

# PROGETTO INGEGNERIA DEL SOFTWARE

## *Applicazione Noleggi Auto*







# L' OBIETTIVO

La nostra idea è quella di realizzare un applicativo che gestisca i noleggi di auto.

Proponiamo 3 tipi di noleggi:

- **Car Sharing** (noleggio giornaliero o weekend)
- **Noleggio di breve periodo** (noleggio di massimo 3 mesi)
- **Noleggio di lungo periodo** (noleggio di massimo un anno)

Viene inoltre offerta la possibilità di scegliere tra 3 tipi di auto:

- **Utilitarie**
- **Business**
- **Luxury**

Al termine di ogni noleggio verranno **accreditati agli utenti dei punti** calcolati sulla base del tipo di noleggio scelto e dei km percorsi.



# I REQUISITI DEL SOFTWARE

I requisiti del software sono stati schematizzati secondo la **tabella MoSCoW**:

REQUISITI FUNZIONALI	MUST HAVE	SHOULD HAVE	COULD HAVE	WON'T HAVE
LOGIN E REGISTRAZIONE	L'utente deve poter effettuare il login e la registrazione			
NOLEGGIO AUTO	L'utente deve poter visualizzare le prenotazioni	L'utente deve poter scegliere il tipo di auto e l'intervallo del noleggio	L'utente può essere notificato sulla disponibilità di un'auto	L'utente non può scegliere gli optional di un'auto
GESTIONE PRENOTAZIONE	L'utente deve visualizzare le prenotazioni		L'utente può modificare o annullare la prenotazione	
GESTIONE PAGAMENTI			Maggior sicurezza sui dati di pagamento	Non potrà visualizzare la cronologia dei pagamenti
REQUISITI NON FUNZIONALI	MUST HAVE	SHOULD HAVE	COULD HAVE	WON'T HAVE
SICUREZZA			I dati degli utenti devono essere protetti	
PRESTAZIONI	L'app deve essere veloce e reattiva			
USABILITÀ	L'interfaccia deve essere intuitiva			
SCALABILITÀ	L'app deve gestire un grande numero di richieste			

# IL CICLO DI VITA DEL SOFTWARE

Il metodo che abbiamo deciso di utilizzare in fase di progettazione è un **metodo Agile**. In particolare abbiamo deciso di usare l'**Extreme Programming (XP)** in quanto si basa su delle best practices come: design **semplice**, **refactoring** e **programmazione in coppia**.

XP è caratterizzato da **5 principi**:



## FEEDBACK RAPIDO

Controllo e testing continui dei cambiamenti a parti di sistema prima di rilasciare al cliente



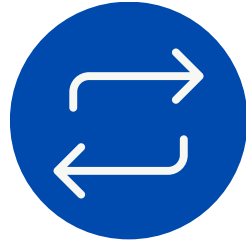
## SEMPlicità'

Si cerca sempre di tenere un design semplice, aggiungendo funzionalità solo quando richiesto ed il codice è spesso oggetto di refactoring



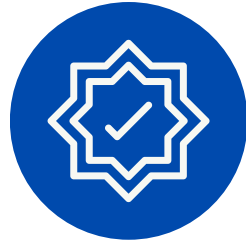
## CAMBIAMENTO INCREMENTALE

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

# ***IL PARADIGMA DI PROGRAMMAZIONE E MODELLAZIONE***

Si è scelto di utilizzare come paradigma di programmazione un paradigma **orientato agli oggetti**. Di fatti abbiamo dato priorità alla produzione di modelli, quali il **class diagram**, da cui poi abbiamo ricavato lo scheletro del codice. Come linguaggio di programmazione abbiamo scelto di usare **JAVA**, con **SQL** per i DB.

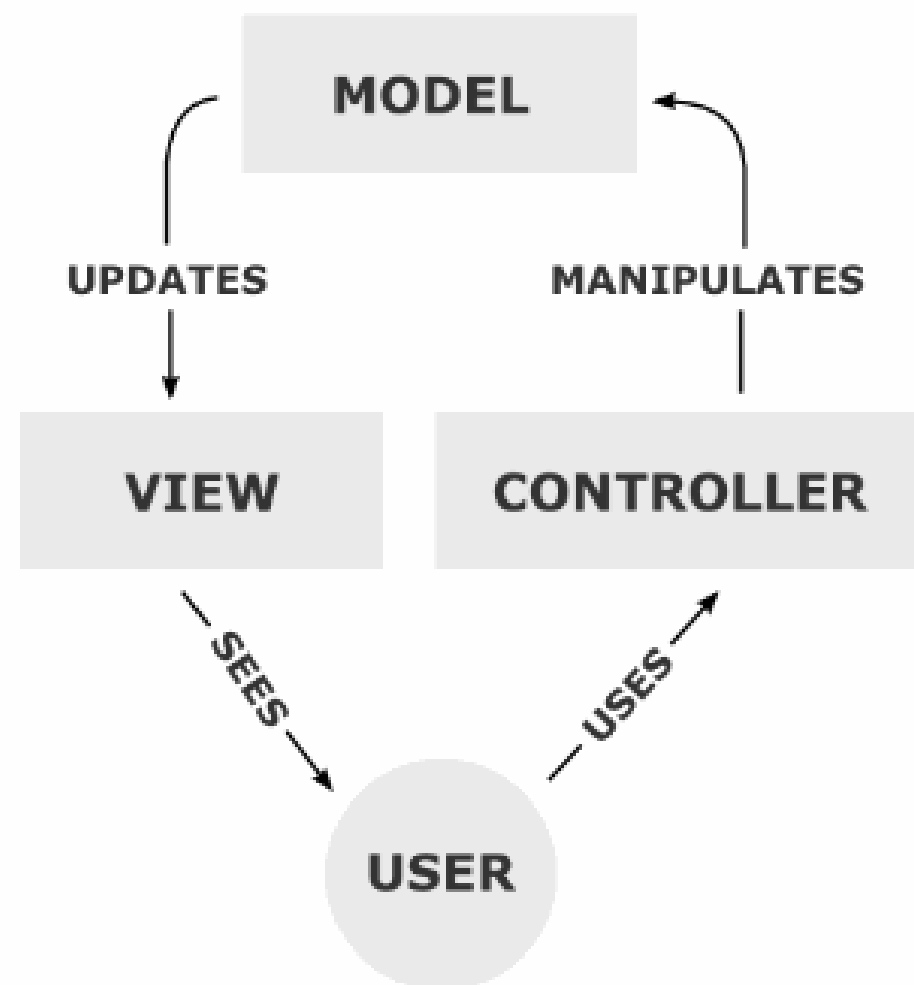
## ***I TOOLS UTILIZZATI***



# L' ARCHITETTURA DEL SOFTWARE

Per l'architettura del software abbiamo optato per 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 **3 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

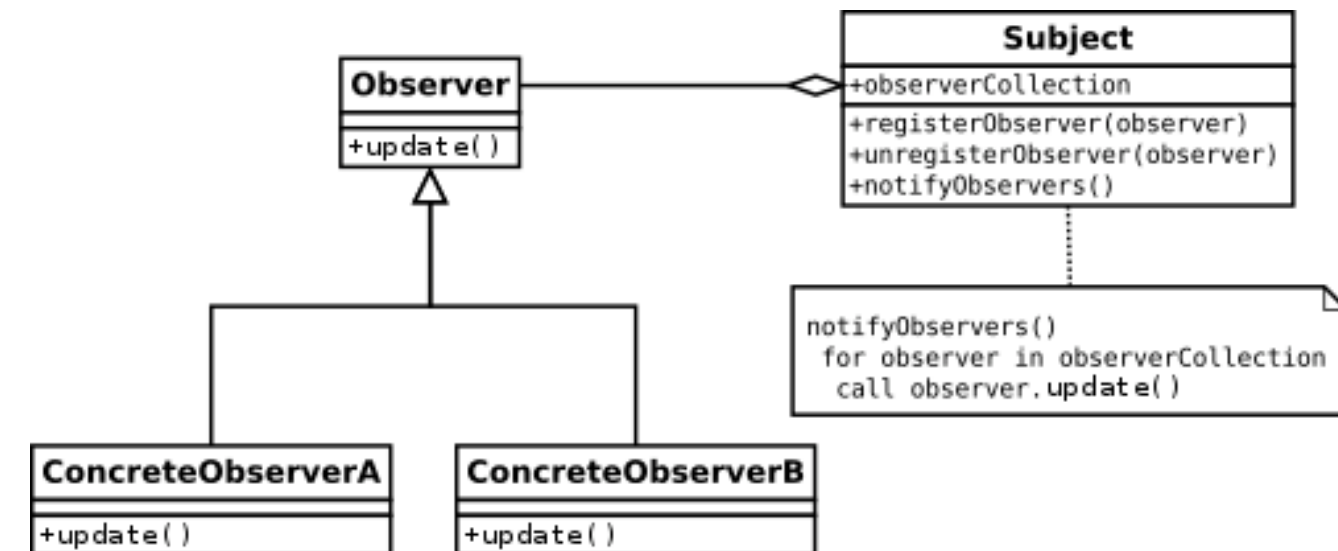
Riceve i comandi dell'utente, generalmente dalla interfaccia grafica (View) e li attua modificando lo stato delle altre due componenti.

# I DESIGN PATTERN UTILIZZATI



## SINGLETON PATTERN

E' 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 PATTERN

E' 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. Nella nostra app questo pattern permette al sistema di inviare notifiche agli utenti quando un'auto richiesta non è disponibile al noleggio. L'Autonoleggio è il subject concreto (Observable) mentre l'utente è l'Observer.

# ***COME ABBIAMO USATO GITHUB***

Github ci ha dato una mano con la fase di **Configuration Management**.

Prevalentemente ciascuno di noi svolgeva il proprio compito e poi creava una **pull request** per ottenere la conferma che le modifiche effettuate andassero bene.

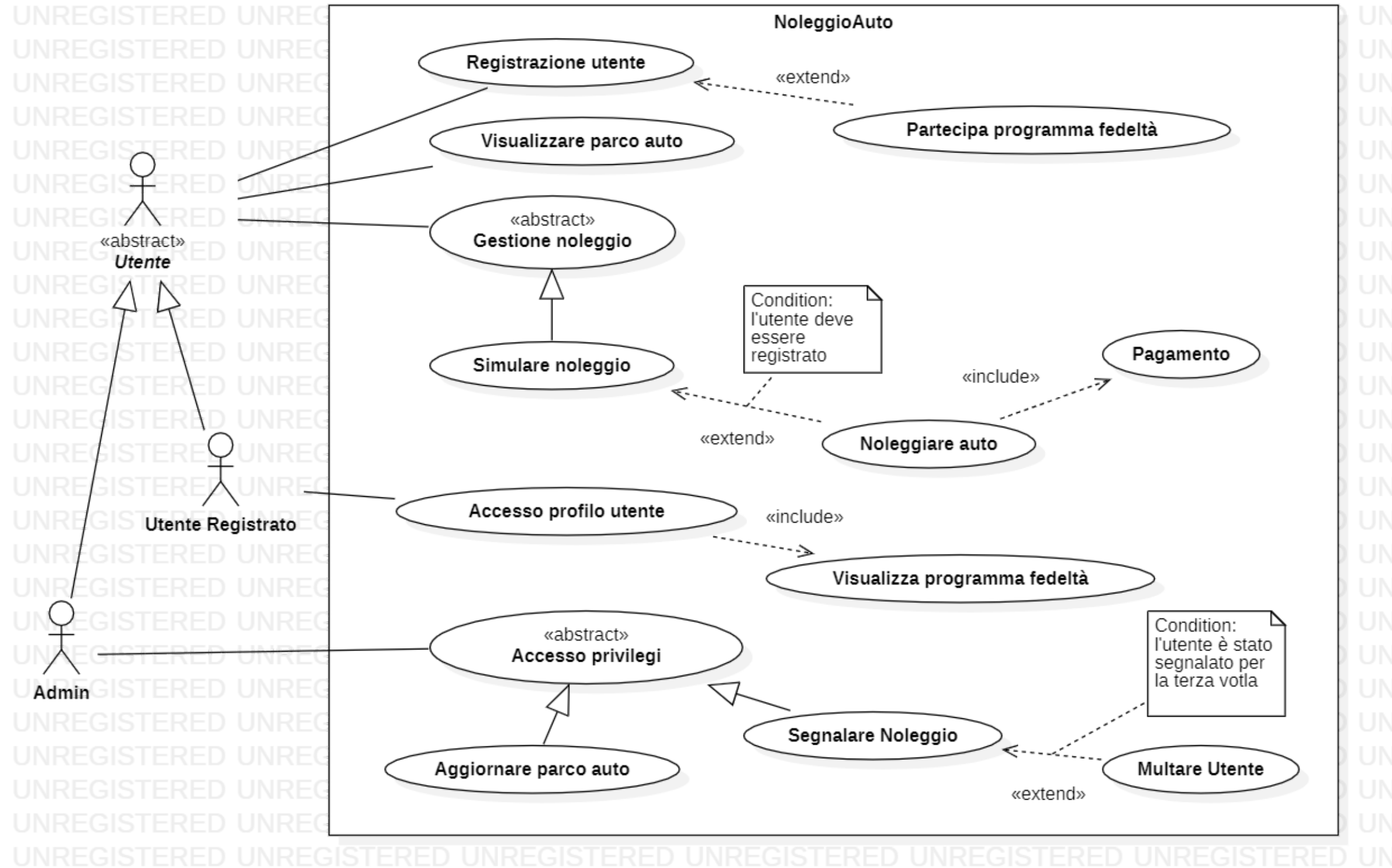
Sono state create inoltre diverse **issues** per i medesimi motivi.

Ciascuno di noi ha anche lavorato sui **branch locali** al fine di testare dei cambiamenti "internamente" per poi riportarli sul **main branch**.



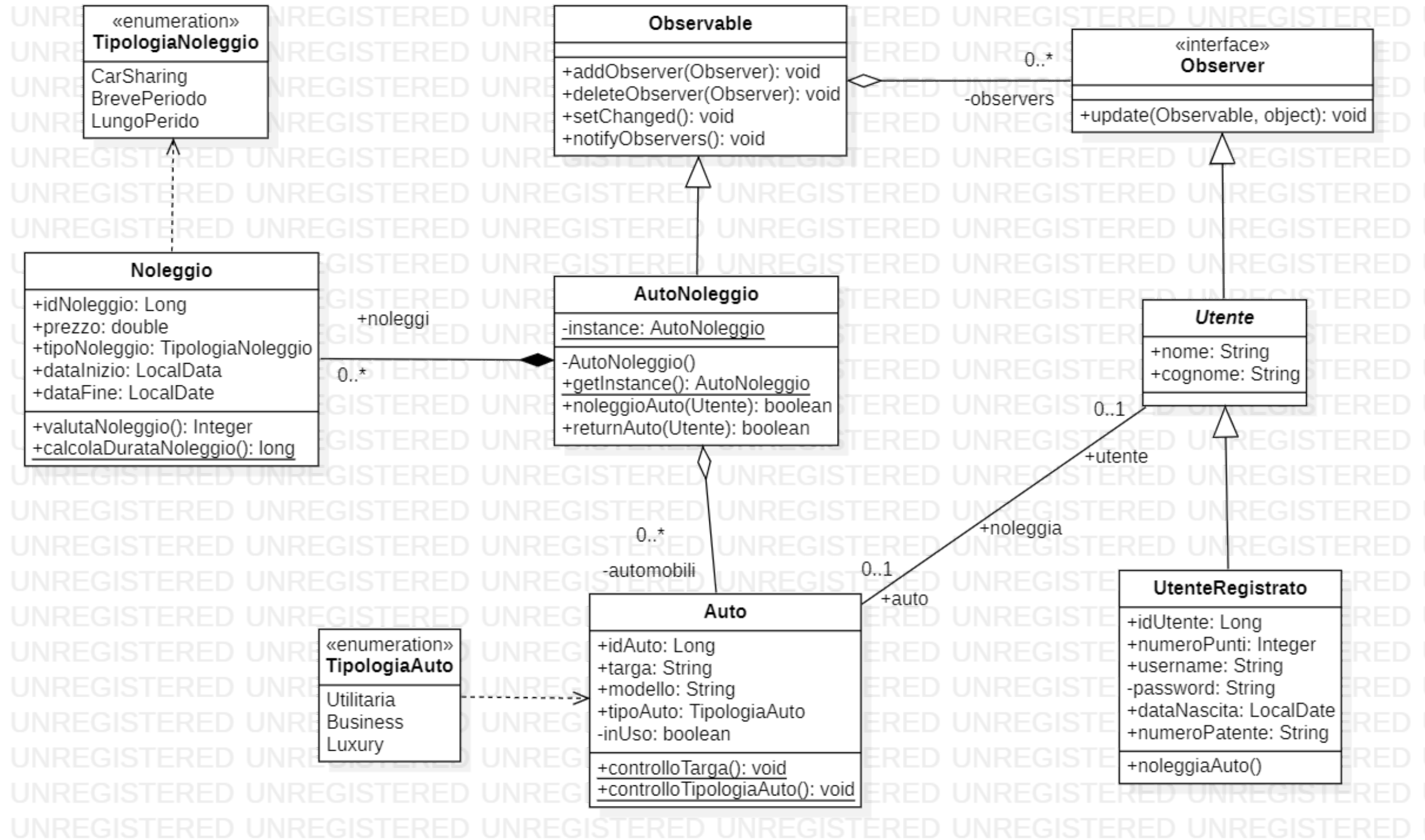
# I DIAGRAMMI UML

## Use Case Diagram



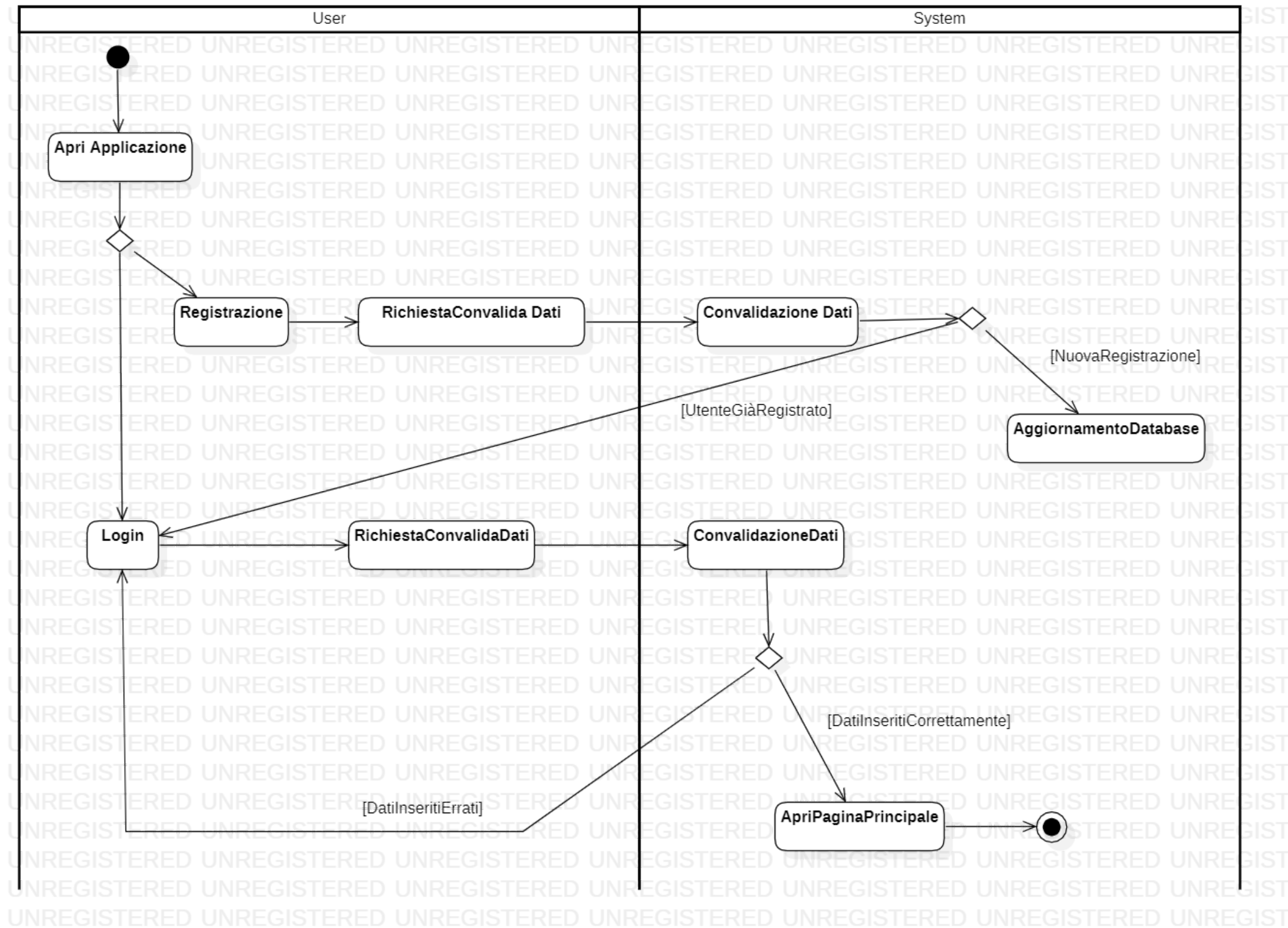
# I DIAGRAMMI UML

## Class Diagram



# I DIAGRAMMI UML

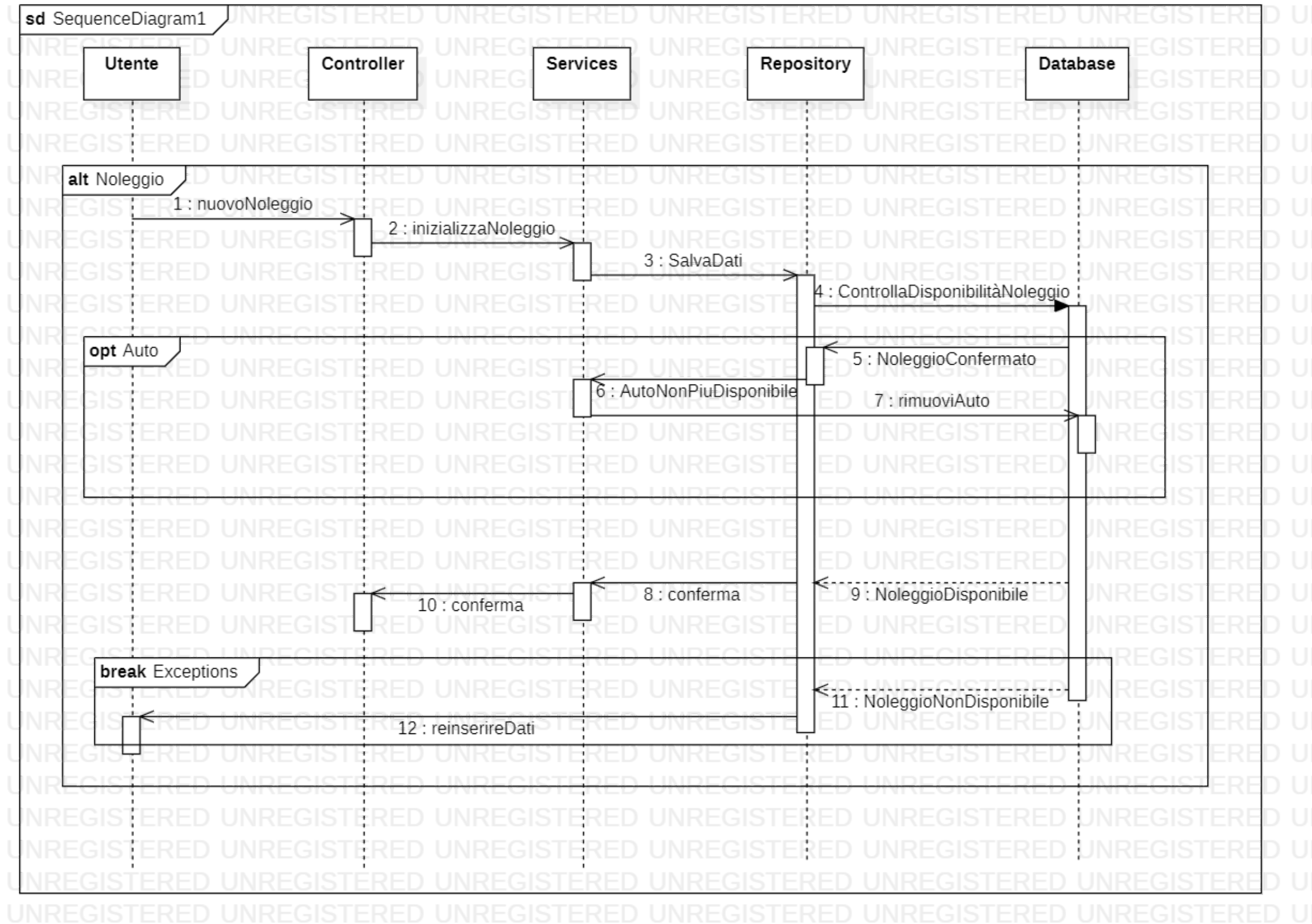
## Activity Diagram





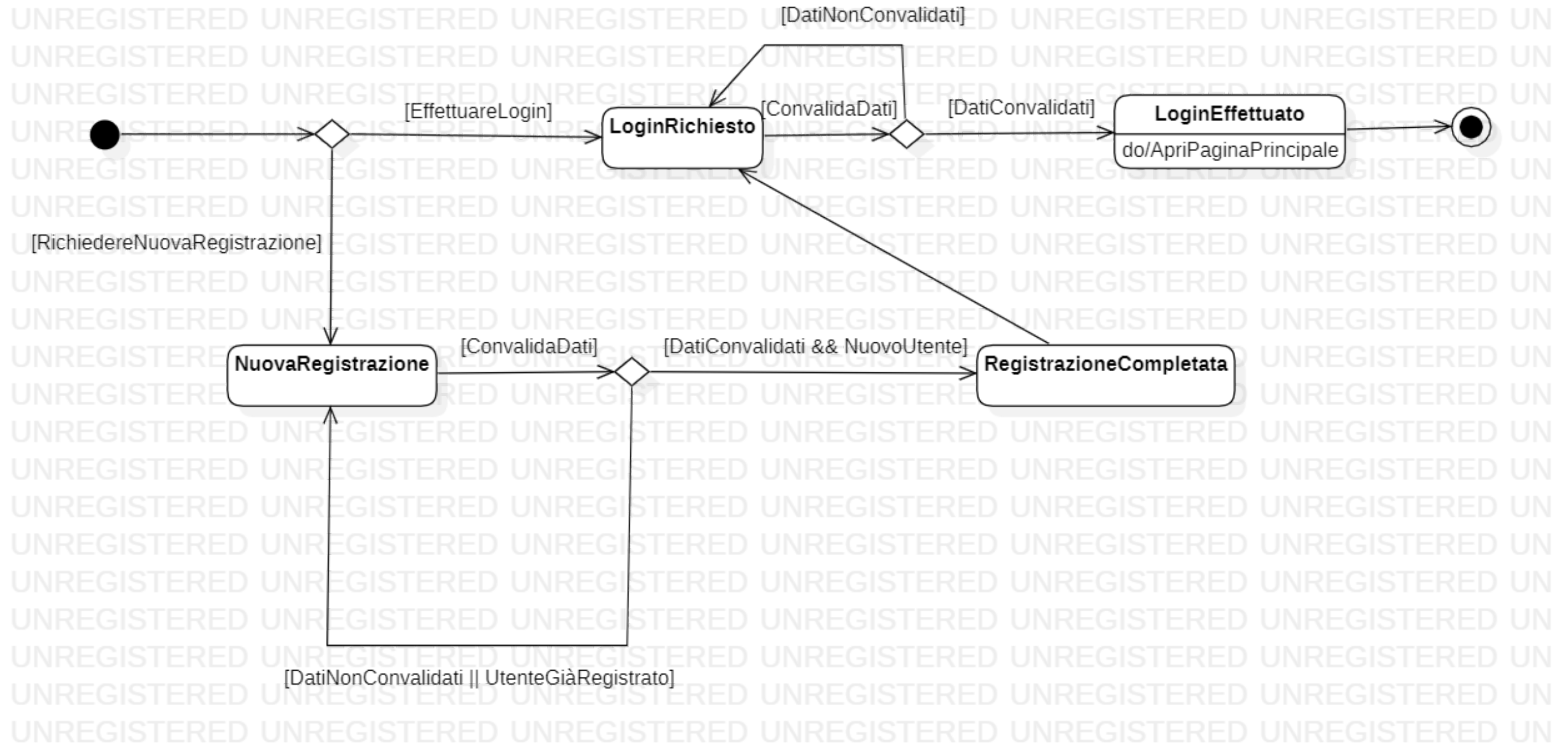
# I DIAGRAMMI UML

## Sequence Diagram



# I DIAGRAMMI UML

## State Machine Diagram



# ***LA FASE DI IMPLEMENTAZIONE***

Siamo partiti dallo **scheletro del codice** generato dal plugin **Rebel** dal Class Diagram.

Da qui abbiamo poi implementato i diversi metodi di ciascuna classe. Abbiamo utilizzato **Maven** per la gestione del progetto e la compilazione del codice. Per inizializzare il progetto abbiamo usato **SpringBoot**.

Concluso il **backend** abbiamo pensato di realizzare il **frontend** sfruttando **Angular** che anche attraverso le sue componenti ci ha permesso di introdurre l'interfaccia grafica della nostra app.



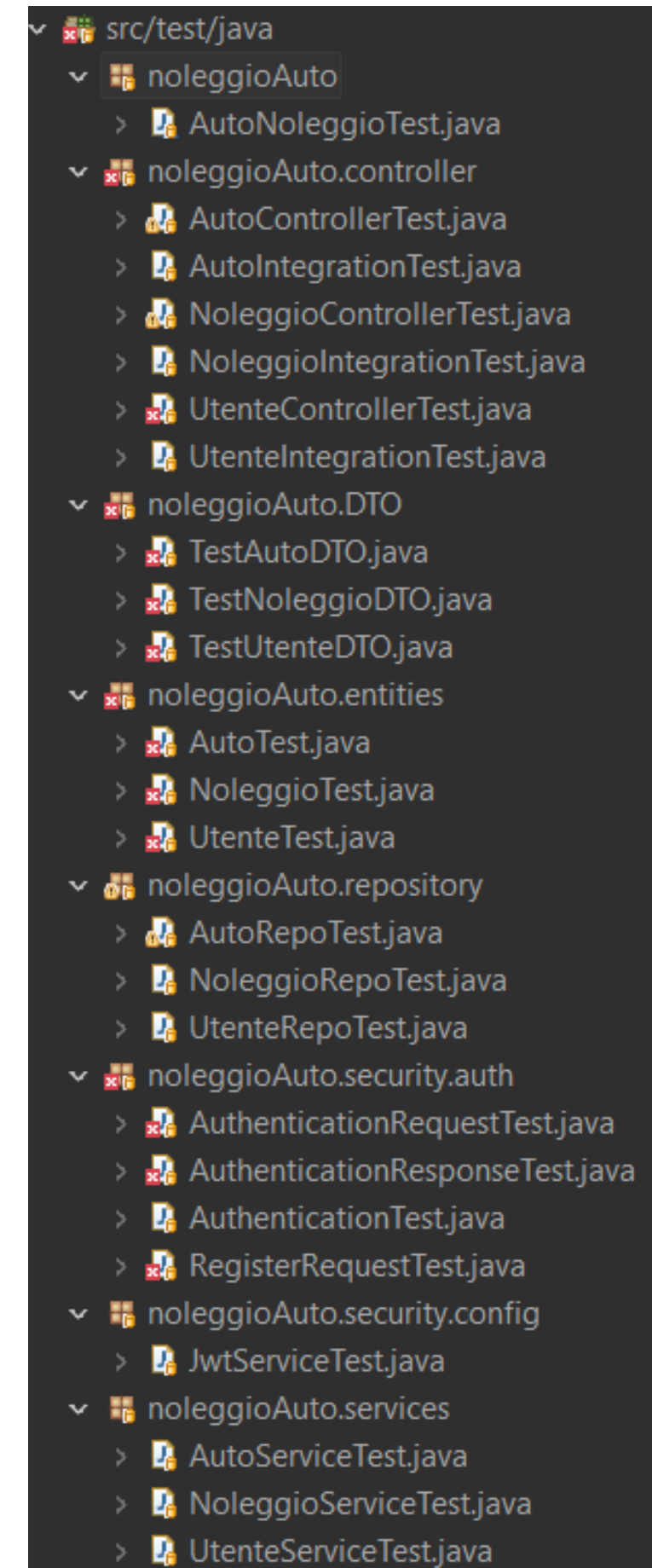


# LA FASE DI TESTING

Lo sviluppo con Extreme Programming viene riconosciuto come **Test Driven Development**. Per questo motivo abbiamo dato enfasi ai test generandone diversi.

I test sono stati effettuati cercando di coprire buona parte del codice. Il **coverage raggiunto** è del **71.6%**. Inizialmente sono stati effettuati i test sulle varie entità (classi Auto, Noleggio, Utente) accertandoci che le funzioni basilari funzionassero al meglio. Di seguito abbiamo realizzato dei test per il controller e per verificare che le richieste http funzionassero correttamente.

Tutti i test sono stati realizzati con **JUnit** e **SpringBoot Test** con **Mockito**.

The JUnit logo, with 'J' in green and 'Unit' in red.

# ***LE DIFFICOLTA' INCONTRATE***

Per la realizzazione di questo progetto abbiamo incontrato diverse difficoltà di vario tipo:



## **GESTIONE DEL TEMPO**

Ciascuno di noi, durante questi mesi è stato impegnato con diverse attività tra cui lavoro ed altri esami, per cui incontrarci periodicamente ha rappresentato un problema per la realizzazione del progetto



## **IMPLEMENTAZIONE DI FUNZIONALITA'**

Abbiamo incontrato alcune difficoltà nell'implementare alcune funzioni tra cui quelle legate al programma fedeltà, per esempio il conteggio dei punti ed i benefici legati ad essi.

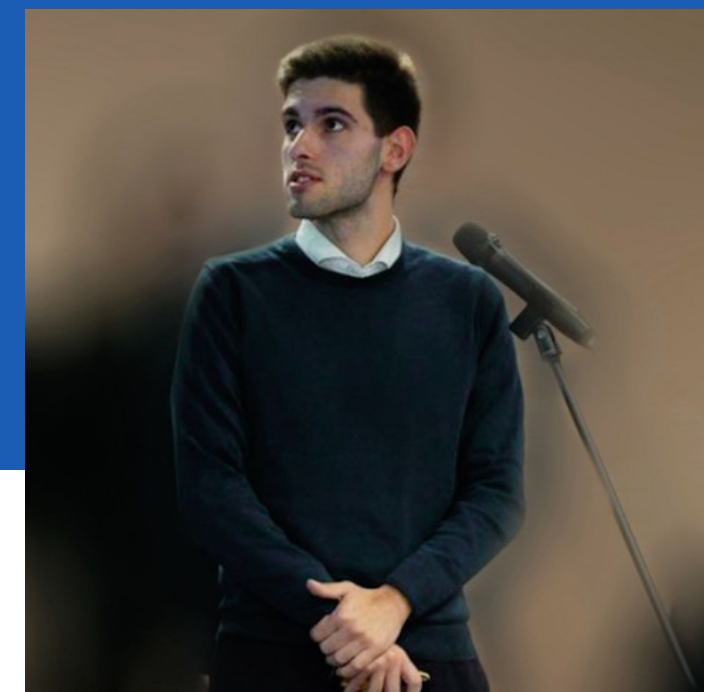
# ***IL TEAM***



**BENIAMINO  
INFANTE**



**IBRAHIMA  
SARR**



**ALEXANDER  
RUBINO**



# IL RISULTATO FINALE

