DOCUMENTAZIONE

Applicazione Noleggi

Indice

1 Ingegneria dei Requisiti 2

1.1 Obiettivo 2

1.2 Target 2

1.3 Servizi offerti all'utente 2

1.4 Servizi offerti all'admin 3

1.5 Principali funzioni dell'applicazione 3

1.6 Requisiti funzionali e non funzionali 3

1.7 Vincoli 4

2 Ciclo di vita del software 4

2.1 AGILE – Extreme programming 4

2.2 I principi di XP 4

3 Organizzazione del team 4

3.1 Struttura del team 4

3.2 Suddivisione del lavoro 5

3.3 Comunicazione all'interno del team 5

3.4 Configuration Management 5

4 Architettura del software 5

5 Qualità del software 6

6 Modellazione – UML 6

6.1 Tool utilizzato 6

6.2 Use Case Diagram 7

6.2.1 Scenario 1 7

6.2.2 Scenario 2 8

6.3 Class Diagram 8

6.4 Activity Diagram 9

6.5 State Machine Diagram 9

6.6 Sequence Diagram 10

7 Design del software 10

7.1 Design pattern utilizzati 10

7.2 Dati generali sul codice 11

7.3 Dati relativi ai packages 11

7.4 Complessità, accoppiamento e coesione 12

8 Testing 12

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 e gli admin che hanno accesso al sistema per gestire le richieste di noleggio. 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.

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 proposti tre tipi di noleggi:

- car sharing (noleggio giornaliero o weekend)
- noleggio nel breve periodo (max 3 mesi)
- noleggio nel lungo periodo (max 1 anno)

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

- utilitaria
- business
- luxury

Se il sistema non permette la possibilità di noleggiare l'auto richiesta dall'utente, allora sarà inserito in una lista prioritaria. Nel momento in cui l'auto diventa disponibile ed è possibile effettuare la prenotazione, l'utente verrà notificato dal sistema.

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 Servizi offerti all'admin

L'admin oltre ad avere la possibilità di usufruire di tutti i servizi presenti nell'applicazione, ha accesso ad alcune funzionalità privilegiate:

- Aggiornare parco auto (aggiungere o eliminare un'auto presente nel sistema)
- Segnalare attività improprie del noleggiante (consegna auto oltre orario previsto o danni sulla vettura) e attivare pratica per addebiti dopo tre segnalazioni.

1.5 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

1.6 Requisiti funzionali e non funzionali

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

1- **MUST HAVE:** sono requisiti assolutamente necessari.
2- **SHOULD HAVE:** sono requisiti importanti, ma non assolutamente necessari per un sistema utilizzabile.
3- **COULD HAVE:** sono requisiti che vengono implementati solo se il tempo lo consente.
4- **WON'T HAVE:** sono requisiti non richiesti che rimarranno per la prossima iterazione.

	MUST HAVE	SHOULD HAVE	COULD HAVE	WON'T HAVE
Requisiti Funzionali				
Login e Registrazione	L'utente deve poter effettuare il login e la registrazione			
Noleggio Auto	L'utente deve poter visualizzare le auto disponibili e noleggiarle	L'utente deve poter scegliere il tipo di auto e la data di inizio e fine noleggio	L'utente deve essere aggiunto alla liste di persone che verranno notificate quando l'auto richiesta diventa disponibile	L'utente deve poter scegliere gli optional per l'auto (es. navigatore)
Gestione Prenotazioni	L'utente deve poter visualizzare le prenotazioni effettuate		L'utente deve poter modificare o annullare una prenotazione	
Gestione Pagamenti			L'utente deve poter inserire i dati di pagamento in modo sicuro	L'utente deve poter visualizzare la cronologia dei pagamenti
Requisiti Non Funzionali				
Sicurezza			I dati degli utenti e dei pagamenti devono essere protetti	
Prestazioni	L'applicazione deve essere veloce e reattiva			
Usabilità	L'interfaccia utente deve essere intuitiva e user-friendly			
Scalabilità	L'applicazione deve poter gestire un grande numero di utenti e di prenotazioni contemporaneamente			
Portabilità			L'applicazione deve essere disponibile su qualsiasi piattaforma	

1.7 Vincoli

Ci sono dei vincoli legati all'uso delle funzionalità:

- L'utente senza aver effettuato la registrazione non può effettuare il login.
- Una e-mail può corrispondere ad un solo account registrato.
- Se l'utente non restituisce l'automobile noleggiata in condizioni ottimali oppure la riconsegna oltre il limite della data di fine noleggio riceverà una segnalazione. Alla terza segnalazione riceverà una multa, la quale se non verrà pagata dall'utente entro una settimana comporterà la sospensione dell'account.

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.

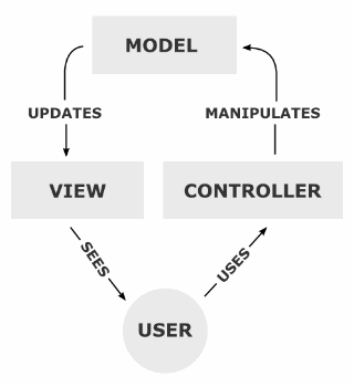
Quando una modifica era pronta per poter essere rilasciata sul main branch solitamente abbiamo effettuato una pull request, di conseguenza accettata o rifiutata dagli altri membri del team. Sono state aperte anche diverse issues.

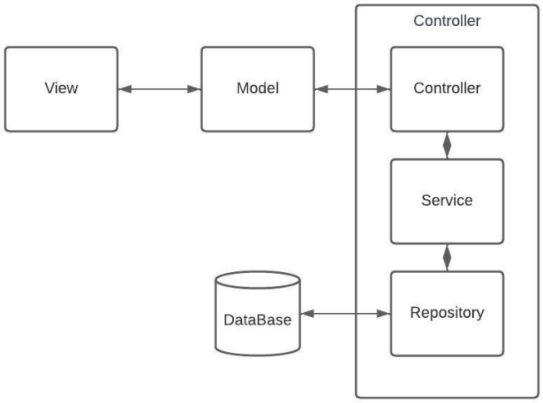
4 Architettura del software

Essendo un'applicazione web abbiamo scelto di basarci su un'architettura di tipo client-server. Inoltre, l'applicazione deve presentare delle viste all'utente sul lato client, le quali mostrano dati precedentemente modellati ed estratti da un database lato server. Per questo motivo abbiamo scelto di adottare il pattern architetturale **Model View Controller (MVC),** dove il controller è a sua volta sviluppato su un'architettura a 3 layers. Il MVC è caratterizzato da tre componenti:

- **MODEL*:*** Il componente Model permette la comunicazione tra il Controller e la View e fornisce i metodi per accedere ai dati utili dell'applicazione.
- **VIEW*:*** Il componente View è l'interfaccia utente che viene aggiornata in base alle dinamiche del controller e si occupa dell'interazione con gli utenti.
- **CONTROLLER*:*** riceve i comandi dell'utente, generalmente dall'interfaccia grafica (View) e li attua modificando lo stato delle altre due componenti. Il componente Controller è il cuore dell'applicazione. È qui che viene sviluppata la business logic. Questo componente è a sua volta suddiviso in 3 layers. Il primo che indichiamo come "Repository", si collega al Database e ci permette di salvare e di cercare i dati di cui abbiamo bisogno. Il secondo layer, o "Service", sviluppa la logica dell'applicazione e ci permette di implementare la sicurezza. Il terzo e ultimo layer è il Controller vero e proprio che tramite il Model riesce a comunicare e gestire le richieste che provengono dall'interazione fra utente e View.

Di seguito sono state riportate due figura che rappresentato una vista statica dell'architettura utilizzata.





5 Qualità del software

Nel glossario IEEE della terminologia dell'ingegneria del software, la qualità è definita come "il grado in cui un sistema, componente o processo soddisfa le esigenze o le aspettative del cliente o dell'utente".

Per il nostro progetto abbiamo scelto che il sistema sia conforme allo standard ISO 9126.

L'ISO 9126 è una norma internazionale che definisce un modello di qualità del software. Il modello si basa su sei caratteristiche di qualità, suddivise in sotto-caratteristiche, che sono le seguenti:

- **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.

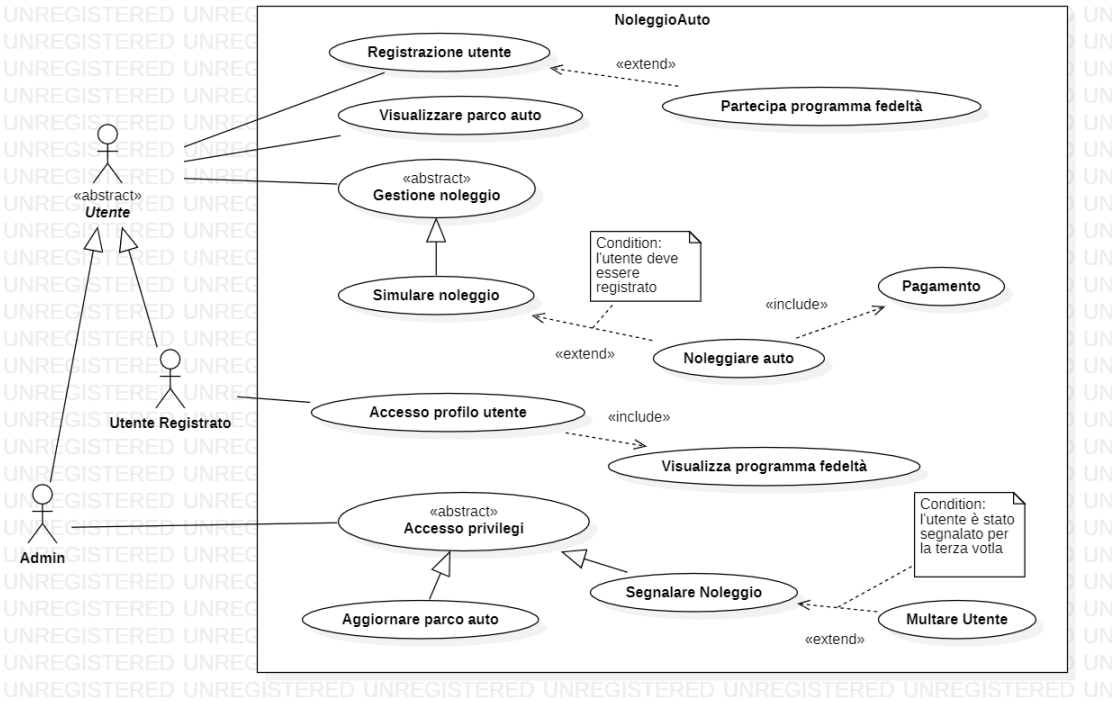
Le caratteristiche e le sotto-caratteristiche dell'ISO, insieme ad un'ampia serie di misure, costituiscono il modello di qualità esterno e interno della normativa. La qualità interna si riferisce al prodotto stesso mentre quella esterna si riferisce alla qualità quando il software viene eseguito.

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



Nella sezione scenari vengono descritti alcune delle possibili interazioni tra gli attori e il sistema.

6.2.1 Scenario 1

REGISTRAZIONE UTENTE

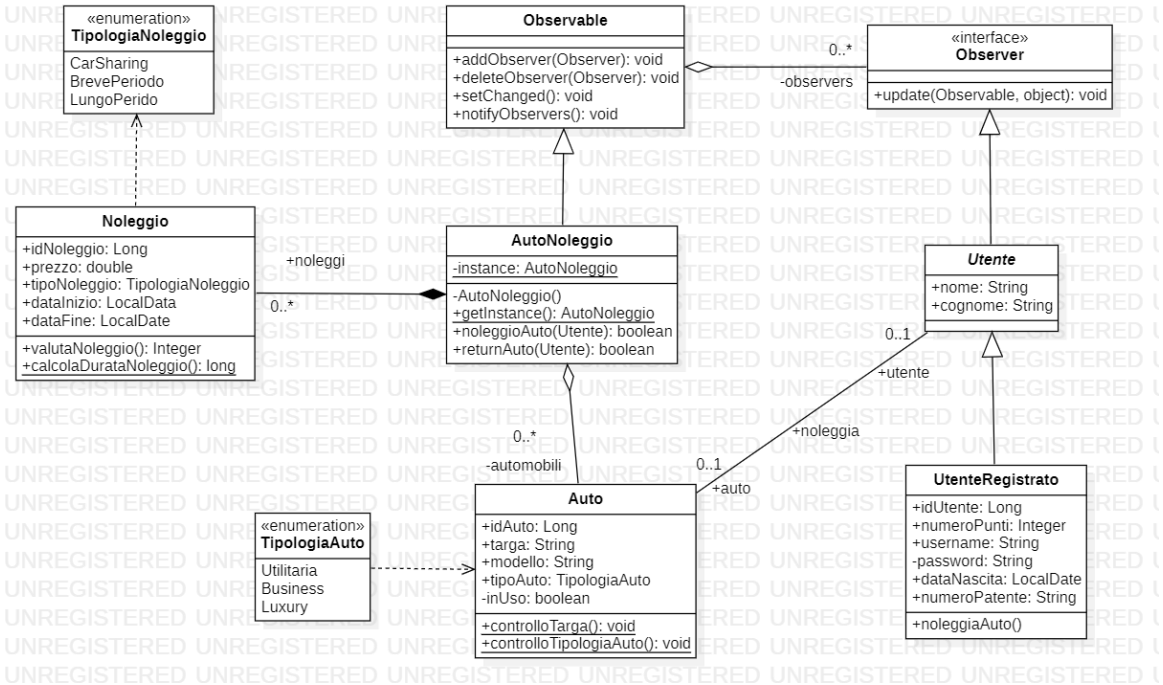
Attore	Utente
Precondizioni	Utente non deve essere registrato
Eventi	1) Utente seleziona registrazione 2) Il sistema mostra una pagina dove poter compilare un form nel quale vengono richieste diverse informazioni: nome, cognome, data di nascita, numero patente, e-mail e una password. 3) Utente inserisce le informazioni richieste dal sistema e clicca sul bottone completa registrazione 4) Infine, le informazioni vengono invitate e salvate nel Data Base.
Post condizioni	L'utente viene registrato
Eccezioni	1) Il sistema non completa la registrazione nel caso in cui l'utente utilizza una e-mail già presente.

6.2.2 Scenario 2

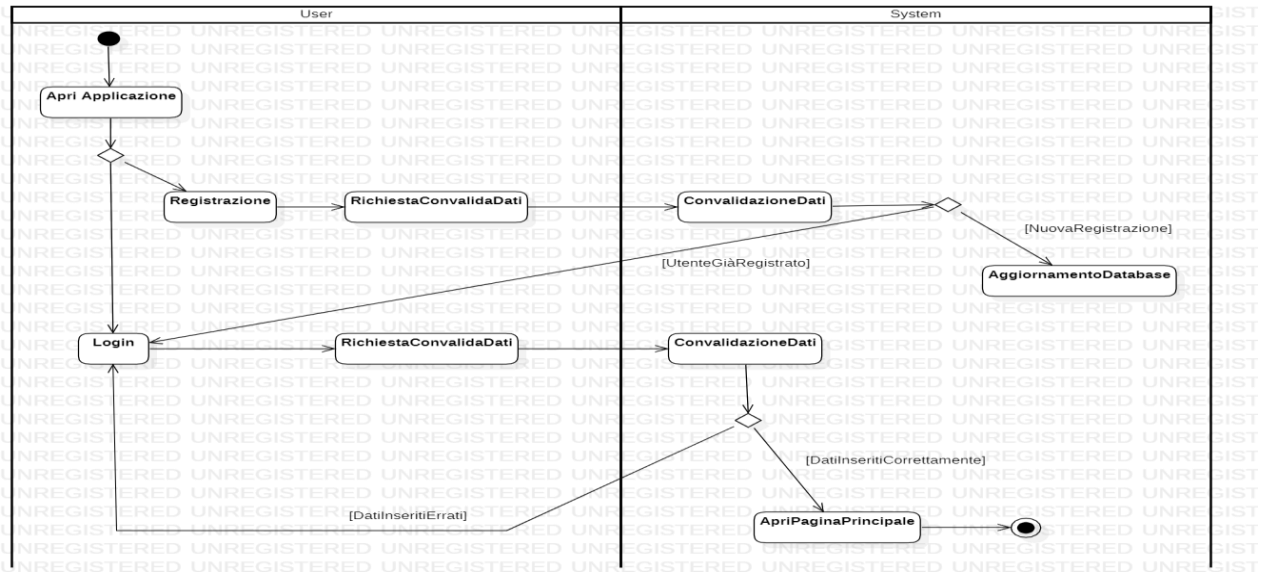
NOLEGGIO AUTO

Attore	Utente
Precondizioni	Utente deve essere registrato
Eventi	1) Utente seleziona la tipologia di noleggio che vuole effettuare tra quelli offerti dal sistema. 2) In seguito, ha la possibilità di selezionare la tipologia di automobile che desidera noleggiare. 3) Fatto ciò, esegue il pagamento con la possibilità di utilizzare i punti fedeltà per uno sconto sul prezzo totale del noleggio. 4) Il noleggio viene preso in carica dal sistema che provvede alla conferma o annullamento. 5) In caso di conferma i dati del noleggio vengono inviati e salvati nel DB.
Post condizioni	L'utente ha un'auto in noleggio.
Eccezioni	1) Il sistema non completa il noleggio nel caso in cui l'auto non è disponibile 2) Il sistema non completa il noleggio nel caso in cui il pagamento non viene accettato.

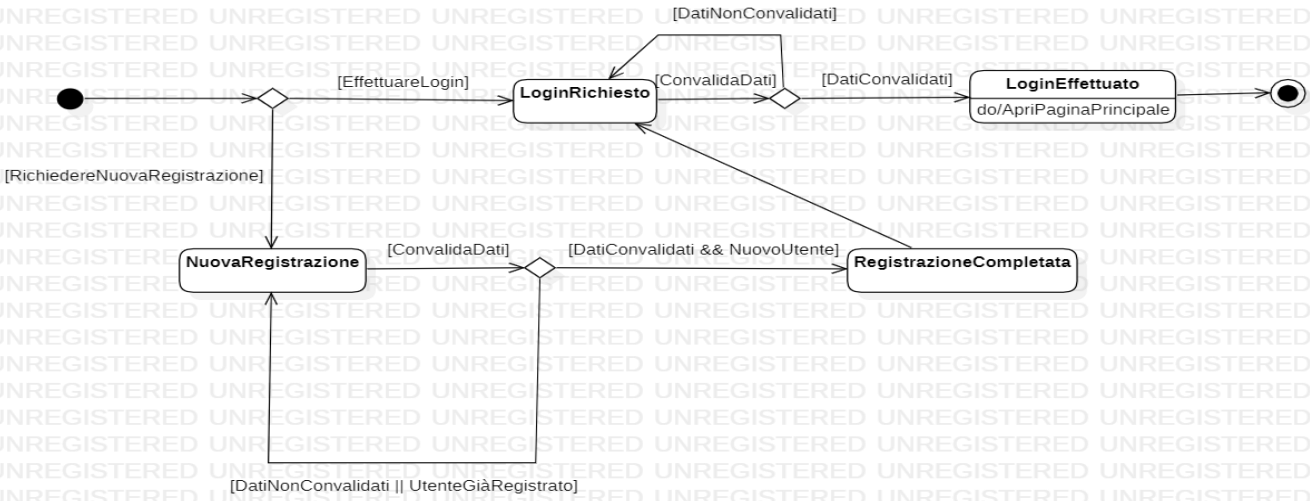
6.3 Class Diagram



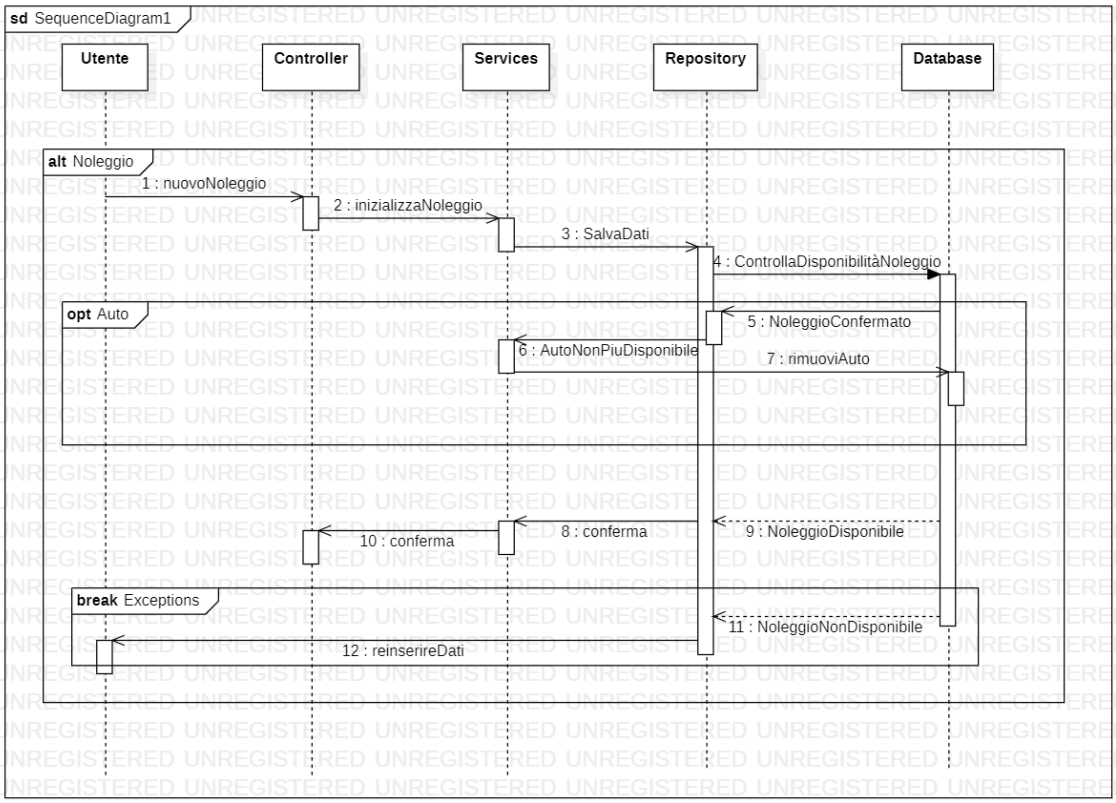
6.4 Activity Diagram



6.5 State Machine Diagram



6.6 Sequence Diagram



7 Design del software

7.1 Design pattern utilizzati

Nello sviluppo di questa applicazione si è scelto di utilizzare due design pattern:

- Singleton: è un pattern creazionale che ha lo scopo di assicurarsi che una classe abbia una sola istanza e provvedere un punto di accesso globale a questa istanza. Nel nostro caso la singola istanza che può essere creata è l'autonoleggio.
- Observer: è un pattern comportamentale che permette di definire una dipendenza uno a molti tra oggetti in modo che quando un oggetto cambia stato, vengono notificati tutti i suoi osservatori. Per la nostra applicazione l'utilizzo dell'observer pattern permette al sistema di inviare notifiche agli utenti riguardo alla disponibilità delle auto nel momento in cui un'auto richiesta per il noleggio non è disponibile. L'Autonoleggio è il subject concreto (Observable) mentre l'utente è l'Observer.

Tuttavia, la rappresentazione dei design pattern è stata svolta solo per la parte riguardante UML, poiché per ragioni di compatibilità con il framework SpringMVC impiegato non è stato possibile sviluppare manualmente i patterns. Di fatto SpringMVC implementa a suo modo il Singleton Pattern attraverso l'utilizzo di alcune annotazioni (es. '@Autowired').

7.2 Dati generali sul codice

Tutti i grafici e metriche relative al codice sono state generati utilizzando **CodeMR Static Code Analyser**.
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

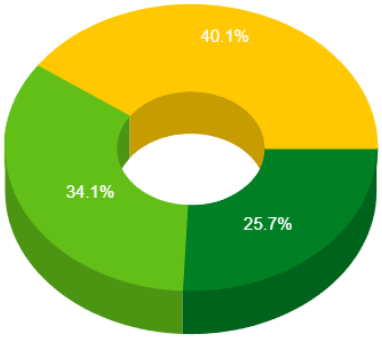
Number of packages: 9

Number of external packages: 47

Number of external classes: 195

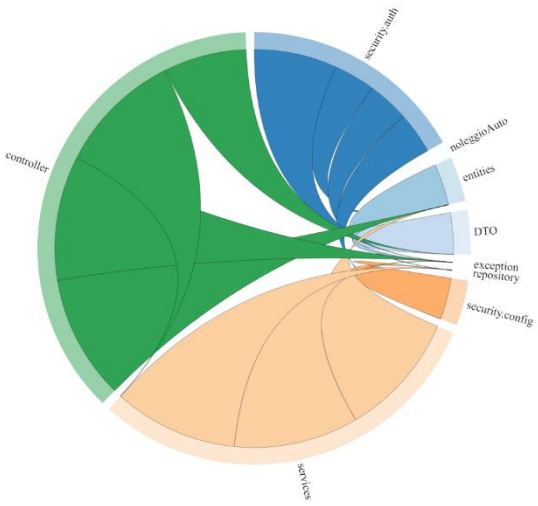
Number of problematic classes: 0

Number of highly problematic classes: 0



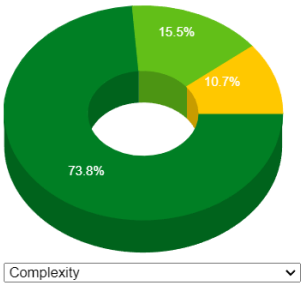
7.3 Dati relativi ai packages

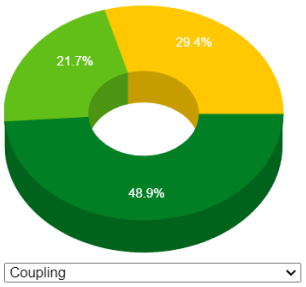
La figura riportata di seguito riporta invece le dipendenze dei vari packages presenti all'interno del progetto.

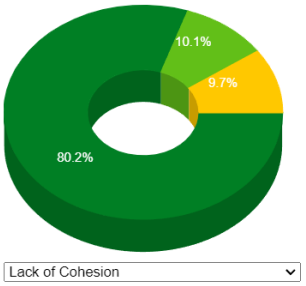


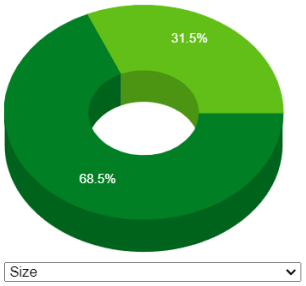
7.4 Complessità, accoppiamento e coesione

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. Per questo motivo abbiamo dato enfasi ai test generandoni diversi.

I test sono stati effettuati cercando di coprire buona parte del codice. Inizialmente sono stati effettuati i test sulle varie entità (classi Auto, Noleggio, Utente) accertandoci che le funzioni basilari funzionassero al meglio.

La percentuale di coverage raggiunta su tutto il codice è del 71,6%.

Di seguito sono stati realizzati i test sulle altre componenti del software. In generale abbiamo voluto testare il completo funzionamento del context e delle richieste http.

Infine sono state testate le funzionalità delle altre componenti per verificare che i metodi più importanti ai fini del funzionamento dell'applicazione fossero stati correttamente implementati.

Tutti i test sono stati realizzati sfruttando JUnit 5 a cui abbiamo affiancato il framework Mockito, sfruttato per simulare il comportamento di alcune componenti mediante l'utilizzo dei Mock.