile.</p>

Nome: Pierrelouis Frascout

Età: 37 anni

Posizione: Team leader

**Obiettivi:**

- Voglio poter gestire i membri del mio team in modo chiaro ed efficiente.
- Voglio poter tenere traccia dei progressi fatti dal mio team e come si sta comportando.
- Voglio condividere con tutte le persone interessate, l'andamento del nostro team.

**Bio:**

Sono un developer di Izdebki, dopo essermi laureato all'università di Cracovia mi sono trasferito a Napoli per lavoro e per amore. Sono grande amatore della filosofia "work smarter not harder" che cerco di applicare in ogni modo possibile.

## 2.3 Requisiti non funzionali e di dominio

### 2.3.1 Requisiti non funzionali

I requisiti non funzionali da noi individuati sono:

- **Permanenza dei dati:** attraverso un database non MBaaS.
- **Utilizzo di linguaggi orientati agli oggetti.**
- **Implementazione di un modello Client-Server.**
- **Elevata manutenibilità.**
- **Efficienza e affidabilità:** non essendo la piattaforma safety-critical, limitazioni di tempo e memoria occupata sono da considerarsi standard e ragionevoli.

### 2.3.2 Requisiti di dominio

Non sono stati individuati requisiti di dominio particolarmente differenti da quelli forniti nella traccia.

## 2.4 Formalizzazione di un requisito

Qui di seguito riportiamo la formalizzazione di un requisito quale la visualizzazione di una issue, prima mediante il suo mockup:

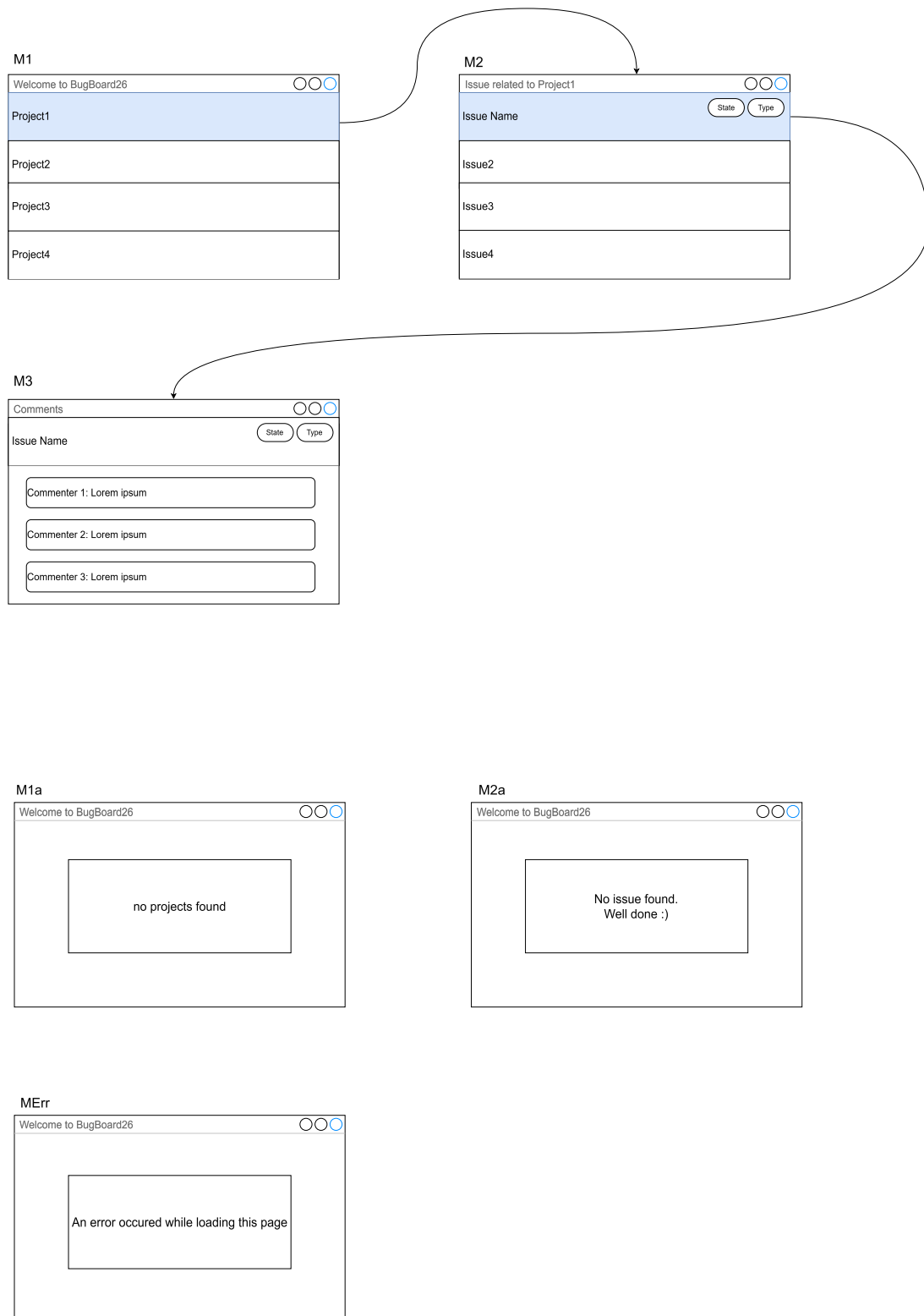


Figura 2: Visualizza issue

E qui di seguito riportiamo l'inerente tabella di Cockburn:

USE CASE	<i>Visualizza issue</i>		
<b>Goal</b>	Un utente vuole visualizzare una issue, le sue proprietà e i commenti.		
<b>Preconditions</b>	L'utente ha un account e si è autenticato.		
<b>Success end conditions</b>	Il sistema mostra la issue e i suoi commenti.		
<b>Failed end conditions</b>	Il sistema mostra una pagina di errore.		
<b>Primary actor</b>	Qualsiasi tipo di user.		
<b>Trigger</b>	L'utente fa accesso.		
<b>Main scenario</b>	<b>Step n.</b>	<b>Utente</b>	<b>Sistema</b>
	1		Mostra M1
	2	Clicca su un progetto	
	3		Mostra M2
	4	Clicca su una issue	
	5		Mostra M3
<b>Extension n° 1 (User has no projects)</b>	1a		Mostra M1a
<b>Extension n° 2 (Project has no issues)</b>	2b		Mostra M2b
<b>Extension n° 3 (Generic error)</b>	1,3,5 err		Mostra MErr



## 3 SYSTEM DESIGN

### 3.1 Architettura del sistema

L'architettura del sistema adottata per la realizzazione della piattaforma è di tipo **Client-Server**, più specificatamente a microservizi. Questa scelta, ideale per applicazioni web, disaccoppia nettamente il Frontend (lato utente) dal Backend (logica e dati su server).

I microservizi da noi individuati sono:

- **Servizio di gestione utenze:** si occupa della gestione delle utenze e dell'autenticazione.
- **Servizio di gestione issue:** si occupa della gestione delle issue e dei relativi commenti.

L'organizzazione del Backend in microservizi distinti è stata guidata da tre obiettivi principali:

- **Scalabilità Indipendente:** Consente di allocare risorse computazionali in modo mirato. È possibile, ad esempio, scalare orizzontalmente il servizio di gestione delle issue (soggetto a traffico più intenso) senza dover replicare inutilmente il servizio di gestione utenze.
- **Disaccoppiamento e Manutenibilità:** La suddivisione in moduli riduce la complessità del codice del singolo servizio. Questo favorisce uno sviluppo distribuibile, manutenibile, evolvibile e con test più mirati.
- **Tolleranza ai guasti:** Un eventuale errore critico in un microservizio non compromette necessariamente la disponibilità dell'intera piattaforma.

### 3.2 Decomposizione del sistema

Il sistema è composto dai seguenti elementi:

1. **Frontend:** interfaccia utente accessibile tramite browser web.
  - Login/Registrazione
  - Visualizzazione progetti e issue
  - Creazione/modifica progetti e issue
  - Gestione profili utente (solo per admin)
2. **Backend:** microservizi che gestiscono le funzionalità principali della piattaforma.
  - **Utenze:** gestione utenti e profili
  - **Gestione issue:** gestione progetti e issue
3. **Database:** sistema di gestione dei dati persistenti.
  - **Utenze:** memorizzazione dati utenti e profili
  - **Gestione issue:** memorizzazione dati progetti e issue

### 3.3 Scelta delle tecnologie

Per la realizzazione della piattaforma BugBoard26, abbiamo scelto le seguenti tecnologie:

- **Frontend:** *Angular.ts* per la costruzione dell'interfaccia utente, grazie alla sua modularità e facilità di integrazione con backend RESTful.
- **Backend:** *Java Spring* per la creazione dei microservizi, grazie alla sua robustezza.
- **Database:** *PostgreSQL* per la gestione dei dati relazionali, grazie alla sua affidabilità e scalabilità.

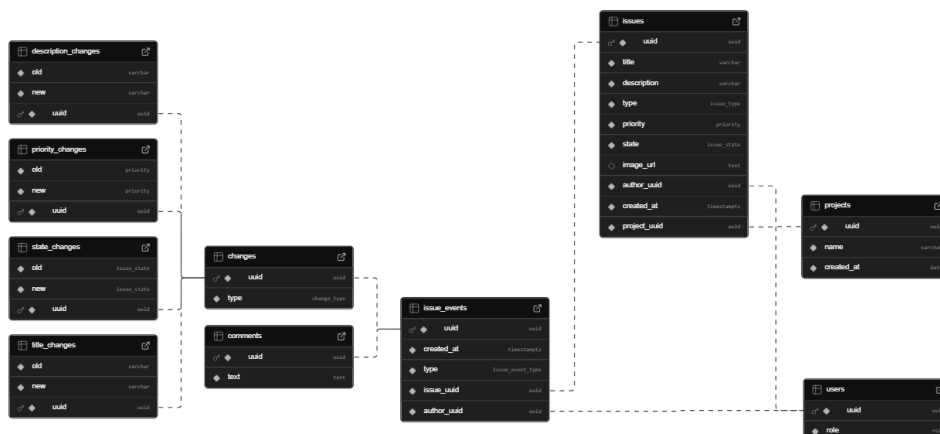
## 4 SOFTWARE DESIGN

### 4.1 Persistenza dei dati

Per la parte di persistenza di dati ci siamo affidati a Supabase, non per le funzionalità di MBaaS, ma come provider di database. Come annunciato in precedenza, il backend è diviso in due microservizi, ognuno dei quali fornito del proprio database. Qui di seguito riportiamo lo schema per entrambi:

users		
uuid	uuid	
name	varchar	
surname	varchar	
email	varchar	
password_hash	varchar	
is_admin	bool	
is_active	bool	

Questo è il database dello UsersService. La sua funzione principale è quella di contenere per intero i dati delle varie utenze. Separare le informazioni delle utenze dalle informazioni proprie e caratteristiche di BugBoard26 ci permette una certa scalabilità oltre a impostare BugBoard26 verso quella che potrebbe diventare una famiglia di servizi in un futuro ipotetico.



Questo è il database del BugBoardService, cuore pulsante della piattaforma. È qui che sono conservate tutte le informazioni riguardanti progetti, issue e interazioni degli utenti con queste. Questo possiede un sottoinsieme delle informazioni degli utenti, al fine di semplificare le operazioni intra-database e per permettere il suo corretto funzionamento anche nel caso di down del database dello UsersService.

## 4.2 Strumenti di versioning

Lo strumento di versioning da noi utilizzato è stato GitHub, scelta facile e dettata dalla sua diffusione e semplicità d'uso. È possibile visualizzare la repo di BugBoard26 al seguente link: <https://github.com/PierluigiFrascogna/BugBoard26>

Ovviamente, qui è anche possibile visualizzare ogni possibile statistica relativa a software e contributors, in modo molto più esaustivo di quanto potremmo mai offrire noi.

## 4.3 Qualità del codice

Per la generazione di report sulla qualità del codice la nostra scelta è ricaduta su SonarQube. Questa scelta è stata dettata dalla sua estensività, semplicità d'uso e la profonda integrazione con GitHub. Qui di seguito riportiamo il report sulla qualità del codice inerente al backend:

