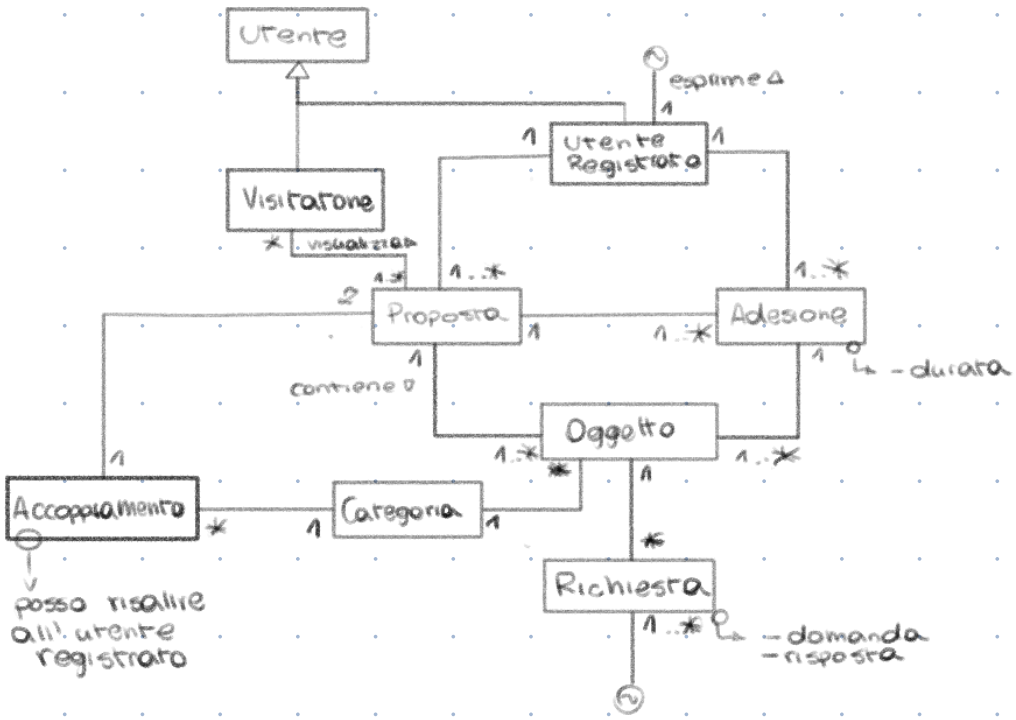
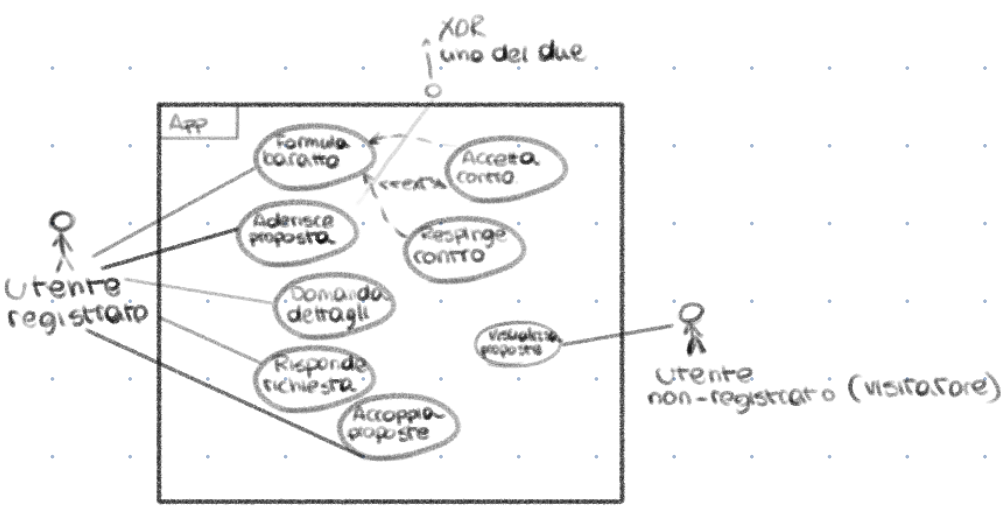
Domain model



Use case



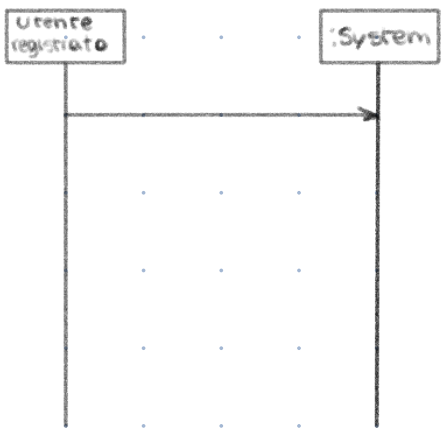
UC: Aderisce proposta
Attore: Utente registrato
Pre: Utente registrato alla piattaforma
Utente abbia visualizzato proposte filtrando per categoria

1. Utente seleziona oggetti per controparttita
2. Sistema richiede inserimento dati
3. Utente inserisce dati
4. Utente inserisce categoria
5. Sistema visualizza corretto inserimento dei dati

Post: Sistema allega alla proposta del proponente l'adesione appena avvenuta

Sequence diagram

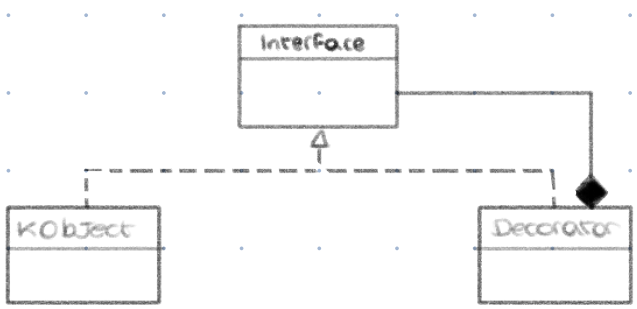
Stessa cosa del main path!



state machine diagram

1. Proposto
2. Accettato
3. Respinto
4. Visualizzato
5. Elaborato
6. Attivo
7. Scaduto
8. Attesa
9. Osservato
10. Associato

② Decorator é un pattern strutturale del catalogo GoF, attuato affinché l'architettura interna del sistema sia manipolabile e adattiva dinnanzi a modifiche.
Il pattern rappresenta un chiaro esempio di come il solo utilizzo di inheritance per modellare certe tematiche non sia sufficiente.
L'implementazione secondo una gerarchia di sottotipi, qualora il sistema software risulti essere caratterizzato da un numero crescente di combinazioni comportamentali, potrebbe rivelarsi non idonea; rispetto a quanto detto, affinché sia favorita condivisione comportamentale, la struttura interna diverebbe complicata da gestire, soprattutto qualora si presenti la necessità di sviluppare ulteriori funzionalità, le quali, pur di non violare principi SOLID, saranno incapsulate all'interno di ulteriori entità software, causando nuovamente un accrescamento delle classi da gestire.
Sicuramente se adeguato un meccanismo simile, potrebbe causare la violazione di Liskov substitution principle, data la mancata soddisfazione della compatibilità comportamentale proprio per questa ragione il polimorfismo non può essere adoperato per la sovrascrittura totale da metodi ereditati, e di Open-closed principle, data che molto probabilmente, metodi già sviluppati siano modificati per necessità di condivisione comportamentale, portando inevitabilmente alla violazione del principio citato.
Decorator ammette un meccanismo capace di evitare avversità poiché permette di arricchire il comportamento di un oggetto durante run-time, pertanto favorendo una concreta suddivisione.
L'intento consiste nella realizzazione di gerarchia di classi che possano variare le funzionalità caratterizzanti di un oggetto analizzato, ovviando ad un'implementazione forzata dall'ereditarietà.
A livello implementativo è posta un'interfaccia tra classe concreta e la gerarchia/e di decoratori, in modo tale che sia soddisfatto il DIP, ma soprattutto affinché possano modellare gli stessi comportamenti.
In tal senso un decoratore opera come se fosse un contenitore, ossia arricchisce il comportamento mantenendo al suo interno un riferimento dell'istanza della classe. In questo modo classi finali manterranno al loro interno un riferimento alla propria superclasse, pertanto contenendo il livello superiore fino all'entità che contiene la referenza dell'oggetto concreto.



③ Creator é un software pattern, attuato affinché sia possibile responsabilizzare entità della realizzazione di oggetti di classi, usufruendo in questo modo dei metodi incapsulati al loro interno.
Il pattern è adeguato se soddisfatte almeno una delle seguenti condizioni, dove adeguando una semplificazione: la classe B è responsabile della creazione della classe A se:

- B contiene istanze di A
- B aggrega oggetti di A
- B utilizza fortemente metodi di A
- B possiede tutti i parametri di input del costruttore di A

Nonostante é bene prestare molta attenzione a quale classe sia affidata la responsabilità, poiché se attuato a high level classes potrebbe provocare la violazione del dipendancy inversion principle, poiché potrebbe creare istanze di LLC usufruendo dei metodi implementati, causando un erronea gestione di dipendenze e design smells. Rispetto quanto detto occorre sovrapporre tra le due entità un' astrazione, isolandole da modifiche reciproche.