



**Corso di Laurea Magistrale in Ingegneria Informatica**

**Software Engineering 2023/2024**

# **Pre-Game phase**

## **IFTTT project**

**Gruppo 04**



Andrea Alberti

[a.alberti2@studenti.unisa.it](mailto:a.alberti2@studenti.unisa.it)

Marco Bove

[m.bove23@studenti.unisa.it](mailto:m.bove23@studenti.unisa.it)

Giulia Minichiello

[g.minichiello9@studenti.unisa.it](mailto:g.minichiello9@studenti.unisa.it)

Domenico Schettini

[d.schettini5@studenti.unisa.it](mailto:d.schettini5@studenti.unisa.it)

## Tool utilizzati:

- NetBeans ;
- Scene Builder;
- Linguaggio Java con jdk 8 e junit 4;
- Trello Dashboard;
- GitHub repository.

## Application Description

Viene presentato il modello di una applicazione che permette l'automatizzazione di Task semplici simile al web service IFTTT (If This Than That).

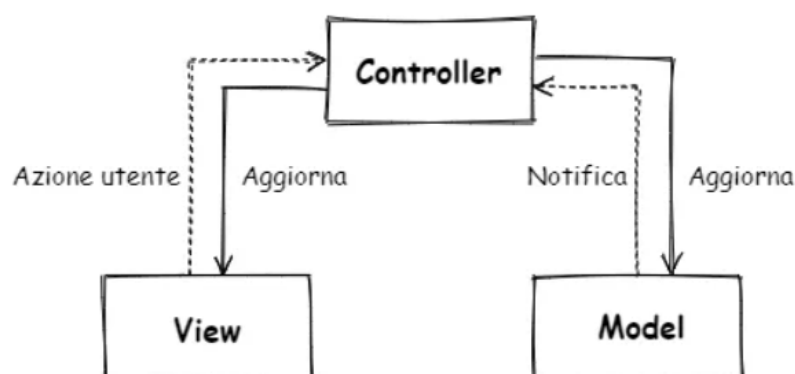
Ad ogni Task, chiamato regola, è associato un trigger e un'azione.

- La verifica della condizione associata al trigger rende la regola attiva
- all'attivazione della regola vengono eseguite le azioni associate se presenti.
- sono consentite le operazioni di modifica, eliminazione e abilitazione della regola.



## Architectural Design Pattern

Viene scelto, per la realizzazione dell'applicazione, il pattern MVC (*Model View Control*) che prevede la suddivisione di un'interfaccia grafica in tre parti:

- Model, che contiene i dati e la logica dell'applicazione
- View, che contiene rappresenta i dati
- Controller, che in base all'input dell'utente aggiorna Model o View



## Design HomePage interfaccia grafica


NAME	STATE
RULE 1.	
RULE 2.	

ADD

MODIFY

REMOVE

RULE1. ADDED.  
RULE2. ADDED.  
RULE 2 DISABLED



## **View Structure**

La Vista sarà la parte che interagisce con l'utente. Qui ci sarà l'interfaccia grafica dell'applicazione desktop.

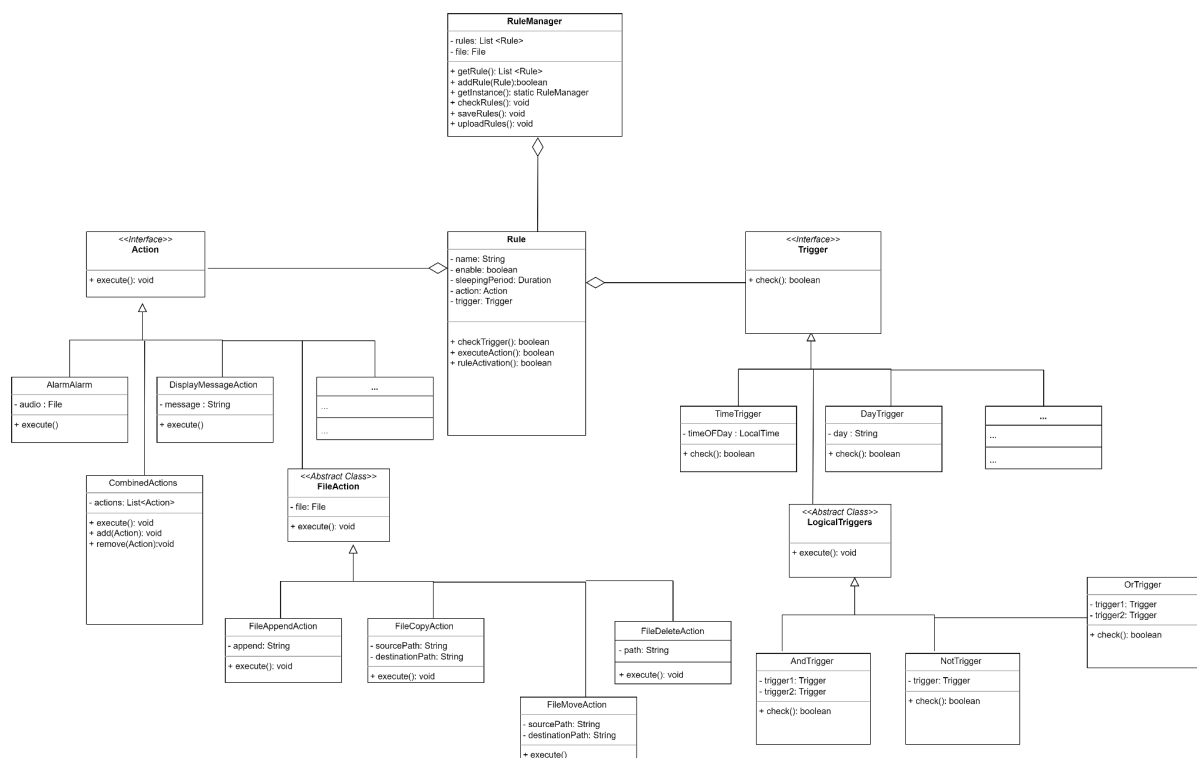
- L'app si presenta con una home page iniziale
- Dalla home page iniziale avviene l'accesso alle altre finestre interattive .
- Nella home page saranno presenti:
  - la tabella in cui sono presenti le regole definite.
  - pulsanti per:
    - aggiunta/eliminazione/modifica delle regole;
    - creazione contatori / accesso alla pagina dedicata ai contatori.
- Dalla home page iniziale avviene l'accesso alle altre finestre interattive per le altre funzionalità del programma .
- Premendo il pulsante di aggiunta di una regola si accede alla finestra di creazione della stessa, dalla quale è possibile procedere alla definizione dei trigger e delle azioni.
- Inoltre selezionando una regola si accede alla visualizzazione delle azioni e dei triggers ad essa collegati.

## Model Structure

Il Model rappresenterà la logica di business e i dati dell'applicazione. In questo caso, si dovrebbero avere classi che gestiscono le regole definite dall'utente, i contatori e la logica di esecuzione delle azioni quando una regola viene attivata dalla condizione di un trigger. Queste classi gestiranno anche il salvataggio e il caricamento delle regole su un file.

Per quanto riguarda l'applicazione delle funzionalità richieste:

- Gestione delle regole: Il Model conterrà una struttura dati che memorizzerà le regole create dall'utente, inclusi trigger, azioni e configurazioni di esecuzione.
- Periodicità delle verifiche delle regole: Il Model include un meccanismo per controllare periodicamente se i trigger delle regole sono soddisfatti.
- Persistenza dei dati: Le regole saranno salvate su file e caricate automaticamente all'avvio dell'applicazione. Questa gestione del file sarà parte del Model.
- Manipolazione delle azioni: Il Model include la logica per eseguire azioni come la scrittura su file, l'esecuzione di programmi esterni, la gestione dei contatori e la visualizzazione di messaggi.
- Variabili e contatori: Il Model terrà traccia dei contatori definiti dall'utente, permettendo la loro visualizzazione e modifica tramite l'interfaccia.



## **Controller Structure**

Il Controller fa da ponte tra la Vista e il Model. Sarà responsabile di gestire le azioni dell'utente nella Vista e di aggiornare il Model di conseguenza. Ad esempio, quando l'utente crea una nuova regola o modifica un contatore, il Controller si assicurerà che queste modifiche siano riflesse nel Model e viceversa.

## **Design Pattern**

Design Pattern Strutturali:

- **Composite :**
  - Il pattern Composite per i trigger permette di trattare i trigger singoli e quelli composti allo stesso modo. Questo permette di costruire logiche avanzate senza dover ripensare l'interfaccia o la gestione di base.
  - Il pattern Composite sulle azioni permette di trattare le azioni singole e quelle composte in modo uniforme. Puoi eseguire operazioni sulle azioni singole e sulle composizioni senza differenziazioni, semplificando la gestione.

Design Pattern Creazionali:

- **Factory Method:**
  - Utilizzato per la gestione di diversi tipi di trigger, fornendo un modo standardizzato per crearli senza specificare direttamente le classi concrete, delegando la creazione dei trigger alle sottoclassi
  - Analogamente al caso dei trigger, se si hanno diverse tipologie di azioni con comportamenti specifici viene così fornito un modo standardizzato per crearli senza conoscere le classi concrete.
- **Singleton**
  - utilizzato per il FileManager : per avere sicurezza che in qualsiasi punto dell'applicazione si utilizzi il FileManager, operando sulla stessa istanza, evitando conflitti di accesso o duplicazione di risorse per la gestione dei file.
  - Avendo un'unica istanza del RuleManager, si garantisce che la gestione delle regole e delle azioni sia centralizzata e coerente. Tutte le parti dell'applicazione che utilizzano il RuleManager operano sulla stessa istanza, evitando discrepanze e problemi derivanti da più istanze separate.

