1. Software Life Cycle

Per lo sviluppo dell'applicazione ho scelto di adottare un metodo di sviluppo agile, dato che ho svolto il progetto da solo. Questo approccio mi ha permesso di concentrarmi sull'implementazione incrementale di piccole parti del software, mantenendo il sistema sempre funzionante.

Il processo di sviluppo è stato fortemente basato sulla prototipazione. Ho costruito gradualmente un prototipo, aggiungendo funzionalitàprogressivamente, partendo da una versione iniziale e via via migliorandola e ampliandola. In questo modo, i requisiti e le conseguenze delle scelte implementative sono emersi e sono stati migliorati man mano che il lavoro procedeva.

Ogni nuova funzionalità implementata è stata sottoposta a test manuali per garantire il corretto funzionamento e l'integrazione con le funzionalità esistenti. Questo metodo di test mi ha aiutato a individuare e risolvere prontamente eventuali problemi.

2. Configuration Managment

Per la gestione della configurazione del progetto, ho utilizzato la CLI di Git e ho caricato il codice sulla piattaforma GitHub. Nonostante abbia svolto il progetto in autonomia, ho creato un account secondario per simulare una separazione dei compiti e testare meglio il flusso di lavoro.

Il codice sorgente è organizzato nella cartella code, mentre la documentazione è raccolta nella cartella docs.

Per quanto riguarda i branch, ho mantenuto la seguente struttura:

- Branch **main**: Contiene la versione finale e stabile del progetto.
- Branch **dev**: Utilizzato come branch principale per lo sviluppo.

Ogni volta che doveva essere implementata una nuova funzionalità, creavo un branch dedicato su cui lavorare e testare le modifiche. Una volta completata la funzionalità, aprivo un issue e una pull request per eseguire il merge nel branch

dev. Questo approccio mi ha permesso di mantenere il codice organizzato e di tracciare chiaramente le modifiche e le implementazioni.

3. People Management and Team Organization

Il progetto è stato sviluppato seguendo una metodologia agile, che ha conferito maggiore flessibilità e rapidità al processo di sviluppo. Nonostante abbia lavorato da solo, ho simulato una suddivisione delle attività per meglio strutturare il lavoro.

In particolare:

- **Account principale**: Responsabile della gestione del database e del controller.
- **Account secondario**: Responsabile del model e della view.

Conformemente ai principi delle metodologie agili, ho adottato il concetto di proprietà collettiva del codice. Questo significa che, sebbene ci fosse una suddivisione delle attività, alcune operazioni sono state eseguite indipendentemente dalla divisione dei ruoli, secondo le necessità del momento.

Il lavoro è stato organizzato attraverso la creazione di una lista di funzionalità da implementare, con un focus particolare sui requisiti funzionali piuttosto che sull'estetica. Questo approccio ha garantito che il software fosse sempre in uno stato funzionante e che le funzionalità principali fossero sviluppate in modo efficace e tempestivo.

4. Software Quality

Come previsto dal project plan si è cercato di seguire i principi di qualità definiti da *McCall*. Di seguito sono elencati alcuni attributi seguiti suddivisi per il focus.

Operatività

- **Correttezza:** L'applicazione soddisfa i requisiti e le specifiche definite all'inizio del progetto. Eventuali miglioramenti (sopratutto estetici)

potranno essere aggiunti facilemente al software, grazie anche all'architettura MVC.
- **Affidabilità:** L'applicazione è stata sottoposta a diversi test, risultando con il minor numero possibile di errori.
- **Usabilità:** Il prodotto è progettato per essere intuitivo e non richiede competenze particolari per essere utilizzato.

Revisione

- **Manutenibilità**: C'è stato un particolare sforzo per scrivere un codice funzionante e al contempo leggibile e facilmente manutenibile.
- **Flessibilità**: Grazie all'architettura del codice, il software è flessibile, garantendo facilità di future implementazioni.

Transizione

- **Portabilità**: L'applicazione è stata scritta con Java, pertanto è utilizzabile in tutti gli ambienti che lo supportano a partire da java 17.

5. Requirenment Engineering

Prima di iniziare a scrivere codice sono stati delineati i requisiti di base a partire da alcuni casi d'uso che venivano in mente. Inoltre essendo un'app per la gestione di task ed eventi come ce ne sono già disponibile è stata anche effettua un'analisi del dominio per capire alcuni requisiti funzionali essenziali.

Mano a mano che il progetto procedeva sono state poi scoperte alcune funzionalità in più che potevano essere aggiunti.

I requisiti sono stati poi categorizzati secondo il modell MoSCoW:

Must Have:

- Possibilità di creare task ed eventi
- Possibilità di modificare task ed eventi creati
- Possibilità di eliminare task ed eventi

Should Have:

- Possibilità di scegliere tra varie opzione di visualizzazione: task, eventi o entrambi.

- Highlight delle task scadute non completate

Could have:

- Una schermata di login per rendere l'applicazione più sicura
- Un interfaccia gradevole

Won't have:

- Capacità di connettersi a database in cloud
- Capacità di connettersi a servizi come google per la sincronizzazione

6. Modelling

Il progetto è stato modellato in diversi diagrammi UML:

- diagramma dei casi d'uso
- diagramma delle attività
- diagramma delle macchine a stati
- diagramma di sequenza
- diagramma delle classi

Il progetto poi è suddiviso in 4 componenti principali: Il contreller, la view, il model e il database secondo il modello MVC.

Il controller ha il compito di interfacciarsi sia con il database, per il salvataggio, update o elimazione di task od eventi, che di fare comunicare gli eventi che accadono sulla view con le specifiche azioni che avvengono sul modello.

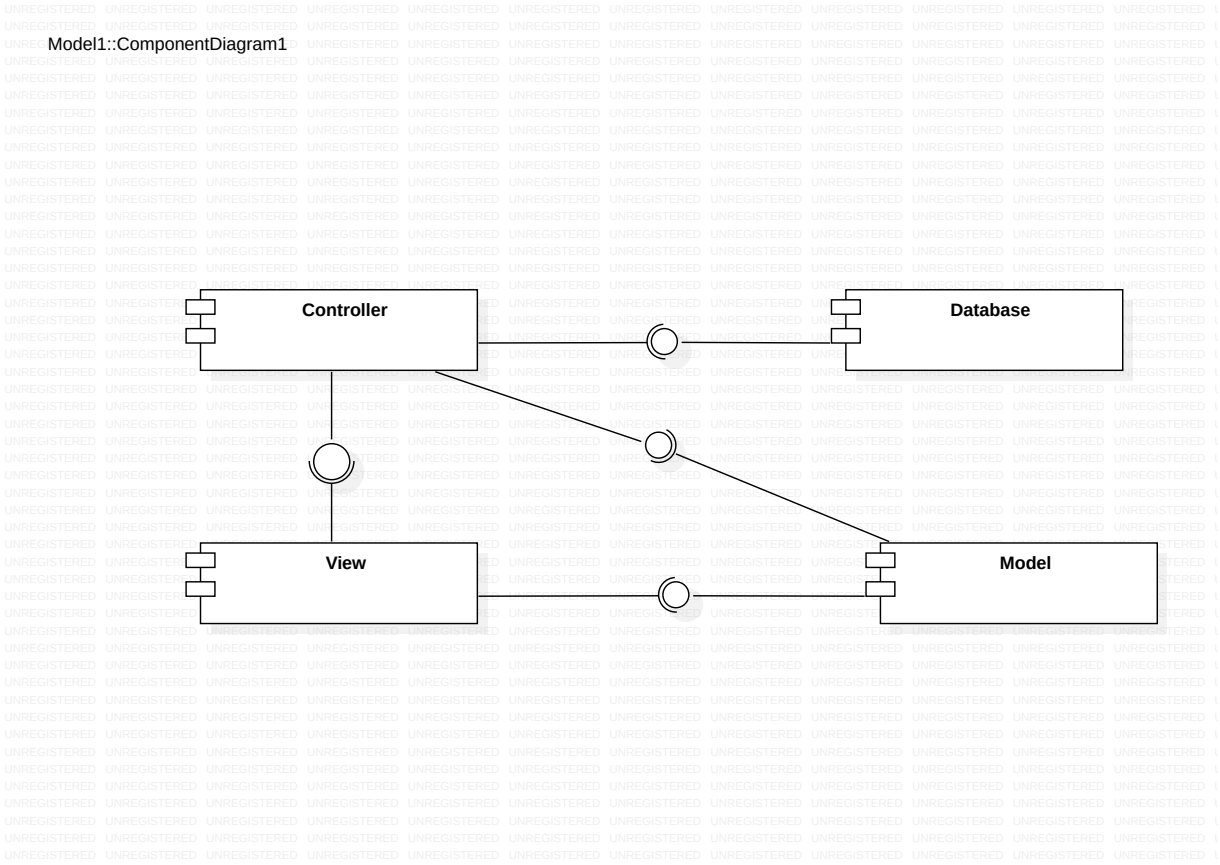
7. Software Architecture

Il software è stato progettato seguendo per quanto possibile dalla poca mia poca esperienza il modello MVC, suddivendo ogni componente nel proprio package e cercando di sfruttare l'information hiding.

Model: Nel package model sono contenute le classi che definiscono la struttura dei dati.

View: Gestisce le classi che si occupano della visualizzazione, prendendo i dati a partire dal modello. Le azioni possibili dalla view vengono controllate dal controller. Separando la view dal modello è possibile riscrivere la view con semplicità.

Controller: Nel controller package sono contenute le classi necessarie per la gestione degli eventi che avvengono sulla view, in particolare l'update, il salvataggio e l'eliminazione di task od eventi connettendosi anche al database.



8. Software Design

Di seguito sono presenti alcune viste UML che descrivono il design del softare.

Diagramma delle classi:

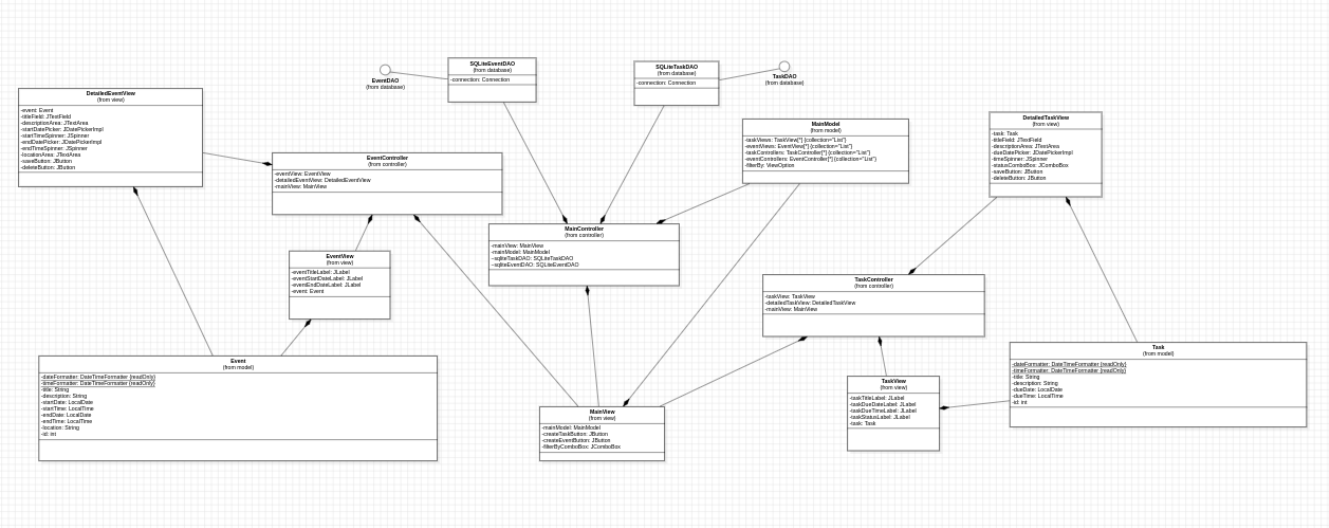


Diagramma di sequenza: Mostra la sequenza dei messaggi per la creazione o update di un task.

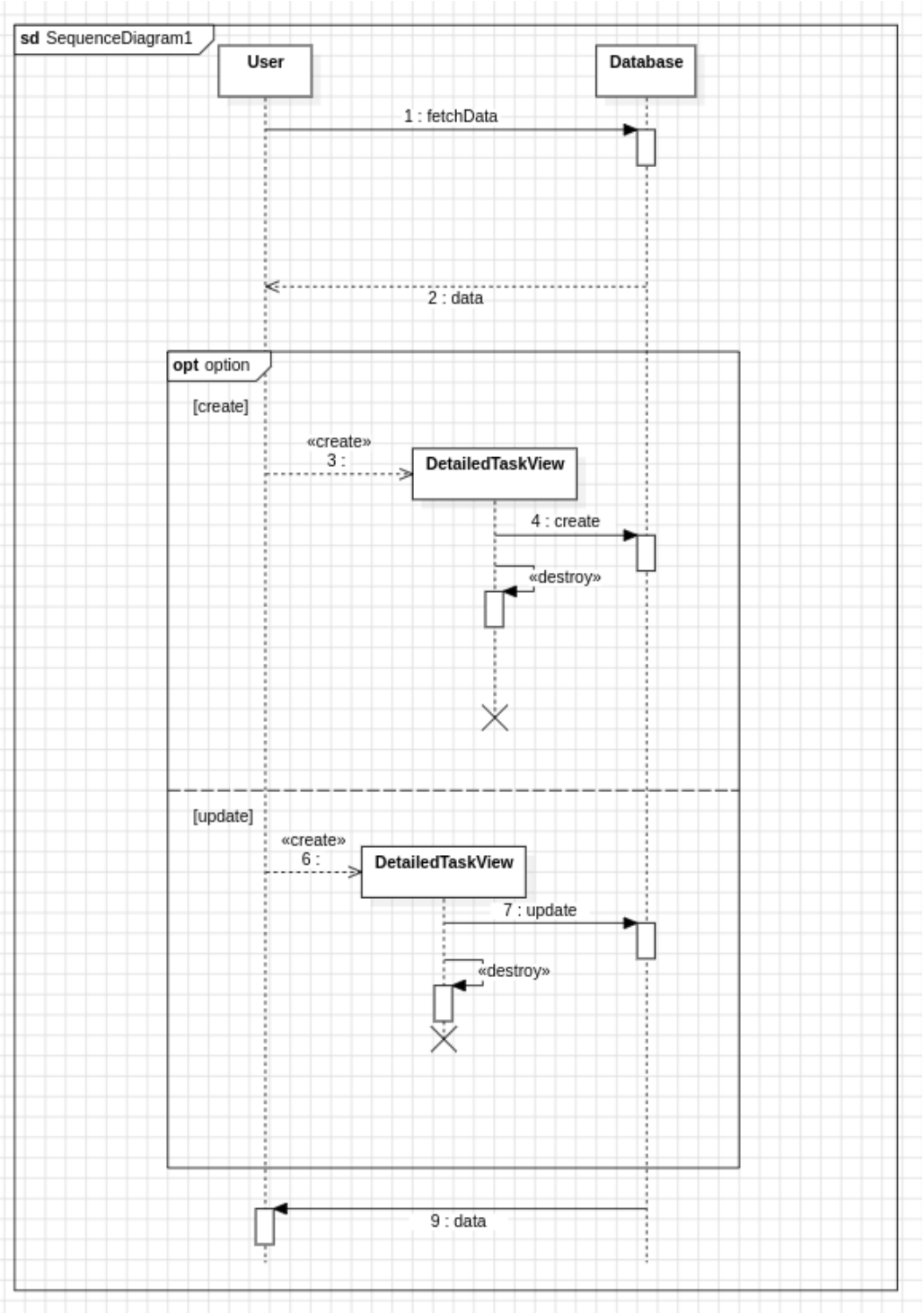


Diagramma delle attività: Mostra la sequenza di attività per la creazione, update o eliminazione di un task od evento.

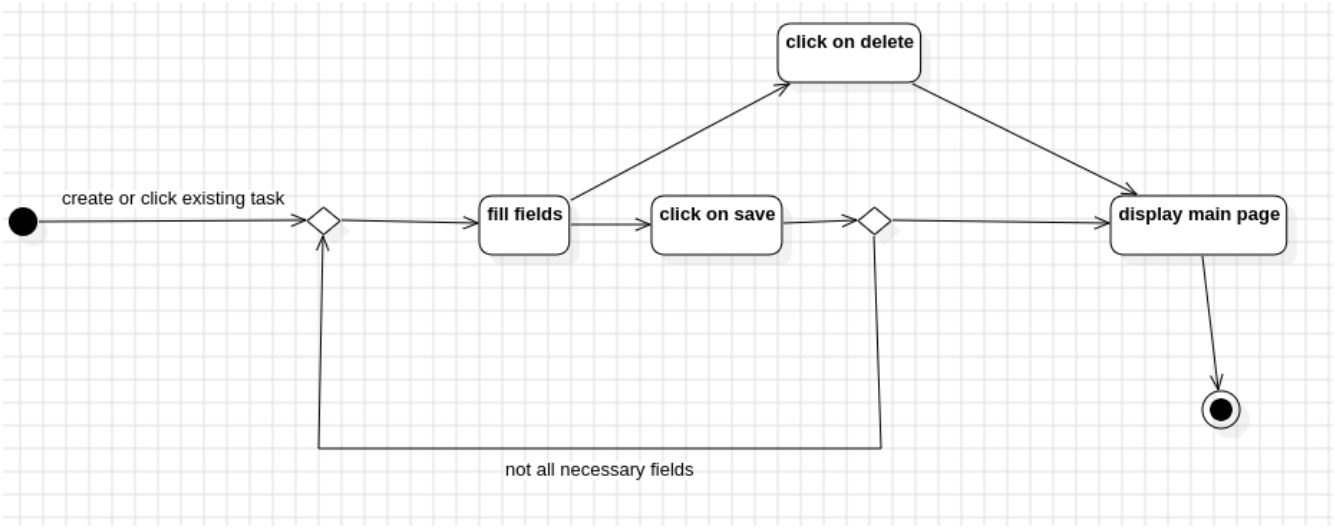


Diagramma state machine: Mostra gli stati dell'applicazione nel processo di visualizzazione, creazione o eliminazione di task od eventi.

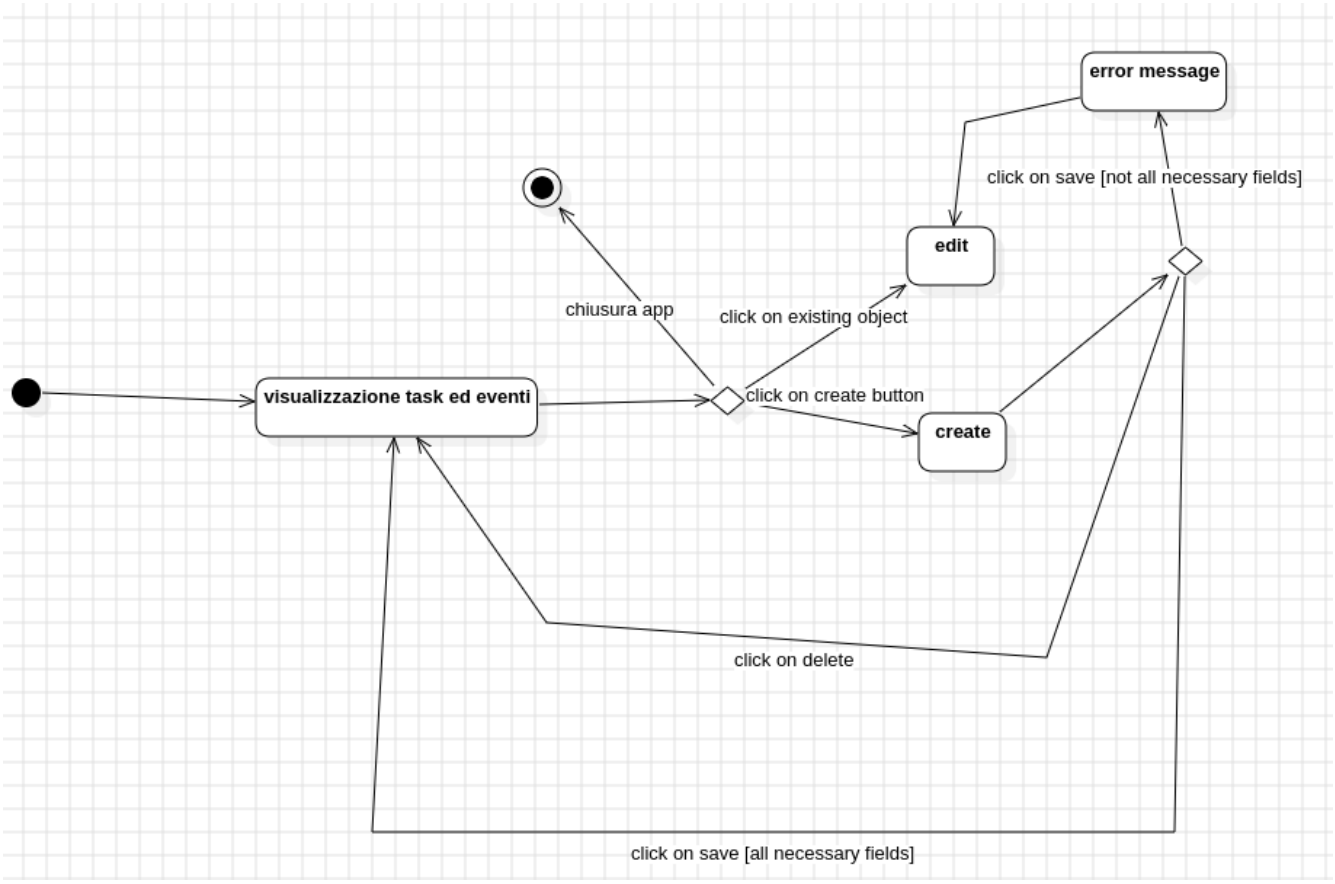
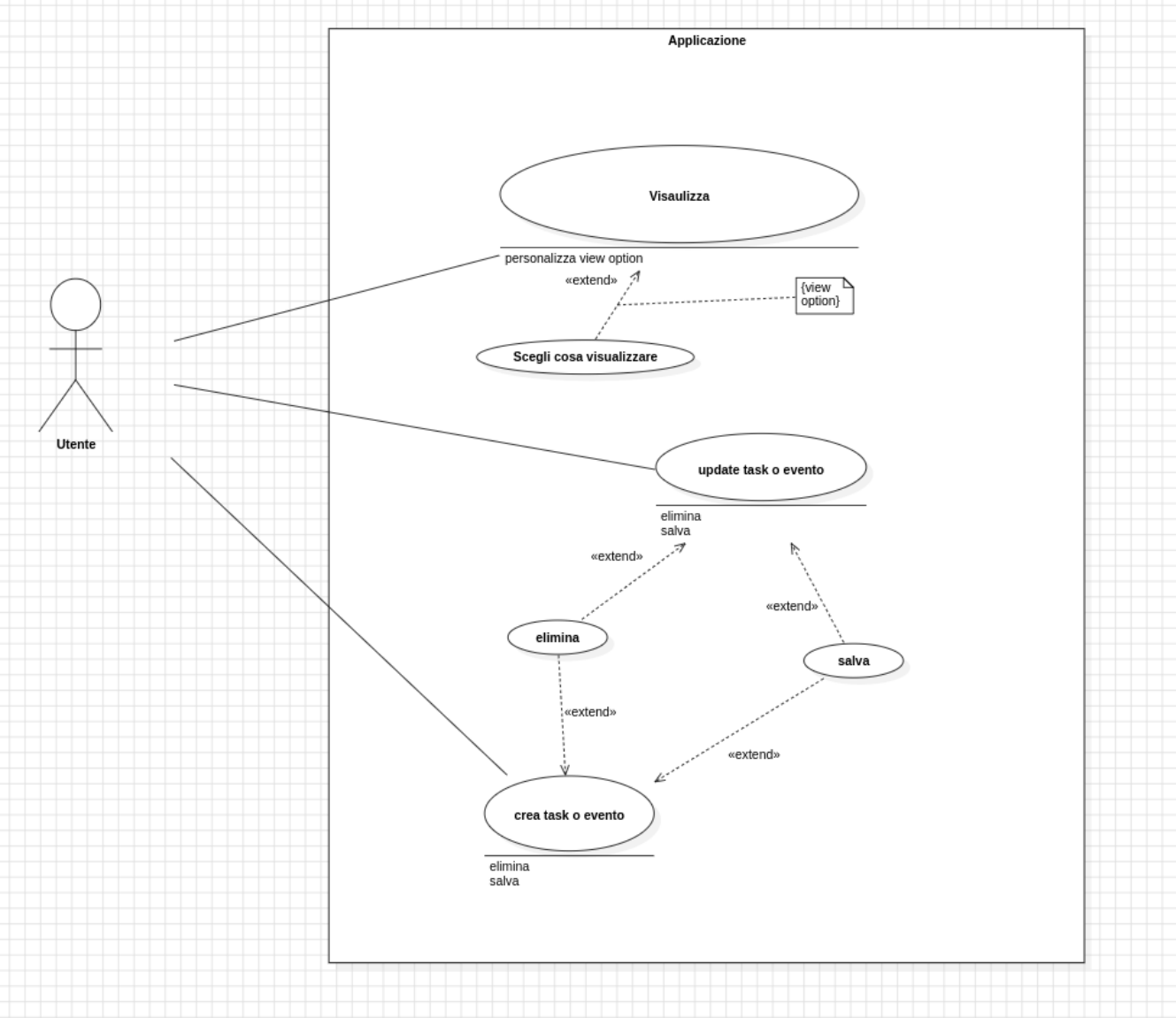


Diagramma dei casi d’uso:



8.1 Design Pattern

Singleton Design Pattern

Il Singleton Design Pattern è stato adottato per la gestione della creazione del database. Questo pattern assicura che una sola istanza della classe possa essere creata e utilizzata durante l'intera esecuzione dell'applicazione. È stato

implementato utilizzando un costruttore privato e un metodo pubblico che restituisce l'unica istanza della classe, se non è già stata istanziata.

DAO (Data Access Object) Design Pattern

Le classi SQLiteTaskDAO e SQLiteEventDAO seguono il DAO Design Pattern. Questo pattern separa la logica relativa alla gestione della connessione al database e l'esecuzione delle operazioni CRUD (Create, Read, Update, Delete) dalla logica business gestita dal modello e dal controller. In questo modo, il codice relativo all'accesso ai dati è isolato e riutilizzabile, migliorando la manutenibilità e la modularità dell'applicazione.

9. Software Testing

La maggior parte dei test sono stati svolti manualmente, testando le varie funzionalità alla ricerca di errori. Questo per rendere l'applicazione sempre funzionante ogni volta che veniva implementata una nuova funzionalità.

Inoltre ho scritto anche un test d'unità automatico con Junit per controllare il funzionamente del comparatore che riordina.

10. Software Maintenance

La manutenzione del codice è stata svolta tramite diverse attività di refactoring, quali:

- ristrutturazione di alcuni parti di codice, sopratutto nella parte di inizializzazione della schermata principale, sfruttando dei metodi invece che riscrivere codice duplicato
- ridenominazione di alcuni metodi per rendere il nome più esplicativo