CdL Triennale in Informatica

Programmazione Object-Oriented

Documentazione di progetto

Piattaforma Hackathon

Anno Accademico: 2025/2026

Matricola: N86005620

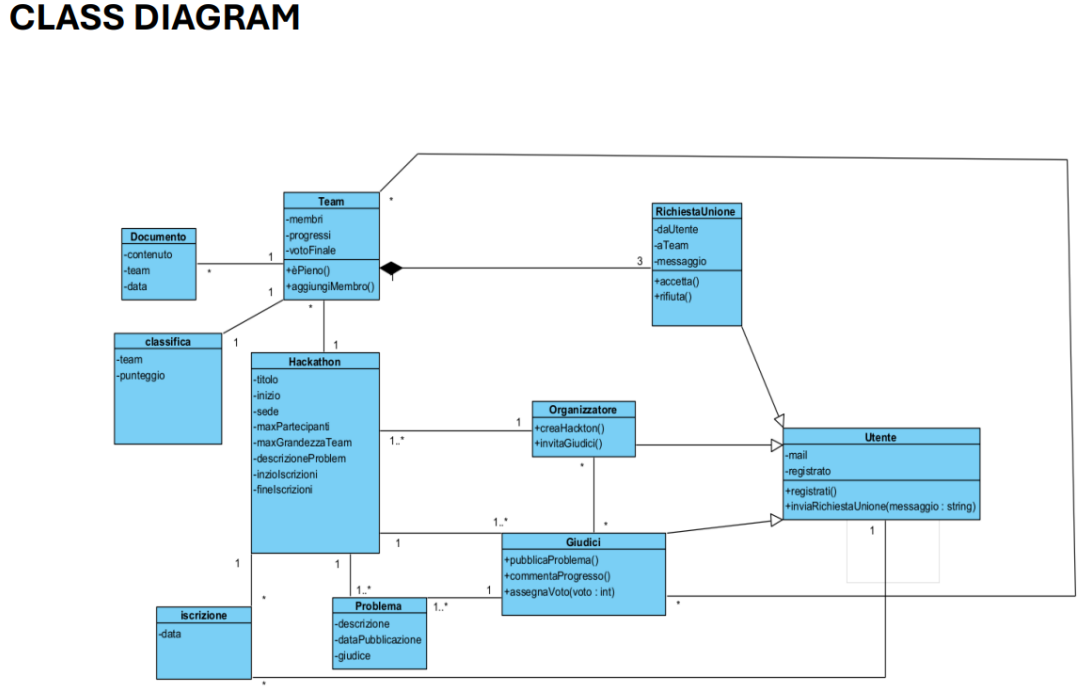
1. Obiettivo del sistema e requisiti

L'obiettivo del progetto è realizzare una piattaforma che supporti l'organizzazione e lo svolgimento di hackathon. Un hackathon è definito da un titolo identificativo, una sede e un intervallo temporale. Un organizzatore (utente registrato) seleziona e invita un insieme di giudici, apre le registrazioni e imposta vincoli come numero massimo di iscritti e dimensione massima dei team. Durante la fase di registrazione gli utenti possono iscriversi all'hackathon e formare team; alla chiusura delle iscrizioni i team diventano definitivi. All'inizio dell'evento i giudici pubblicano il problema da affrontare. Durante l'hackathon i team caricano aggiornamenti sui progressi sotto forma di documenti, che i giudici possono consultare e commentare. Al termine, ogni giudice assegna un voto (0–10) a ciascun team e la piattaforma pubblica la classifica.

2. Modello di dominio

Il modello di dominio descrive le entità principali del sistema e le relazioni tra di esse. Di seguito è riportato il diagramma delle classi (UML) utilizzato come riferimento per la traduzione in Java.

Diagramma delle classi (UML):



2.1 Scelte progettuali

- **Hackathon:** Entità centrale: dati dell'evento (titolo, sede, date) e vincoli (max iscritti, max team).
- **Utente:** Entità base identificata tramite email; da essa derivano i ruoli specializzati.
- **Organizzatore:** Crea hackathon e invita i giudici.
- **Giudice:** Pubblica problemi, commenta i progressi e assegna voti ai team.
- **Team:** Gruppo di utenti partecipanti; gestisce membri e progressi caricati.

- **Iscrizione:** Associazione tra utente e hackathon con data di iscrizione.
- **Documento:** Aggiornamento dei progressi caricato da un team durante l'hackathon.
- **Problema:** Descrizione del problema pubblicata dai giudici per l'hackathon.
- **Classifica:** Riga di classifica (team + punteggio totale) utilizzata per ordinare i risultati finali.
- **RichiestaUnione:** Richiesta di unione tra team a utente

3. Architettura software (BCE + DAO)

L'applicazione adotta un'impostazione BCE (Boundary–Control–Entity) per separare interfaccia, coordinamento della logica e modello dati, e utilizza il pattern DAO per isolare la persistenza su database.

3.1 Componenti

- **Boundary:** Interfaccia grafica Swing: mostra informazioni e raccoglie input.
- **Control:** Controller: coordina i flussi (wizard), applica le regole e invoca i DAO.
- **Entity:** Classi del dominio (package model) e DTO/Info per la visualizzazione (es. TeamInfo, CommentoInfo, InvitoTeamInfo).
- **DAO:** Interfacce dei servizi DB (package dao).
- **DAO Implementations:** Implementazioni PostgreSQL via JDBC (package daoImpl).
- **Database:** Gestione della connessione al DB (package database).

4. Database

Il sistema utilizza PostgreSQL per memorizzare utenti, hackathon, iscrizioni, team e membri, inviti, documenti di progress, commenti, problemi e voti. Le operazioni sul database sono incapsulate nei DAO e nelle relative implementazioni JDBC.

4.1 Tabelle principali

- utente
- hackathon
- iscrizione
- team
- team_membro
- invito_giudice
- voto
- documento
- commento
- problema

5. Istruzioni di esecuzione

Prerequisiti:

- JDK installato (versione coerente con il progetto).

- PostgreSQL attivo e database configurato con lo schema previsto.
- Driver JDBC PostgreSQL disponibile nel classpath del progetto.

Passi generali:

1. Verificare i parametri di connessione al DB nel package database (host, porta, nome DB, credenziali).
2. Avviare l'applicazione da Main e utilizzare le dashboard (Utente/Organizzatore/Giudice) per testare i flussi principali.

6. Repository

Repository GitHub: https://github.com/PerOper-s/HackatonGMC

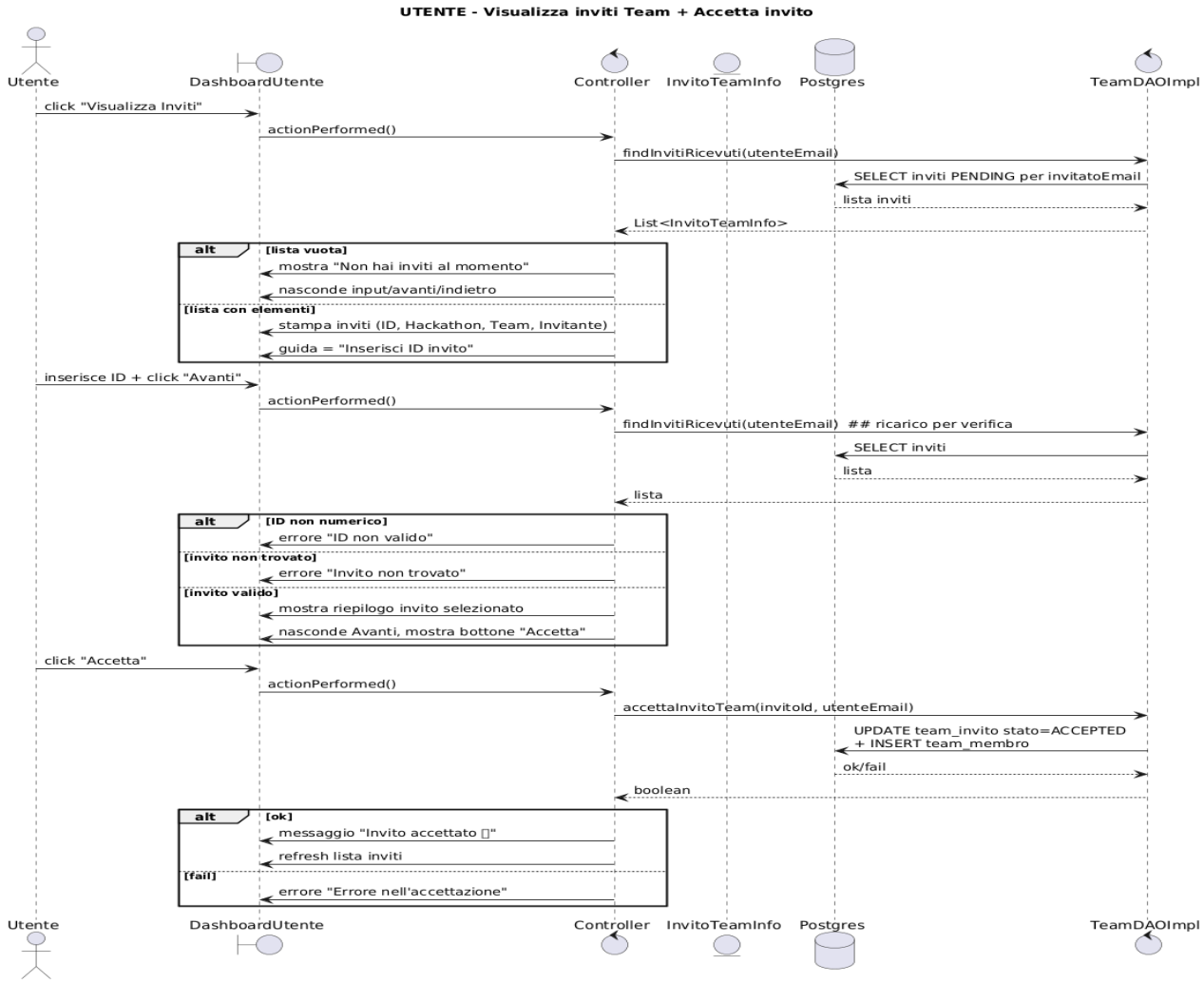
Nel repository sono presenti il codice sorgente, le implementazioni DAO/DAOImpl, la documentazione e gli eventuali file di supporto.

7. Diagrammi di sequenza

In questa sezione riporto due diagrammi di sequenza che mostrano due funzionalità centrali del sistema. L'obiettivo è far vedere chiaramente il flusso BCE + DAO: l'utente interagisce con la GUI (Boundary), la GUI passa l'evento al Controller (Control) e il Controller usa i DAO/DAOImpl per parlare con PostgreSQL.

7.1 Visualizza inviti Team e accetta invito (Utente)

Figura 1 – Diagramma di sequenza: Visualizza inviti Team e accetta invito (Utente).



Spiegazione

Qui l'utente entra nella dashboard e clicca su "Visualizza Inviti". Da lì parte un giro semplice: la GUI manda l'evento al Controller, e il Controller chiede al TeamDAO di recuperare gli inviti ricevuti da quell'utente (filtrati per email).

Il DAO interroga PostgreSQL e torna al Controller con una lista di InvitoTeamInfo (oggetti già pronti per la UI: id invito, team, hackathon, invitante e data).

Se la lista è vuota, mostro un messaggio tipo "non hai inviti" e non abilito i passaggi successivi. Se invece ci sono inviti, la dashboard stampa la lista e chiede l'ID dell'invito da gestire.

Quando l'utente inserisce l'ID e preme "Avanti", il Controller valida l'input (numero, esistenza nella lista, ecc.). Se è tutto ok, fa comparire il pulsante "Accetta".

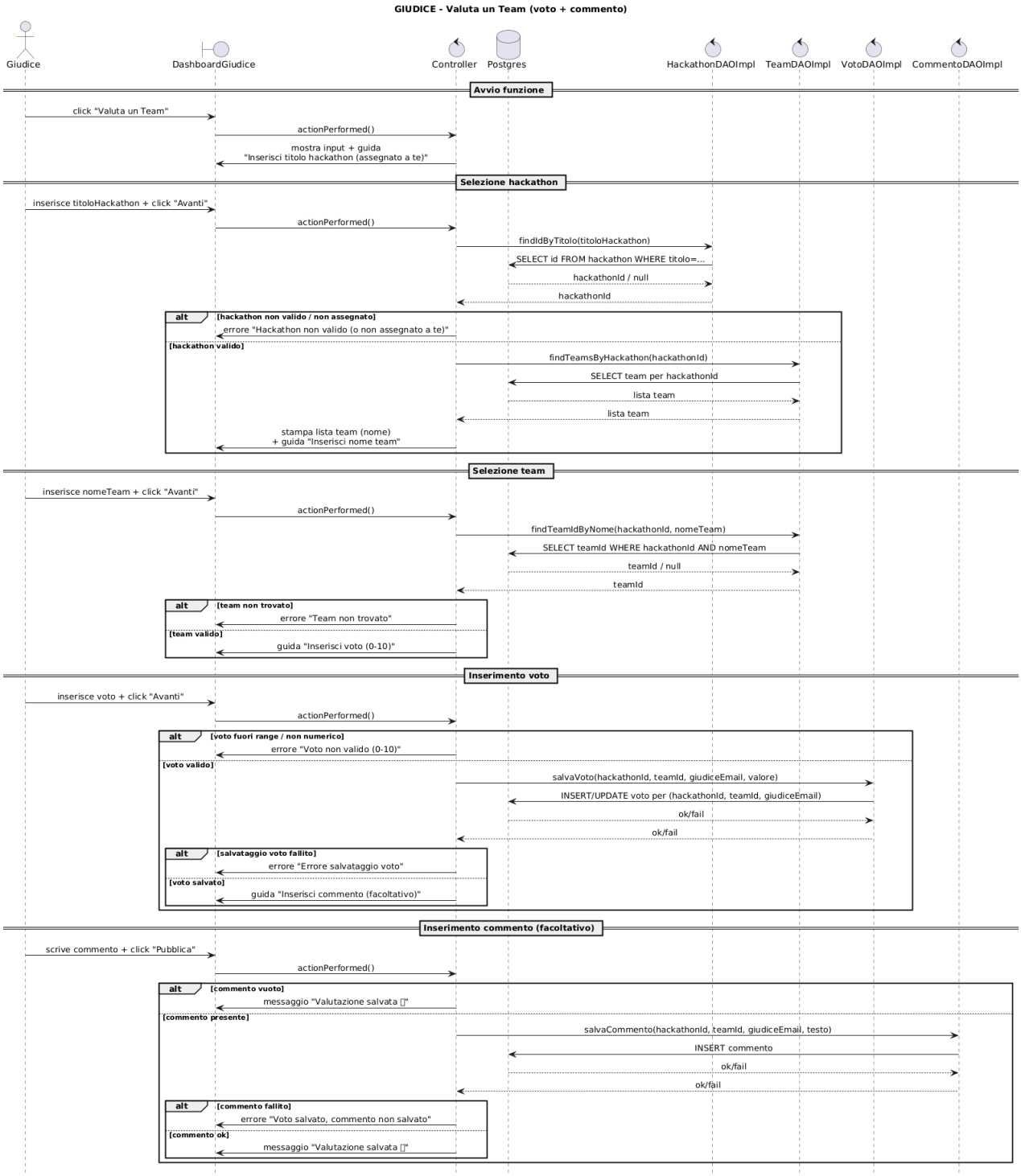
Premendo "Accetta", il Controller chiama il metodo del TeamDAO che registra l'accettazione: in DB viene aggiornato lo stato dell'invito (es. ACCEPTED) e viene inserito il record del nuovo membro nella tabella dei membri del team. Alla fine aggiorno la schermata e confermo l'operazione.

Punti importanti che si vedono nel diagramma:

- Separazione dei ruoli: GUI = Boundary, Controller = Control, DAO = accesso al DB.
- Uso di alt/if: lista inviti vuota vs lista piena; ID non valido vs invito non trovato vs invito valido; esito ok vs fail.
- Persistenza reale: l'accettazione aggiorna davvero il DB (stato invito + team).

7.2 Valuta un Team (Giudice: voto + commento)

Figura 2 – Diagramma di sequenza: Valuta un Team (Giudice).



Spiegazione

In questo flusso il giudice lavora sugli hackathon a lui assegnati. L'azione parte dalla dashboard giudice: clicco su "Valuta un Team" e il Controller guida i passaggi (tipo wizard).

Prima seleziono l'hackathon (titolo). Il Controller recupera l'ID via HackathonDAO e controlla che quell'hackathon sia effettivamente assegnato al giudice.

Poi seleziono il team. Il Controller usa TeamDAO per ottenere l'id del team e verifica che il team appartenga a quell'hackathon.

Inserisco il voto (0–10). Il Controller valida il range e salva su DB tramite VotoDAO. Ogni giudice salva il proprio voto: la classifica finale usa la somma dei voti.

Se voglio, inserisco anche un commento. Se il testo c'è, viene salvato con CommentoDAO. In output la UI mostra i commenti nel formato "Giudice email: commento".

Nel diagramma si vedono anche i casi di errore: hackathon non valido, team non trovato, voto fuori range, salvataggio fallito, con messaggi chiari in dashboard.

Punti importanti che si vedono nel diagramma:

- Il giudice passa sempre dal Controller: la GUI non parla mai direttamente col DB.
- Persistenza separata: voto e commento usano DAO diversi (VotoDAO e CommentoDAO).
- Validazioni lato Controller prima di scrivere nel DB (range voto, esistenza team/hackathon).
- Il risultato finale (classifica) dipende dai voti registrati: per questo il salvataggio è il cuore del flusso.