DOCUMENTAZIONE

Applicazione Noleggi

Indice

1 Ingegneria dei Requisiti

1.1 Obiettivo

Si vuole realizzare un sistema per la gestione di un autonoleggio. L'applicazione dovrà servire gli utenti che desiderano noleggiare un'auto. Gli utenti possono essere sia utenti registrati che utenti non registrati.

L'utente non registrato deve poter registrarsi al sistema inserendo alcuni dati personali. Inoltre, sarà permesso a coloro che non sono registrati di visualizzare il parco macchine disponibile e simulare una prenotazione per vedere la disponibilità e il prezzo del noleggio senza poter finalizzare la prenotazione.

Durante la fase di registrazione verrà offerta la possibilità di scegliere o meno di partecipare al programma fedeltà che includerà diversi vantaggi che verranno spiegati in seguito. (aggiungere una condizione per poter partecipare al programma fedeltà, esempio pagando una quota mensile o annuale).

1.2 Target

Questa applicazione è sviluppata per poter essere utilizzata da tutti. Infatti l'interfaccia di utilizzo è studiata per essere semplice ed intuitiva. Le diverse modalità e tipologie di noleggio vengono visualizzate nella pagina principale. Nessuna conoscenza o requisito pregressa vengono richiesti.

1.3 Servizi offerti all'utente

L'utente registrato può effettuare un noleggio scegliendo tra le tipologie che vengono offerte dal sistema.

Vengono offerti tre tipi di noleggi:

- car sharing (noleggio giornaliero o weekend)
- noleggio nel breve periodo (max 3 mesi)
- noleggio nel lungo periodo (max 1 anno, solo gli utenti aderenti al programma fedeltà possono sceglierlo)

Una volta selezionato il tipo di noleggio, l'utente dovrà scegliere il tipo di auto da noleggiare tra:

- utilitaria
- business
- luxury

All'utente sarà inoltre consentito visualizzare lo storico delle prenotazione effettuate, consentendo di modificare o annullare una prenotazione, questo, è possibile solo in alcuni casi.

Il programma fedeltà permette agli utenti di raccogliere punti in base alla tipologia di noleggio che viene effettuato. Vengono assegnati rispettivamente 10 punti per il noleggio Car Sharing, 25 noleggio nel breve periodo e 50 noleggio nel lungo periodo. Inoltre, l'utente accumula punti in base ai chilometri percorsi e alle condizioni con cui viene riconsegnata l'auto. Il prezzo che l'utente corrisponde per il noleggio viene calcolato in punti fedeltà. I punti permettono di ottenere sconti e agevolazioni per nuovi noleggi.

1.4 Principali funzioni dell'applicazione

- Visualizzare il parco auto

- Simulare la creazione di un noleggio (utente non registrato)
- Eseguire registrazione e/o login
- Inizializzare un noleggio

1.5 Requisiti funzionali e non funzionali

I requisiti sottoquotati sono stati schematizzati secondo la tabella **MoSCoW.**

1- **MUST HAVE:** possibilità di effettuare login e registrazione, inizializzare un nuovo noleggio, selezionare il tipo, il periodo di ogni noleggio, selezionare il tipo dell'auto. Associare ciascun noleggio a ciascun utente.
2- **SHOULD HAVE:** possibilità per ogni utente di accedere allo storico dei noleggi.
3- **COULD HAVE:** gli utenti non registrati possono simulare un noleggio per verificarne disponibilità e prezzo.
4- **WON'T HAVE:** possibilità di inizializzare più noleggi contemporaneamente per ciascun utente

2 Ciclo di vita del software

2.1 AGILE – Extreme programming

Si è scelto di procedere seguendo uno dei metodi Agile: Extreme Programming. Questo promuove lo sviluppo iterativo ed incrementale organizzato in brevi cicli di sviluppo, detti interazioni.

Lo sviluppo è accompagnato dalla stesura di un piano di lavoro che viene continuamente aggiornato a intervalli brevi e regolari.

Dal punto di vista dell'implementazione, Extreme Programming promuove la scrittura di soluzioni semplici che possono essere adattate e migliorate in un secondo momento attraverso refactoring o scrittura di componenti aggiuntive. Nell'Extreme Programming lo sviluppo prevede che tutti lavorino su tutto alternandosi durante alcune fasi, favorendo così il rendimento e mantenendo alto il livello di concentrazione.

2.2 I principi di XP

Cinque principi:

- **Feedback rapido**: ogni volta che vengono aggiornate parti del sistema queste vengono controllate e testate dagli sviluppatori, in quanto si pensa di consegnare sempre una versione funzionante al cliente.
- **Semplicità**: si vuole evitare di crearsi ulteriore complessità, per cui si cerca di mantenere un design semplice e di aggiungere funzionalità solo quando necessario ed il codice sarà oggetto di refactoring.
- **Cambiamento incrementale:** non si prendono decisioni improvvise, i cambiamenti vengono effettuati in maniera graduale.
- **Abbracciare il cambiamento:** non si fanno pianificazioni a lungo termine, i problemi più gravi vengono risolti il prima possibile.
- **Lavoro di qualità:** si cerca di offrire sempre un prodotto qualitativo.

3 Organizzazione del team

3.1 Struttura del team

Il team è composto da tre persone:

- **Alexander Rubino** *(MAT. 1064467)*
- **Ibrahima Sarr** *(MAT. 1046446)*
- **Beniamino Infante** *(MAT.1063452)*

La struttura del team è semplice: tutti e tre i membri sono al centro del progetto e la coordinazione avviene mediante diretta supervisione.

3.2 Suddivisione del lavoro

Il lavoro viene suddiviso in maniera proporzionale, assegnando i vari compiti in base alle proprie conoscenze e competenze. Si procederà con la realizzazione dei diagrammi UML, suddivisi equamente tra i membri del team per delineare le linee guida per la stesura del codice. Si sfrutterà una caratteristica di XP: il pair programming. L'obiettivo è quello di lavorare in coppia (quando possibile) in modo che quando un membro del team lavora sul codice gli altri due lavorano sui test in modo da verificare e confermare che il lavoro stia andando bene.

3.3 Comunicazione all'interno del team

La comunicazione è alla base del progetto. Durante il periodo di sviluppo (prefissato in base agli impegni di ciascuno dei membri del team) ci si incontra almeno una volta a settimana con delle riunioni di "allineamento" sul lavoro svolto nei giorni precedenti e sui prossimi step ed obiettivi da raggiungere.

3.4 Configuration Management

Github è il tool che ci ha aiutato per la gestione del progetto. Questo ci ha offerto la possibilità di lavorare su diversi branch locali in modo da poter lavorare sul codice e testare i vari cambiamenti su di esso senza "sporcare" il lavoro funzionante svolto in precedenza.

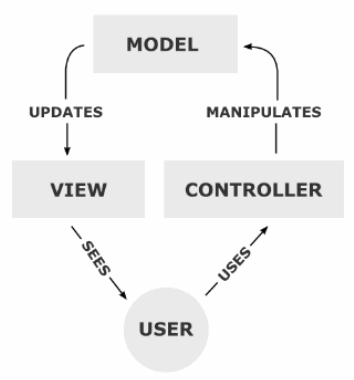
Quando una modifica era pronta per poter essere rilasciata sul main branch solitamente abbiamo effettuato una pull request, di conseguenza accettata o rifiutata dagli altri membri del team. Sono state aperte anche diverse issues.

4 Architettura del software

Per l'architettura del nostro software si è scelto di adottare un approccio di tipo **Model View Controller (MVC).** Questo modello permette di gestire direttamente i dati, la logica e le regole dell'applicazione.

MVC è caratterizzato da tre componenti:

- **MODEL*:*** fornisce i metodi per accedere ai dati utili dell'applicazione.
- **VIEW*:*** visualizza i dati contenuti nel Model e si occupa dell'interazione con utenti e agenti.
- **CONTROLLER*:*** riceve i comandi dell'utente, generalmente dall'interfaccia grafica (View) e li attua modificando lo stato delle altre due componenti.



5 Qualità del software

Il sistema deve essere conforme ad una serie di attributi previsti dallo standard ISO 9126.

Questo prevede i seguenti attributi:

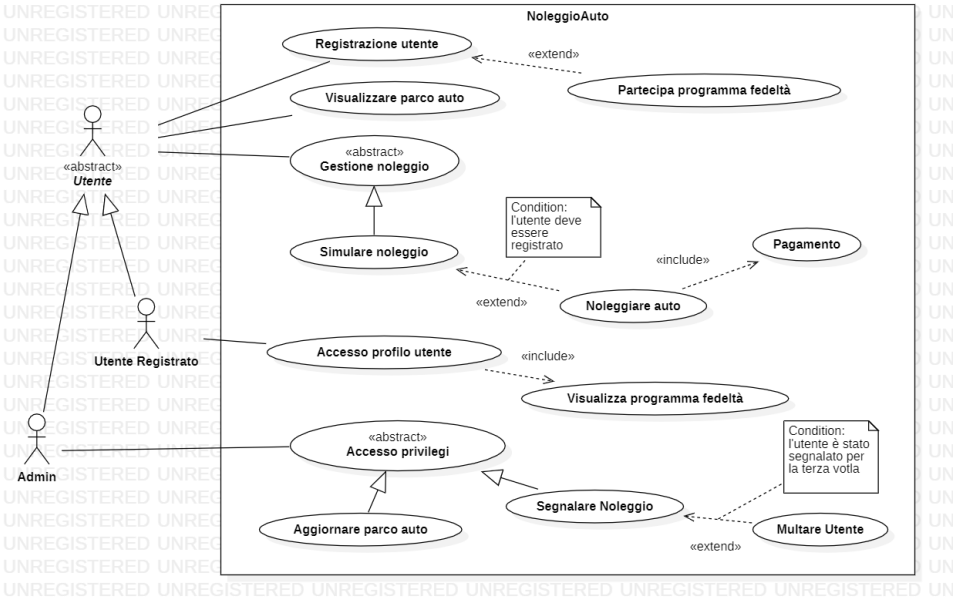
- **Funzionalità:** il sistema deve soddisfare i requisiti prefissati.
- **Affidabilità:** il sistema deve avere una buona tolleranza agli errori e quindi non presentare guasti o malfunzionamenti tali da compromettere l'esperienza all'utente finale.
- **Usabilità:** il sistema deve essere facilmente utilizzabile dall'utente finale.
- **Efficienza**: il sistema non deve usare risorse eccessive. Deve essere ottimizzato per l'uso immediato.
- **Manutenibilità:** il sistema deve essere in grado di evitare effetti non desiderati a seguito di aggiornamenti e modifiche al codice.

6 Modellazione – UML

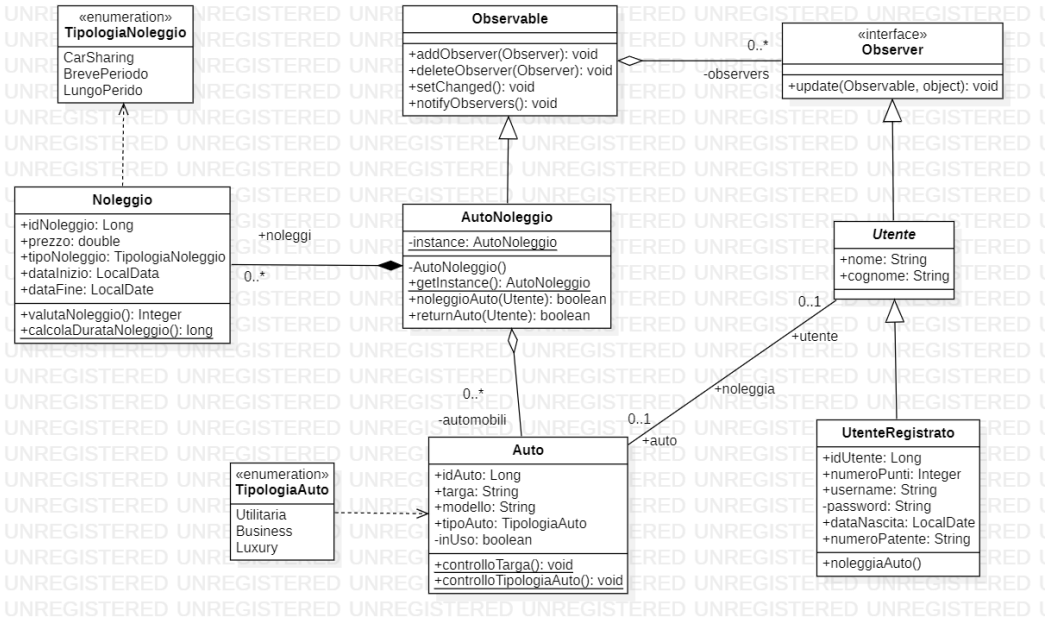
6.1 Tool utilizzato

Tutti i diagrammi UML sono stati generati attraverso il tool StarUML e sono disponibili nel formato *.mdj* nel repository all'interno della cartella "Diagrammi UML". Qui di seguito verranno riportate le illustrazioni dei vari diagrammi UML.

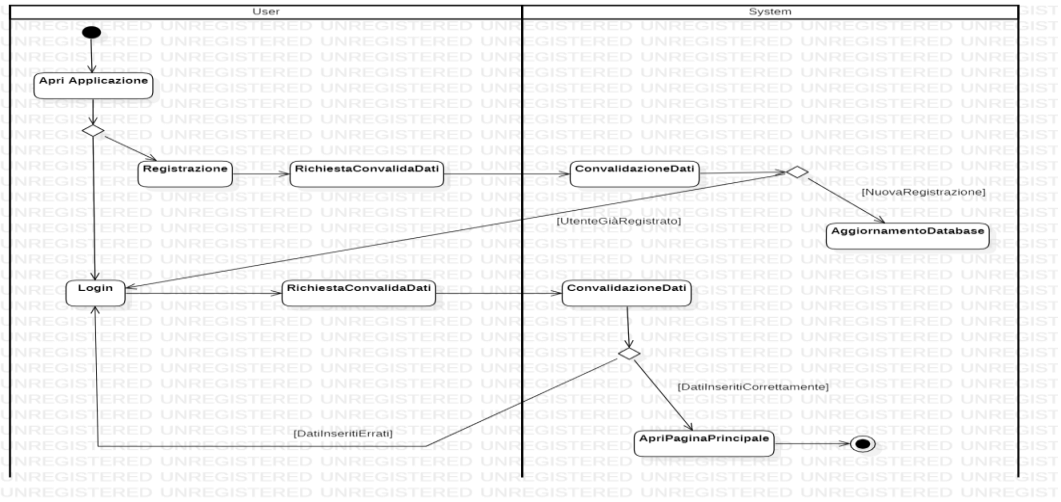
6.2 Use Case Diagram



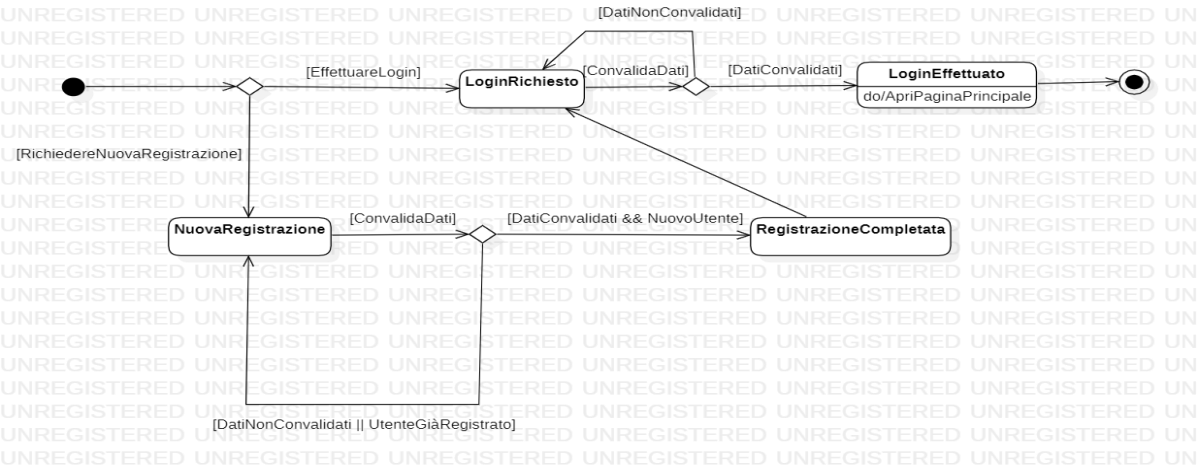
6.3 Class Diagram



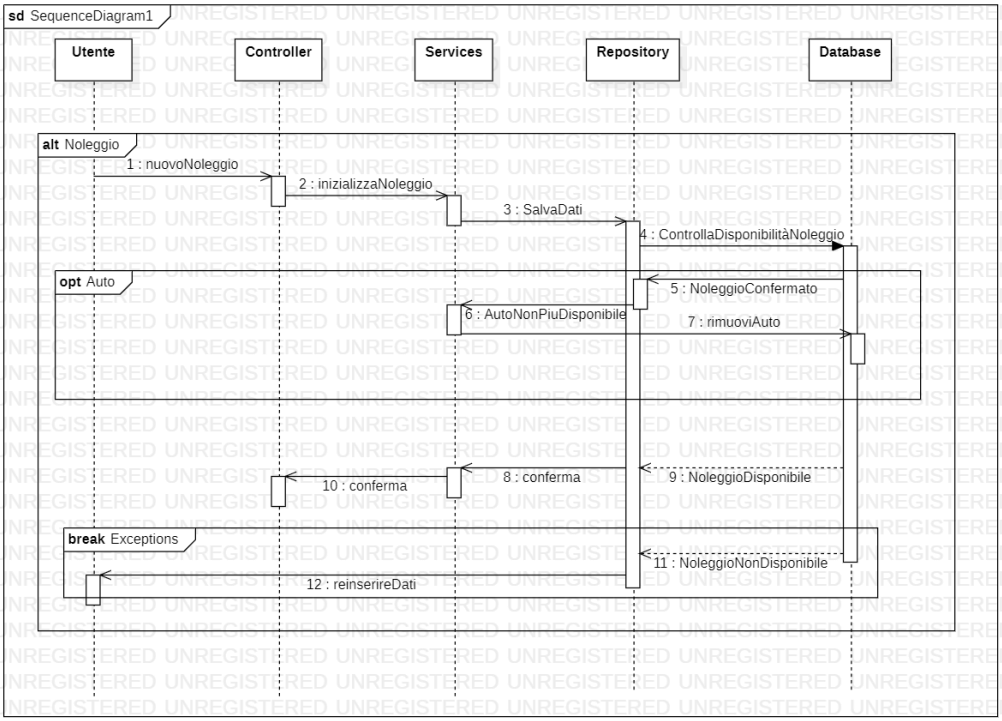
6.4 Activity Diagram



6.5 State Machine Diagram



6.6 Sequence Diagram



7 Design del software

7.1 Design pattern utilizzati

7.2 Dati generali sul codice

La figura riportata qui sotto ci indica i dati relativi al numero di classi e numero di packages, sia interni che esterni presenti all'interno del progetto.

Total lines of code: 812

Number of classes: 44

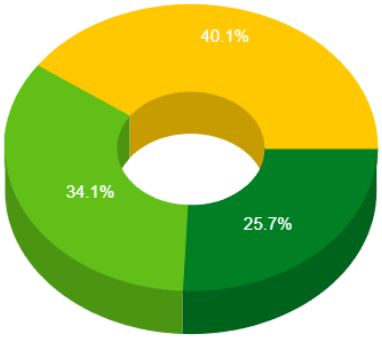
Number of packages: 9

Number of external packages: 47

Number of external classes: 195

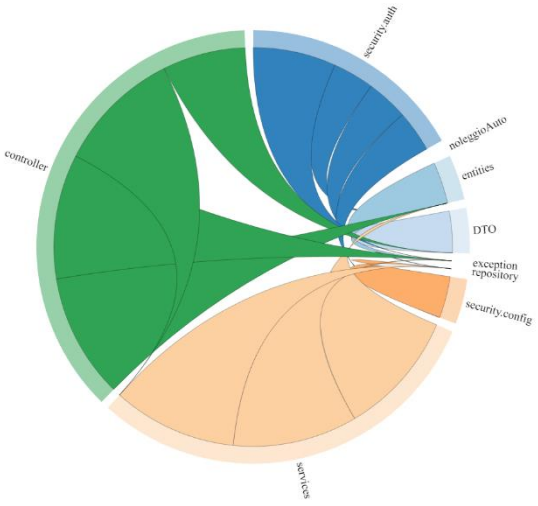
Number of problematic classes: 0

Number of highly problematic classes: 0



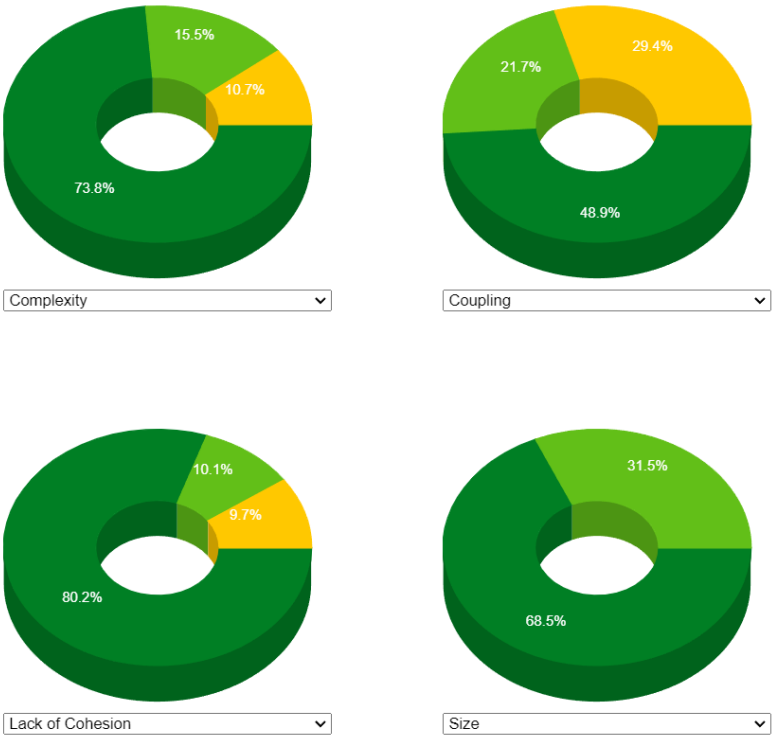
7.3 Dati relativi ai packages

La figura riportata di seguito riporta invece le dipendenze dei vari packages presenti all'interno del progetto.



7.4 Complessità, accoppiamento e coesione

Si è cercato di sviluppare il progetto cercando di attribuire ad esso delle caratteristiche importanti. Di fatto l'obiettivo era quello di mantenere un alto livello di coesione ed un basso livello di accoppiamento. Come riportato dai dati seguenti l'80% delle classi presentano un valore basso di "mancaza di coesione". Il livello di complessità invece risulta essere basso per il 70% delle classi.



8 Testing

L' Extreme Programming viene riconosciuto come un Test Driven Developement ed appunto per questo il nostro è stato un approccio orientato al testing. Ogni modifica che è stata effettuata veniva testata manualmente insieme all'utilizzo di alcuni casi di test creati con JUnit.