Relazione di Progetto: J-Time

Saverio Maria Piersigilli
Matricola: 119159

18 febbraio 2026

1 Descrizione delle Funzionalità Implementate

Il sistema **J-Time** è un'applicazione desktop per la gestione di progetti e attività, progettata per aiutare gli utenti a tracciare il lavoro, pianificare le giornate e monitorare i progressi. Le principali funzionalità implementate sono:

- **Gestione Progetti**:
  - **Creazione**: Possibilità di creare nuovi progetti specificando un nome.
  - **Visualizzazione**: Lista dei progetti con indicazione dello stato (Attivo/Completato).
  - **Chiusura**: Funzionalità di chiusura progetto, vincolata al completamento di tutte le attività associate.
  - **Riapertura**: Possibilità di riportare un progetto chiuso allo stato attivo.
  - **Eliminazione**: Rimozione di un progetto e di tutte le sue attività correlate.

- **Gestione Attività (Task)**:
  - **Aggiunta**: Inserimento di nuove attività con Titolo, Descrizione, Priorità (Bassa, Media, Alta, Critica), Data Pianificata e Stima Temporale (in ore).
  - **Dettaglio**: Visualizzazione dei dettagli del task, inclusa la descrizione e lo stato.
  - **Completamento**: Marcatura di un'attività come completata, con inserimento delle ore effettive impiegate.
  - **Eliminazione**: Rimozione di singole attività.

- **Pianificazione Giornaliera**:
  - Visualizzazione di tutti i task pianificati per una specifica data selezionata tramite calendario.
  - Calcolo automatico del carico di lavoro totale (ore stimate) per la giornata.
  - Segnalazione visiva (alert) in caso di sovraccarico di lavoro (superamento delle 8 ore giornaliere).
- **Reportistica**:
  - Generazione di report testuali riepilogativi per progetto.
  - Analisi dello scostamento (varianza) tra le ore stimate e le ore effettive per ogni attività completata.

2 Responsabilità Individuate e Architettura

L'architettura del sistema segue una separazione logica dei livelli, dove ogni componente ha una responsabilità ben definita.

2.1 Livello di Dominio (Model)

Rappresenta i dati core del business e le regole fondamentali intrinseche alle entità.

- `Project`: Responsabile di mantenere i dati del progetto (nome, stato) e la lista delle attività associate. Include logica di dominio semplice come l'associazione bidirezionale con i task.
- `Task`: Rappresenta la singola unità di lavoro. Mantiene lo stato dell'attività, priorità, date e tempi.
- `Tag`: Entità per la categorizzazione trasversale delle attività.
- `ProjectStatus` / `TaskStatus` / `Priority`: Enumerazioni che definiscono gli stati ammissibili e i livelli di priorità, garantendo consistenza nei dati.

2.2 Livello di Persistenza (Repository)

Responsabile dell'astrazione dell'accesso ai dati, nascondendo i dettagli della tecnologia sottostante (Hibernate/SQL).

- `Repository<T, ID>`: Interfaccia generica che definisce il contratto per le operazioni CRUD (Create, Read, Update, Delete). Segue il principio di segregazione delle interfacce.
- `AbstractHibernateRepository<T, ID>`: Implementa la logica comune di accesso al database tramite Hibernate, gestendo le transazioni e le sessioni. Evita la duplicazione di codice per le operazioni standard.
- `ProjectRepository` / `TaskRepository`: Implementazioni concrete che estendono la classe astratta per entità specifiche. Responsabili di eventuali query personalizzate sul DB.

2.3 Livello di Servizio (Service)

Cuore della logica di business dell'applicazione. Coordina le operazioni, applica i vincoli e orchestra i repository.

- `ProjectService`:
  - Gestisce il ciclo di vita del progetto.
  - Applica il vincolo di business per cui un progetto non può essere chiuso se contiene attività non completate.
- `TaskService`:
  - Gestisce le operazioni sui task.
  - Responsabile dell'aggiornamento coerente delle relazioni tra Task e Progetto
  - Registra i tempi effettivi al momento del completamento.
- `PlanningService`:
  - Responsabile della logica di pianificazione.
  - Calcola il carico di lavoro giornaliero.
  - Determina se una giornata è in "sovraccarico" rispetto ai limiti definiti.
- `ServiceLocator`:
  - Applica il pattern Singleton per fornire un punto di accesso centralizzato ai servizi.
  - Gestisce l'inizializzazione e l'iniezione delle dipendenze.
- `ProjectSummaryReport`:
  - Implementa `ReportStrategy`.
  - Responsabile della logica di formattazione e calcolo dei dati per il report di riepilogo.

2.4 Livello di Presentazione (View/Controller)

Gestisce l'interazione con l'utente e la visualizzazione dei dati.

- `MainController`:
  - Orchestra la vista principale.
  - Gestisce gli eventi UI (click, selezioni).
  - Collega i dati del modello alle tabelle e liste dell'interfaccia (Data Binding).
- `TaskDialogController`:
  - Responsabile della gestione del form di input per i nuovi task.
  - Valida e converte l'input utente in oggetti di dominio.

3 Organizzazione dei Dati e Persistenza

La persistenza è garantita tramite **Hibernate ORM** (Object-Relational Mapping) su un database embedded **H2**.

- **Database Embedded**: Il database è salvato localmente su file (nella cartella `data`), garantendo che i dati siano preservati tra i riavvii dell'applicazione senza richiedere un server database esterno.
- **Mapping Trasparente**: Le classi di dominio (`Project`, `Task`, etc.) sono annotate con JPA (`@Entity`, `@Table`, ecc.). Hibernate gestisce automaticamente la creazione schema e la traduzione da oggetti Java a record SQL.
- **Transazionalità**: Le operazioni di modifica (scrittura/aggiornamento/cancellazione) sono incapsulate in transazioni gestite da `AbstractHibernateRepository`, garantendo l'integrità dei dati.
- **Gestione Sessioni**: `HibernateUtil` fornisce una `SessionFactory` configurata, ottimizzando la gestione delle connessioni al DB.

4 Integrazione di Nuove Funzionalità

Il sistema è stato progettato seguendo principi di modularità e pattern specifici per facilitare l'estensione:

- **Nuovi Report**: Grazie all'uso del pattern **Strategy** (`ReportStrategy`), è possibile aggiungere nuovi formati di report (es. PDF, CSV, o analisi statistiche diverse) semplicemente creando una nuova classe che implementa l'interfaccia, senza modificare il codice esistente nel controller.
- **Nuove Entità**: L'architettura Repository generica permette di aggiungere nuove entità (es. *Utente*, *Team*) creando semplicemente la classe modello e estendendo `AbstractHibernateRepository`, ereditando immediatamente tutte le funzionalità CRUD.
- **Logica di Business Estesa**: Nuove regole o servizi possono essere aggiunti creando classi Service dedicate e registrandole nel `ServiceLocator`, mantenendo separata la logica di presentazione da quella di business.
- **Gestione Flessibile tramite Tag**: Il sistema supporta l'uso di **Tag** per categorizzare le attività. Questa funzionalità è progettata per essere estensibile: nuovi tipi di tag o logiche di filtraggio basate su di essi possono essere implementati senza modificare la struttura core dei Task, permettendo una classificazione dinamica e personalizzabile del lavoro.
- **Interfaccia Utente**: L'uso di **JavaFX** con **FXML** separa la definizione del layout dalla logica di controllo. Nuove schermate possono essere disegnate in SceneBuilder e collegate a nuovi controller senza impattare le viste esistenti.