



UNIVERSITÀ^{DEGLI STUDI} DI
NAPOLI FEDERICO II

DOCUMENTAZIONE SOFTWARE

RentApp: WebApp per la gestione di Autonoleggi

Anno Accademico 2020/2021



RentAPP

Professoressa

Fasolino Anna Rita

Alunni

Ambrosio Aniello

Matricole

0001/19618

Aramu Stefano

0001/19650

Di Monaco Stefano

M63001220

Picillo Giovanni

M63001100

Picillo Giulio

M63001099

Indice

PRESENTAZIONE DEL PROGETTO	5
CAPITOLO 1 : Descrizione del progetto.....	6
1.1 Processo di Sviluppo	6
1.2 Gestione del team e suddivisione del lavoro	6
1.3 Stima dei costi di sviluppo.....	6
1.4 Primo workshop dei requisiti.....	10
1.5 Secondo workshop dei requisiti.....	10
1.6 Terzo workshop dei requisiti	11
1.7 Strumenti utilizzati	11
CAPITOLO 2 : SPECIFICA DEI REQUISITI.....	12
2.1 Descrizione testuale dei requisiti	12
2.2 Prospettive.....	12
2.3 Requisiti non funzionali.....	13
2.3.1 Sicurezza.....	13
2.3.2 Usabilità.....	13
2.3.3 Affidabilità	13
2.3.4 Performance	14
2.3.5 Supportabilità	14
2.4 Requisiti funzionali e parti interessate	15
2.5 Regole di Business non Descrittive: Vincoli e Derivazioni	16
2.6 Regole di Business Descrittive : GLossario	17
CAPITOLO 3 : ANALISI DEI REQUISITI.....	18
3.1 Identificazione degli attori.....	18
3.2 Diagramma dei casi d'uso	18
3.3 Descrizione breve dei casi d'uso	19
3.3.1 Caso d'uso 1 : Effettua login	19
3.3.2 Caso d'uso 2 : Registrazione	19
3.3.3 Caso d'uso 3: Crea Parco Auto	19
3.3.4 Caso d'uso 4 : Inserisci nuovo veicolo.....	19
3.3.5 Caso d'uso 5 : Visualizza Lista Prenotazioni.....	19
3.3.6 Caso d'uso 6 : Prenota veicolo	19
3.3.7 Caso d'uso 7 : Ricerca veicolo (incluso in prenota veicolo)	19
3.4 Descrizione di dettaglio dei casi d'uso	20
3.4.1 Caso d'uso 1 : Effettua login.....	20
3.4.2 Caso d'uso 2 : Registrazione	20

3.4.3 Caso d'uso 3 : Crea Parco Auto.....	22
3.4.4 Caso d'uso 4 : Inserimento nuovo veicolo	23
3.4.5 Caso d'uso 5 : Visualizza lista prenotazioni	24
3.4.6 Caso d'uso 6 : Prenota Veicolo	25
3.4.7 Caso d'uso 7 : Ricerca Veicolo	26
3.4.8 Caso d'uso 8 : Visualizza Lista Prenotazioni Cliente	27
3.4.9 Caso d'uso 9 : LOGOUT.....	28
3.5 Tabella Obiettivi Attori	29
3.6 System Domain Model	29
3.7 Sequence diagram di analisi	30
3.7.1 Effettua login - Sequence Diagram Di Analisi.....	30
3.7.2 Registrazione - Sequence Diagram Di Analisi	30
3.7.3 Crea Parco Auto - Sequence Diagram Di Analisi.....	31
3.7.4 Inserisci nuovo veicolo - Sequence Diagram Di Analisi.....	31
3.7.5 Visualizza lista prenotazioni - Sequence Diagram Di Analisi.....	32
3.7.6 Prenota Veicolo - Sequence Diagram Di Analisi.....	32
3.7.7 Ricerca Veicolo - Sequence Diagram Di Analisi	33
3.7.8 Visualizza lista Prenotazioni Cliente - Sequence Diagram Di Analisi	33
3.7.9 Logout- Sequence Diagram Di Analisi	33
3.8 Activity diagram di analisi	34
3.8.1 Effettua login - Activity Diagram Di Analisi.....	34
3.8.2 Registrazione - Activity Diagram Di Analisi	34
3.8.3 Crea Parco Auto - Activity Diagram Di Analisi.....	35
3.8.4 Inserisci nuovo veicolo - Activity Diagram Di Analisi.....	36
3.8.5 Visualizza lista prenotazioni - Activity Diagram Di Analisi.....	37
3.8.6 Prenota Veicolo - Activity Diagram Di Analisi.....	37
3.8.7 Ricerca Veicolo - Activity Diagram Di Analisi	38
3.8.8 Visualizza lista prenotazioni Cliente - Activity Diagram Di Analisi.....	38
3.8.9 Logout - Activity Diagram Di Analisi	39
CAPITOLO 4 : Architettura Software	40
4.1 Vista Architetturale di Alto Livello	41
4.2 Viste Architetturale di Dettaglio	41
4.2.1 Vista Architetturale di dettaglio (GESTORE)	42
4.2.2 Vista Architetturale di dettaglio (Cliente).....	42
4.3 Diagramma di Contesto.....	43
4.4 Diagramma delle Componenti.....	44
4.5 Sequence Diagram di Dettaglio	45

4.5.1 Effettua login - Sequence Diagram Di Dettaglio	45
4.5.2 Registrazione - Sequence Diagram Di Dettaglio.....	46
4.5.3 Crea Parco Auto - Sequence Diagram Di Dettaglio.....	46
4.5.4 Inserisci nuovo veicolo - Sequence Diagram Di Dettaglio	47
4.5.5 Visualizza lista prenotazioni - Sequence Diagram Di Dettaglio.....	47
4.5.6 Prenota Veicolo - Sequence Diagram Di Dettaglio	48
4.5.7 Ricerca Veicolo - Sequence Diagram Di Dettaglio.....	48
4.5.8 Visualizza lista Prenotazioni Cliente - Sequence Diagram Di Dettaglio.....	49
4.5.9 Logout - Sequence Diagram Di Dettaglio.....	49
CAPITOLO 5 : Persistenza Dei Dati	50
5.1 Definizione delle Relazioni tra entità.....	51
CAPITOLO 6 : Implementazione	52
6.1 Deployment Diagram	53
6.2 Spring MVC	54
6.2.1 Utilizzo di Spring MVC	56
6.3 Spring Security	59
6.3.1 Utilizzo di Spring Security	59
6.4 Hibernate	61
6.5 Iterazioni Di Implementazione	62
CAPITOLO 7 : Gestione degli errori eD Eccezioni	64
CAPITOLO 8 : Testing	66
CAPITOLO 9 : Installazione e Congifurazione.....	72
9.1 Apache Tomcat	72
9.2 Configurazione del Database.....	76
CAPITOLO 10 : Fututre Implementazioni	84

PRESENTAZIONE DEL PROGETTO

Il progetto prevede la realizzazione di un software di gestione e di prenotazione di veicoli da parte dei clienti, con il fine di poter noleggiare auto a breve e lungo termine in base al periodo di interesse.

Il software permette l'interazione di gestori e clienti attraverso differenti interfacce che permettono di effettuare funzioni specifiche.

Di seguito la cronologia delle revisioni effettuate al documento.

VERSIONE	DATA	DESCRIZIONE	AUTORE
Stesura prima bozza	07/06/2021	Descrizione testuale della problematica da affrontare.	aVerySADProjectTeam
Raffinamento bozza	24/06/2021	Raffinamento della descrizione testuale.	aVerySADProjectTeam
1° Aggiornamento	05/07/2021	Descrizione Del Progetto	aVerySADProjectTeam
2° Aggiornamento	06/07/2021	Descrizione Del Progetto	aVerySADProjectTeam
3° Aggiornamento	06/07/2021	Studio Di Fattibilità	aVerySADProjectTeam
4° Aggiornamento	07/07/2021	Specifiche Dei Requisiti	aVerySADProjectTeam
5° Aggiornamento	07/07/2021	Glossario	aVerySADProjectTeam
6° Aggiornamento	07/07/2021	Analisi dei Requisiti - Use Case	aVerySADProjectTeam
7°Aggiornamento	07/07/2021	Analisi dei Requisiti – Sequence di Analisi	aVerySADProjectTeam
8°Aggiornamento	07/07/2021	Analisi dei Requisiti - Activity Diagram	aVerySADProjectTeam
9°Aggiornamento	08/07/2021	Aggiornamento Analisi dei Requisiti – Sequence	aVerySADProjectTeam
10°Aggiornamento	09/07/2021	Specifiche dei requisiti – requisiti non funzionali	aVerySADProjectTeam
11°Aggiornamento	09/07/2021	Specifiche dei requisiti – requisiti funzionali	aVerySADProjectTeam
12°Aggiornamento	09/07/2021	Analisi dei requisiti – Identificazione attori	aVerySADProjectTeam
13°Aggiornamento	10/07/2021	Analisi dei requisiti – Casi d'uso	aVerySADProjectTeam
14°Aggiornamento	11/07/2021	Analisi dei requisiti – SDM e Sequence di Analisi	aVerySADProjectTeam
15°Aggiornamento	13/07/2021	Architettura software – Viste Architettoniche	aVerySADProjectTeam
16°Aggiornamento	14/07/2021	Architettura software -Sequence di Dettaglio	aVerySADProjectTeam
17°Aggiornamento	15/07/2021	Specifiche dei Requisiti – Glossario, Vincoli e Derivazioni	aVerySADProjectTeam
18°Aggiornamento	15/07/2021	Analisi dei Requisiti - Use Case	aVerySADProjectTeam
19°Aggiornamento	15/07/2021	Analisi dei requisiti – Sequence di Analisi, Activity Diagram	aVerySADProjectTeam
20°Aggiornamento	15/07/2021	Architettura Software – Sequence di dettaglio	aVerySADProjectTeam
21°Aggiornamento	19/07/2021	Architettura Software – Viste Architettoniche	aVerySADProjectTeam
22°Aggiornamento	20/07/2021	Persistenza dei dati	aVerySADProjectTeam
23°Aggiornamento	20/07/2021	Implementazione e Gestione errori ed eccezioni	aVerySADProjectTeam
24°Aggiornamento	21/07/2021	Testing	aVerySADProjectTeam
25°Aggiornamento	22/07/2021	Manuale di Utilizzo del software	aVerySADProjectTeam
26°Aggiornamento	22/07/2021	Installazione	aVerySADProjectTeam
27°Aggiornamento	22/07/2021	Implementazioni Future	aVerySADProjectTeam

CAPITOLO 1 : DESCRIZIONE DEL PROGETTO

1.1 PROCESSO DI SVILUPPO

Per la realizzazione del progetto ci si è avvalsi di tecniche di sviluppo agili, per far fronte alle possibili modifiche in corso d'opera e per ridurre al minimo i possibili fallimenti.

In particolare, è stata adottata una tecnica di sviluppo agile, incrementale e iterativo basata su *Unified Process*, realizzata nell'arco di circa sette settimane.

1.2 GESTIONE DEL TEAM E SUDDIVISIONE DEL LAVORO

Il team è composto da cinque membri. La maggior parte delle fasi di progettazione si è svolta in gruppo in modo tale da ottimizzare la qualità del lavoro ed avere la massima efficienza per il suo sviluppo. Le attività svolte autonomamente dai singoli membri o da sottogruppi sono state in fase di finalizzazione sottoposte ad un'attenta analisi da parte dei restanti membri del team.

1.3 STIMA DEI COSTI DI SVILUPPO

Per il calcolo dei costi di sviluppo del progetto, ci si è basati sulla tecnica di calcolo degli *Use Case Point*. Dopo un'attenta analisi sui possibili use-case del progetto si è fatta una stima dei differenti 8 indici di stima: come prima approssimazione si è proceduto al calcolo dell'indicatore UUCP (Unadjusted Use Case Point):

$$UUCP = UUCW + UAW = 95 + 13 = 108.$$

Dove **UUCW** o *Unadjusted Use Case Weight* può essere ottenuto sommando i prodotti di ciascun caso d'uso per il peso assegnato ad esso.

La complessità, e di conseguenza il peso, per il singolo caso d'uso è stato calcolato in base al numero di Transazioni calcolato contando sia quelle relative allo scenario di successo che quelle relative a flussi alternativi come potrebbero essere quelli di gestione delle eccezioni.

La complessità degli use case è stata attribuita utilizzando la seguente guida tabellare.

Use Case Complexity	Number of transaction	Weight
Semplice	3 o meno	5
Moderato	Da 4 a 7	10
Complesso	Piu di 7	15

Per le transazioni dettagliate riguardanti i casi d'uso si rimanda il lettore al secondo capitolo della documentazione.

Di seguito il calcolo in dettaglio della UUCW

Complessità	Peso	Numero di Use Case	Prodotto
Semplice	5	1	5
Moderato	10	6	60
Complesso	15	2	30
UUCW			95

In particolare, la complessità è stata assegnata come di seguito:

Use Case	Transazioni	UC Complexity
Login	6	Moderato
Registrazione	8	Complesso
Crea Parco Auto	4	Moderato
Inserisci Nuovo Veicolo	5	Moderato
Visualizza Lista Prenotazioni	4	Moderato
Prenota Veicolo	8	Complesso
Ricerca Veicolo	4	Moderato
Visualizza Lista Prenotazioni Cliente	4	Moderato
Logout	2	Semplice

Il **UAW** o *Unadjusted Actor Weight* può essere ottenuto similmente al caso precedente, cioè sommando i risultati del prodotto tra il numero di attori e il peso associato al loro tipo.

Il tipo , e di conseguenza il peso, per il singolo attore è stato calcolato in base al modo in cui essi interagiscono con il sistema.

Per il calcolo del tipo si è scelto di utilizzare la seguente guida tabellare.

Actor Type	Tipo di Interazione	Weight
Semplice	Un sistema esterno che interagisce tramite API o CUI	1
Moderato	Un sistema esterno che interagisce tramite protocollo	2
Complesso	Una persona che interagisce tramite una UI testuale o una GUI	3

Per conoscere il ruolo e lo scenario d'utilizzo di ogni utente si rimanda il lettore al secondo capitolo della documentazione. Di seguito il calcolo in dettaglio della UAW

Complessità	Peso	Numero di Attori	Prodotto
Semplice	1	1	1
Moderato	2	0	0
Complesso	3	4	12
UAW			13

In particolare, la complessità è stata assegnata come di seguito:

Attore	Tipo Interazione	Actor Type
Utente non registrato	GUI	Complesso
	GUI	Complesso
	GUI	Complesso
	GUI	Complesso
Sistema esterno di pagamento	API	Semplice

Lo UUCP non è in grado di darci un'informazione sufficiente per effettuare una corretta analisi di fattibilità. C'è bisogno di apportare degli accorgimenti dovuti alla complessità di alcuni fattori associati allo sviluppo del sistema. È possibile dunque calcolare un *Technical Complexity Factor* o **TFC**.

$$TCF = 0,6 + (0,01 * TFactor) = 0,6 + (0,01 * 38,5) = 0,985$$

Il *TFactor* è un indice che viene calcolato su 13 fattori sommando la moltiplicazione tra il peso del relativo fattore con l'impatto attribuito a tale fattore.

Il peso dei vari fattori è stato attribuito utilizzando la seguente guida tabellare.

Factor	Descrizione	Peso
T1	Sistema distribuito	2.0
T2	Performance	1.0
T3	Efficienza per l'utente finale	1.0
T4	Complessità di elaborazione interna	1.0
T5	Riusabilità del codice	1.0
T6	Facile da installare	0.5
T7	Facile da usare	0.5
T8	Portatile	2.0
T9	Facile da cambiare	1.0
T10	Elaborazione parallela	1.0
T11	Caratteristiche di sicurezza	1.0
T12	Accesso diretto da terzi	1.0
T13	Formazione per utenti finali	1.0

Di seguito il calcolo dettagliato del TFactor

Factor	Descrizione	Peso	Impatto	Prodotto
T1	Sistema distribuito	2.0	2	4
T2	Performance	1.0	5	5
T3	Efficienza per l'utente finale	1.0	5	5
T4	Complessità di elaborazione interna	1.0	2	2
T5	Riusabilità del codice	1.0	1	1
T6	Facile da installare	0.5	0	0
T7	Facile da usare	0.5	5	2.5
T8	Portatile	2.0	4	8
T9	Facile da cambiare	1.0	3	3
T10	Elaborazione parallela	1.0	0	0
T11	Caratteristiche di sicurezza	1.0	4	4
T12	Accesso diretto da terzi	1.0	3	3
T13	Formazione per utenti finali	1.0	1	1
TFactor				38.5

C'è inoltre bisogno di effettuare un'ultima accortezza per valutare la fattibilità del sistema in quanto esistono degli aggiustamenti dovuti alla complessità dell'ambiente. L'indice che ci stima la complessità dell'ambiente è chiamato *Enviroment Factor* o EF e viene calcolato come segue.

$$EF = 1,4 + (-0,03 * EFactor) = 1,4 + (-0,03 * 32) = 0,44$$

L'*EFactor* è un indice ottenuto sommando le moltiplicazioni fra il peso associato ad un fattore ambientale e la relativa importanza che si da a quest'ultimo. Esistono 8 fattori diversi il cui peso è stato attribuito utilizzando la seguente guida tabellare.

Factor	Descrizione	Peso
E1	Familiarità con il processo di sviluppo	1.5
E2	Esperienza di applicazione	0.5
E3	Esperienza object-oriented del team	1.0
E4	Capacità di analisi	0.5
E5	Motivazione del team	1.0
E6	Stabilità dei requisiti	2.0
E7	Personale part-time	-1.0
E8	Complessità del linguaggio utilizzato	2.0

Di seguito il calcolo dettagliato del TFactor

Factor	Descrizione	Peso	Impatto	Prodotto
E1	Familiarità con il processo di sviluppo	1.5	5	7,5
E2	Esperienza di applicazione	0.5	3	1,5
E3	Esperienza object-oriented del team	1.0	5	5
E4	Capacità di analisi	0.5	4	2
E5	Motivazione del team	1.0	5	5
E6	Stabilità dei requisiti	2.0	3	6
E7	Personale part-time	-1.0	1	-1
E8	Complessità del linguaggio utilizzato	2.0	3	6
EFactor				32

Infine, si calcola lo *Use Case Points* o **UCP**, che fornisce una stima delle dimensioni del progetto, in questo modo:

$$UCP = UUCP \times TCF \times EF = 108 \times 0,985 \times 0,44 = 46,8072$$

In assenza di dati storici aziendali si adotta il rapporto *ore* × *UCP* proposto da *Kirsten e Ribu* al fine di stimare la durata del progetto:

$$Durata_{min} = 15h \times 46,8072 = 702,108 \cong 702h\ 6min$$

$$Durata_{max} = 30h \times 46,8072 = 1404,216 \cong 1404h\ 13min$$

Supponendo poi che ogni membro del team lavori per circa 35 ore a settimana, facendo un rapido calcolo approssimativo saranno necessarie dalle 4 alle 8 settimane lavorative per completare lo sviluppo.

1.4 PRIMO WORKSHOP DEI REQUISITI

Nella prima settimana si è tenuto il primo workshop dei requisiti, durante il quale sono stati stabiliti gli obiettivi della prima iterazione ed è stata effettuata una raccolta di requisiti funzionali alto livello. Sono stati poi identificati i principali requisiti non funzionali che il sistema deve rispettare.

Nella prima iterazione si è scelto di concentrarsi sui casi d'uso “**Login**”, “**Inserisci nuovo veicolo**”, “**Ricerca Veicolo**”, “**Prenota Veicolo**”

1.5 SECONDO WORKSHOP DEI REQUISITI

A partire dalla terza settimana il team ha raggiunto tutti gli obiettivi della prima iterazione e ha iniziato la seconda iterazione, fase in cui è stato ultimato il materiale del primo workshop. È stato poi scelto altri casi d'uso da analizzare: “**Crea Parco Auto**”, “**Visualizza Prenotazioni**”, “**Registrazione**”.

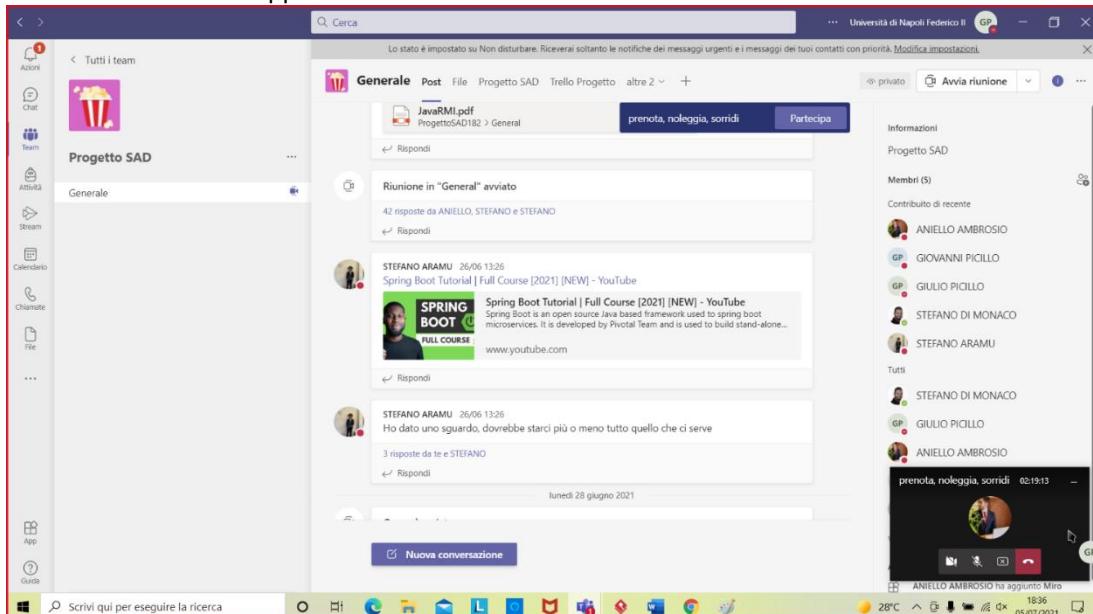
1.6 TERZO WORKSHOP DEI REQUISITI

A partire dalla quarta settimana il team ha raggiunto tutti gli obiettivi della seconda iterazione, corretto alcuni bug emersi in fase di testing su quanto prodotto nella prima settimana e ha iniziato la terza iterazione, fase in cui è stato ultimato il materiale dei primi due workshop. È stato poi scelto altri due casi d'uso da analizzare: “**Visualizza Prenotazioni Utente**”, “**Logout**”.

1.7 STRUMENTI UTILIZZATI

Il progetto è stato interamente sviluppato in Smart working grazie all'utilizzo di tecnologie ICT che hanno permesso il corretto svolgimento delle attività progettuali a distanza.

L'utilizzo principalmente del software **Microsoft Teams** ha permesso al Team di lavorare in modo coordinato e di utilizzare una serie di tool messi a disposizione da quest'ultimo al fine di facilitare il lavoro e migliorarne la qualità. All'interno del software è stato possibile creare uno spazio di lavoro condiviso tra i membri del team. Per lo sviluppo dell'applicazione sono stati utilizzati ulteriori strumenti di supporto:



- **Spring Tool Suite 4**: ambiente di sviluppo completo per sviluppare applicazioni web, sfruttando differenti linguaggi di programmazione, in particolare il team ha scelto di utilizzare il linguaggio Java per lo sviluppo del software.
- **Microsoft SQL Server Management Studio**: database di supporto al sistema.
- **Trello** : valido strumento di project management.
- **Visual Paradigm**: software di supporto per i design di diagrammi UML relativi alla progettazione del sistema.
- **Microsoft Office**: piattaforma utilizzata nella sua forma collaborativa per la redazione della documentazione di supporto al sistema.
- **Miro**: è una lavagna collaborativa online utilizzata per collaborare e scambiare idee
- **Bootstrap**: Bootstrap è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web.
- **Tomcat**: Tomcat è un server web open source. Implementa le specifiche JavaServer Pages e servlet, fornendo una piattaforma software per l'esecuzione di applicazioni web
- **Maven**: Maven è uno strumento di gestione di progetti software basati su Java e build automation.

CAPITOLO 2 : SPECIFICA DEI REQUISITI

Nel seguente capitolo si specificano i requisiti del sistema software “RentAPP”.

2.1 DESCRIZIONE TESTUALE DEI REQUISITI

Si vuole realizzare un sistema software per la gestione di autonoleggi, in particolare si vogliono amministrare uno o più parchi auto dei gestori con le relative prenotazioni da parte dei clienti

Un utente per accedere al sistema deve effettuare la registrazione inserendo le proprie generalità (nome, cognome, data e luogo di nascita, codice fiscale, partita iva, indirizzo, numero patente), username e password. Il sistema prevede l'esistenza di un utente di tipo gestore, responsabile e proprietario dei propri parchi auto, ed un utente di tipo cliente.

Al fine di effettuare l'accesso al sistema, un utente deve aver effettuato il login (codice identificativo/username, password). Il sistema poi verifica la correttezza dei dati inseriti, in caso di riscontro, mostra la dashboard corretta nel caso in cui si tratti di un utente gestore o cliente.

Dopo essersi autenticato il gestore può visualizzare la lista delle prenotazioni dei suoi parchi auto, inserire un nuovo veicolo nel parco auto desiderato selezionandolo nella lista dei propri parchi auto e specificando i relativi attributi del veicolo (targa, modello, marca, segmento, alimentazione, tipo, kilometraggio, colore, cilindrata, prezzo). Confermato l'inserimento il database viene aggiornato e il sistema mostra a video un messaggio di conferma.

Inoltre, un gestore deve poter, all'occorrenza, inserire un nuovo parco auto nel sistema, specificandone nome, dislocazione e indirizzo. Il sistema assegnerà al nuovo parco auto un id univoco.

Un cliente, dopo essersi loggato, per poter effettuare la ricerca di un veicolo deve specificare il periodo della prenotazione (data inizio e data fine), luogo di ritiro e consegna (necessariamente stessa sede), il sistema mostrerà a video la lista di veicoli disponibili, permettendo all'utente di selezionare quello che più si adatta alle proprie esigenze. Successivamente al cliente verrà mostrato il dettaglio della prenotazione che include le caratteristiche del veicolo, il prezzo totale per il periodo selezionato e dopo aver dato conferma, il software dovrà interfacciarsi con un sistema esterno di pagamento che permetterà di validare o meno la prenotazione del cliente. Inoltre, Il cliente può visualizzare la lista delle prenotazioni effettuate.

Gli utenti potranno poi effettuare il logout dalla loro area riservata.

2.2 PROSPETTIVE

Lo scopo di questo progetto è la realizzazione di un software, RentAPP, che si inserisce nel mercato al fine di fornire uno strumento funzionale che permetta in maniera rapida ed efficiente la prenotazione di veicoli predisposti al mercato del noleggio. Il sistema si pone come principale obiettivo quello di soddisfare tutti i vincoli non funzionali al fine di proporsi qualitativamente sul mercato come migliore software. A tale scopo sono stati analizzati i cinque requisiti non funzionali al fine di sviluppare un sistema che rispettasse tutti i vincoli imposti.

2.3 REQUISITI NON FUNZIONALI

2.3.1 SICUREZZA

Per evitare che chiunque possa effettuare operazioni attraverso il sistema, ogni utente che vuole interagire con esso deve essere autenticato.

Inoltre, per garantire la sicurezza dei pagamenti ed evitare il trafugamento di dati sensibili dell'utente, il sistema deve interfacciarsi con un istituto di riscossione noto ed affidabile.

2.3.2 USABILITÀ

Il cliente deve essere in grado di visualizzare le schermate dell'applicativo in maniera chiara.

Quindi:

- Il testo deve risultare facilmente visibile.
- Evitare l'utilizzo di colori non complementari ed utilizzare una sola gamma di colori funzionali.

Il sistema inoltre deve garantire una ricerca veloce basata su parametri essenziali e che ponga subito all'attenzione del suddetto un preventivo veritiero per il noleggio del veicolo a cui è interessato.

La prenotazione deve essere avvertita dall'utente come una sequenza fluida di operazioni al fine di non incutere timori nel processo di pagamento e aumentare dunque la probabilità di un secondo utilizzo da parte di un nuovo cliente.

Le informazioni devono essere trasmesse al sistema tramite una GUI user-friendly, in modo che il loro inserimento risulti il più semplice possibile così da abbattere possibili errori di inserimento dovuti ad indicazioni non chiare fornite dal sistema.

2.3.3 AFFIDABILITÀ

La persistenza dei dati è garantita dall'utilizzo di un database relazionale, in modo da assicurare un corretto recupero delle informazioni anche in caso di fallimento del software.

2.3.4 PERFORMANCE

Essendosi proposto sul mercato come uno dei miglior software, gli obiettivi di RentAPP sono quelli di offrire funzionalità anche complesse in modo rapido e performante:

- Un gestore deve:
 - Poder creare un parco auto in meno di 30 secondi.
 - Poder inserire un nuovo veicolo in meno di un minuto.
 - Poder leggere le proprie prenotazioni relative a tutti i parchi auto con un click.
 - Effettuare il logout con un click
- Un cliente:
 - Deve poter effettuare una ricerca inserendo solo 3 informazioni principali.
 - Deve poter concludere la prenotazione tramite un click (escludendo il processo di pagamento gestito esternamente).
 - Poder leggere le proprie prenotazioni relative a tutti i parchi auto con un click.
 - Effettuare il logout con un click
- Un utente che si collega alla web app:
 - Deve poter registrarsi in maniera rapida indicando poche informazioni essenziali per legge alla sua identificazione in quanto individuo.
 - Deve potersi loggare se registrato in maniera rapida evitando fallimenti del sistema.

Tutte queste operazioni devono concludersi con una percentuale di successo maggiore del 95%.

2.3.5 SUPPORTABILITÀ

Si divide in :

- **Adattabilità:** I diversi gestori che usufruiscono di RentAPP hanno esigenze di elaborazione e regole di business uniche durante l'elaborazione. Dunque, in diversi punti definiti dello scenario verrà abilitata la regola di business collegabile ad esso.
- **Configurabilità:** I differenti utenti dell'applicazione hanno esigenze differenti ed eseguono diverse operazioni. Pertanto, l'utilizzatore si aspetta che ad ogni tipo di utente corrisponda una configurazione di interfacce differenti.

2.4 REQUISITI FUNZIONALI E PARTI INTERESSATE

Un utente dopo essersi registrato ed autenticato al sistema può effettuare differenti operazioni in base al proprio ruolo all'interno del sistema.

Il generico utente non registrato può:

1. Registrarsi sulla WEBAPP

L'utente registrato può:

2. Effettuare l'autenticazione al sistema
3. Effettuare il Logout

Il sistema è in grado di riconoscere l'utente che ha effettuato il login e fornire funzionalità diverse in base al ruolo.

Se l'utente autenticato è un gestore esso può:

4. Creare un Parco Auto
5. Inserire un Nuovo Veicolo
6. Visualizzare la lista delle Prenotazioni

Se l'utente autenticato è un cliente esso può:

7. Ricercare un Veicolo
8. Prenotare un Veicolo
9. Visualizzare il proprio storico prenotazioni

Inoltre, un Sistema esterno di pagamento deve poter:

10. Offrire un Interfaccia di Pagamento

Si individuano dunque differenti parti interessate nel nostro sistema:

- Utente Non Registrato: è una persona che vuole visita il sito ed è visto come un potenziale cliente del sito a cui viene offerta la possibilità di iniziare ad utilizzarlo seguendo pochi passaggi essenziali offerti dalla registrazione.
- Utente: è una persona già registrata sul sito che può effettuare il login.
- Sistema di pagamento : è un sistema esterno che gestisce le transazioni bancarie atte alla conferma delle prenotazioni dei veicoli.

Per quanto riguarda gli utenti essi si specializzano in :

- Gestore: Proprietario di uno o più autonoleggi che vuole rendere disponibili sul sito i propri veicoli al fine di incrementare i propri affari.
- Cliente: Persona che necessita di noleggiare un'auto e si affida al sito per concludere l'operazione in pochi minuti.

2.5 REGOLE DI BUSINESS NON DESCrittive: VINCOLI E DERIVAZIONI

ID	<i>Regola</i>	<i>Modificabilità</i>	<i>Sorgente</i>
R-1	Il codice fiscale deve essere composto da 16 caratteri	Non modificabile	Governo italiano
R-2	La partita iva deve essere composta da 11 caratteri	Non modificabile	Unione Europea
R-3	Il numero di patente deve essere composto da 10 caratteri	Non modificabile	Ministero dei trasporti
R-4	L'username deve essere unico ed identificativo di un utente	Non modificabile	aVerySADProjectTeam
R-5	Il Nome deve essere composto da 2 a 50 caratteri	Non modificabile	aVerySADProjectTeam
R-6	Il Cognome deve essere composto da 2 a 50 caratteri	Non modificabile	aVerySADProjectTeam
R-7	L'indirizzo deve essere composto da 10 e 80 caratteri	Non modificabile	aVerySADProjectTeam
R-8	Il Luogo di nascita deve essere composto da 2 a 50 caratteri	Non modificabile	aVerySADProjectTeam
R-9	Le date devono essere inserite nel formato "DD/MM/YYYY" o "YYYY/MM/DD"	Non modificabile	Date di SQL
R-10	La data di inizio di una prenotazione deve essere anteriore alla data di fine prenotazione	Non modificabile	aVerySADProjectTeam
R-11	La data di inizio di una prenotazione deve essere posteriore alla data corrente	Non modificabile	aVerySADProjectTeam
R-12	La targa di un veicolo deve essere unica e composta da 7 caratteri	Non modificabile	Ministero dei Trasporti
R-13	Il modello di un veicolo non può essere nullo	Non modificabile	aVerySADProjectTeam
R-14	La marca di un veicolo non può essere nulla	Non modificabile	aVerySADProjectTeam
R-15	Il segmento di un veicolo non può essere nullo	Non modificabile	aVerySADProjectTeam
R-16	Il tipo di un veicolo non può essere vuoto	Non modificabile	aVerySADProjectTeam
R-17	Il colore di un veicolo non può essere nullo	Non modificabile	aVerySADProjectTeam
R-18	Il nome di un parco auto non può essere nullo	Non modificabile	aVerySADProjectTeam
R-19	La dislocazione di un parco auto non può essere nulla	Non modificabile	aVerySADProjectTeam
R-20	L'indirizzo di un parco non può essere nullo	Non modificabile	aVerySADProjectTeam
R-21	L'id univoco di un parco auto viene generato automaticamente dal sistema in maniera incrementale	Non modificabile	aVerySADProjectTeam
R-22	L'id univoco di una prenotazione viene generato automaticamente dal sistema in maniera incrementale	Non modificabile	aVerySADProjectTeam
R-23	L'id ruolo di un utente può assumere solo i valori "2" o "3" (il valore "1" è dedicato ad un ruolo di testing che permette ai developer di testare le funzionalità dell'intero sistema)	Non modificabile	aVerySADProjectTeam

2.6 REGOLE DI BUSINESS DESCRITTIVE : GLOSSARIO

<i>Termine</i>	<i>Definizione e Informazioni</i>	<i>Regole di Validazione</i>	<i>Alias</i>
<i>Prenotazione</i>	Raccoglie le informazioni relative alla prenotazione di un veicolo da parte di un cliente.	La prenotazione è identificata da un codice univoco, dove è precisata la data inizio e fine, dell'id del cliente responsabile della prenotazione e dell'id del relativo parco. Può essere effettuata solo da un cliente registrato e autenticato dal sistema.	
<i>Parco Auto</i>	Luogo dove stazionano autoveicoli.	Il parco auto è identificato da un codice, dal luogo, indirizzo e dell'id del gestore.	Autonoleggio
<i>Gestore</i>	Proprietario e responsabile dei suoi parchi auto.	Il gestore è distinto da un id di ruolo	Proprietario dell'autonoleggio
<i>Dati Utente</i>	Generalità dell'utente come nome, cognome, codice fiscale, data e luogo di nascita, partita iva, indirizzo, numero patente.	Il codice patente è univoco per ogni cliente che intende effettuare una prenotazione.	
<i>Cliente</i>	Persona che intende effettuare una prenotazione di un veicolo	Il cliente distinto da un id di ruolo	
<i>Veicolo</i>	Mezzo di trasporto	Identificato dalla targa. Possiede una serie di attributi come marca, modello, motorizzazione, tipo, km, colore, cilindrata.	
<i>Sistema esterno di pagamento</i>	Sistema che permette di validare un pagamento		
<i>Utente</i>	Utente di base che interagisce per autenticarsi.		
<i>RentAPP</i>	Nome della WebAPP che permette il noleggio di veicoli da parte di un cliente e la gestione di un parco auto da parte di un gestore		Sistema

CAPITOLO 3 : ANALISI DEI REQUISITI

Nel seguente capitolo verranno analizzati nel dettaglio i requisiti funzionali e gli attori che sono stati individuati nel capitolo 2.

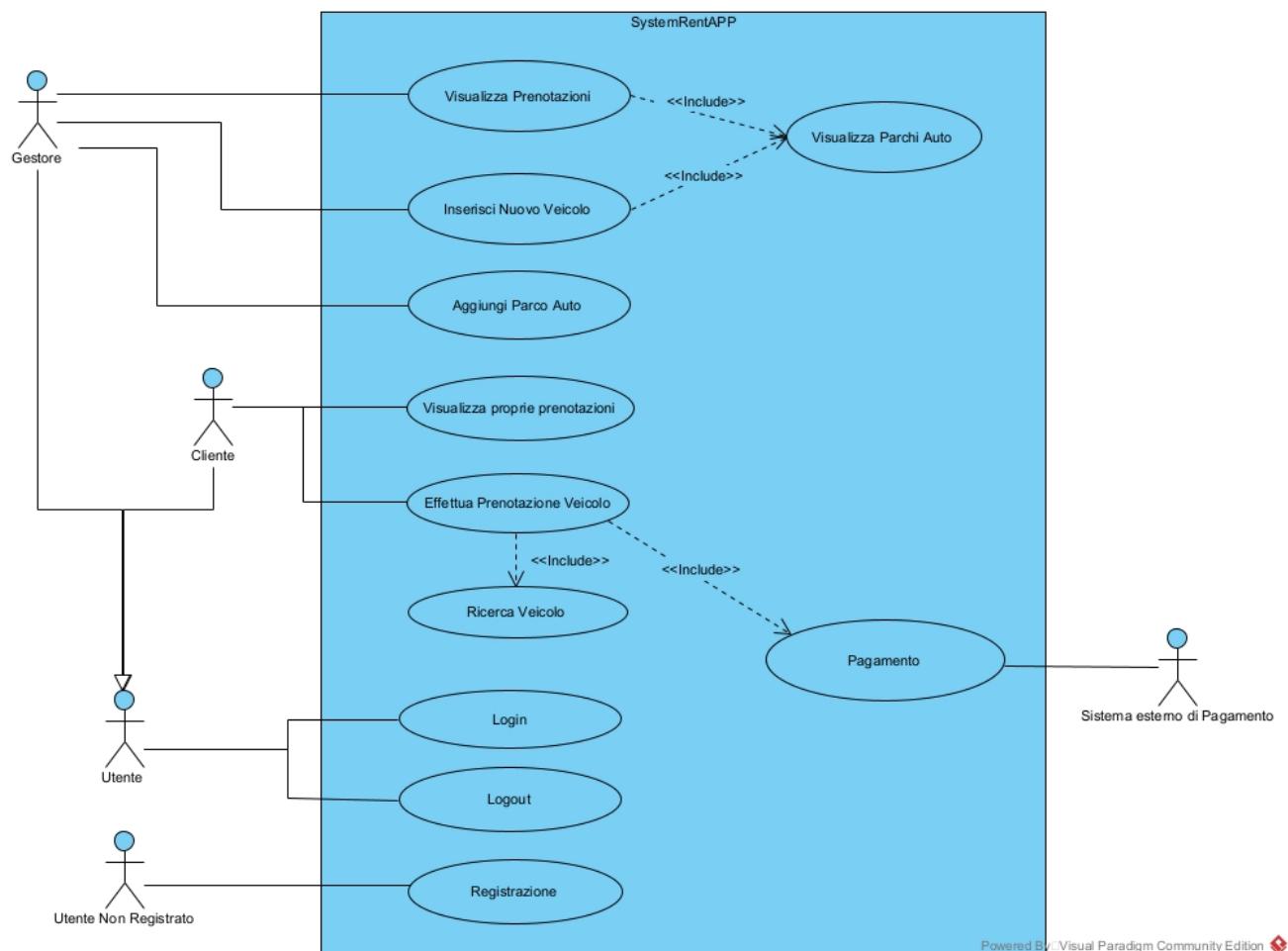
3.1 IDENTIFICAZIONE DEGLI ATTORI

Durante lo sviluppo del sistema sono stati individuati i seguenti attori primari:

- UTENTE NON REGISTRATO
- UTENTE:
 - CLIENTE
 - GESTORE
- SISTEMA ESTERNO DI PAGAMENTO

Inoltre, il sistema interagisce con un sistema esterno di pagamento per permettere al cliente di finalizzare la prenotazione di un veicolo.

3.2 DIAGRAMMA DEI CASI D'USO



3.3 DESCRIZIONE BREVE DEI CASI D'USO

3.3.1 CASO D'USO 1 : EFFETTUA LOGIN

Il caso d'uso permette all'utente di inserire username e password per accedere al sistema. Il sistema poi verifica se l'utente esiste o meno e mostra la giusta interfaccia per l'utente registrato.

3.3.2 CASO D'USO 2 : REGISTRAZIONE

Il caso d'uso permette all'utente non registrato di registrarsi nel sistema inserendo username, password e i propri dati personali. (nome, cognome, data di nascita, luogo di nascita, partita iva, codice fiscale, indirizzo). Inoltre, il sistema deve permettere all'utente se registrarsi come cliente oppure come gestore.

3.3.3 CASO D'USO 3: CREA PARCO AUTO

Il caso d'uso permette al gestore di creare un nuovo parco auto inserendo i dati di quest'ultimo. (nome del parco, indirizzo, dislocazione).

3.3.4 CASO D'USO 4 : INSERISCI NUOVO VEICOLO

Al fine di inserire un nuovo veicolo nel sistema, il gestore seleziona il parco auto all'interno della UI e inserisce i dati relativi ad un nuovo veicolo. (modello, targa, marca, segmento, alimentazione, tipo, kilometro, colore, cilindrata e prezzo giornaliero) . Il sistema deve, dopo aver controllato che il veicolo non sia già presente nel DB, inserisce il veicolo nel database e restituisce un messaggio di conferma.

3.3.5 CASO D'USO 5 : VISUALIZZA LISTA PRENOTAZIONI

Il gestore può interrogare il sistema per conoscere la lista di tutte le prenotazioni attive nei proprio parchi auto.

3.3.6 CASO D'USO 6 : PRENOTA VEICOLO

Il cliente, dopo aver effettuato la ricerca inserendo il periodo e il luogo d'interesse, intende prenotare un veicolo fra la lista dei veicoli disponibili messi a disposizioni dal sistema. Per finalizzare la prenotazione da parte del cliente, il sistema interagisce con un sistema esterno di pagamento e in caso di successo, viene mostrato al cliente il resoconto della prenotazione da lui richiesta. (data inizio, data fine, nome parco, indirizzo del parco, veicolo)

3.3.7 CASO D'USO 7: RICERCA VEICOLO (INCLUSO IN PRENOTA VEICOLO)

Al fine di poter prenotare un veicolo, il cliente deve interrogare il sistema, con il periodo e il luogo di interesse. Il sistema dopo aver effettuato una ricerca, e calcolato il costo sull'intero periodo, restituisce all'utente una lista di veicoli disponibili alla prenotazione con i relativi dettagli.

3.4 DESCRIZIONE DI DETTAGLIO DEI CASI D'USO

3.4.1 CASO D'USO 1 : EFFETTUA LOGIN

Nome del caso d'uso	Effettua login
Ambito	Applicazione RentApp
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate	Utente vuole accedere al sistema.
Precondizioni	L'utente deve essere registrato all'interno del sistema
Garanzia di successo	Il sistema mostra la corretta interfaccia
Principale scenario di successo	<ol style="list-style-type: none">1. L'utente inserisce ID e Password2. Il sistema verifica la validità dei dati3. L'utente accede all'interfaccia specifica
Estensioni	<p>Se l'utente non è registrato</p> <p>2a. Il sistema risponde con un messaggio d'errore</p> <p>Se la password inserita non è valida</p> <p>2b. Il sistema risponde con un messaggio d'errore</p> <p>2c. L'utente torna alla schermata di accesso.</p>
Requisiti particolari	Il sistema deve essere in grado, in caso di correttezza dei dati, di eseguire sempre il login con successo.
Tecnologie e variazioni dei dati	Il sistema deve essere in grado di riconoscere la tipologia di utente che sta effettuando l'accesso.
Frequenza di occorrenza	Potenzialmente continuo
Varie	

3.4.2 CASO D'USO 2 : REGISTRAZIONE

Nome del caso d'uso		Effettua registrazione
Ambito		Applicazione RentApp
Livello		Obiettivo utente non registrato
Attore primario		Utente non registrato
Parti interessate		Utente non registrato vuole registrarsi al sistema.
Precondizioni		L'utente non deve essere registrato all'interno del sistema
Garanzia di successo		L'utente viene correttamente registrato in seguito all'inserimento dei dati all'interno del database. Il sistema mostra l'interfaccia di login.
Principale scenario di successo		<ol style="list-style-type: none"> 1. L'utente sceglie di registrarsi al sistema 2. L'utente inserisce i dati richiesti dal sistema. 3. Il sistema verifica la validità dei dati. 4. Il sistema elabora la richiesta e memorizza i dati inseriti. 5. Il sistema reindirizza l'utente all'interfaccia di login, comunicandogli l'avvenuta registrazione
Estensioni		<p>Se l'utente è già registrato</p> <p>3a. Il sistema risponde con un messaggio d'errore</p> <p>3b. L'utente torna alla schermata di registrazione</p> <p>Se il formato dei dati inseriti non è valido o non vengono inseriti tutti i dati richiesti</p> <p>3c. Il sistema risponde con un messaggio d'errore</p>
Requisiti particolari		Il sistema deve essere in grado di gestire la registrazione sia per un cliente che per un gestore.
Tecnologie e variazioni dei dati		La registrazione dell'utente "Cliente" viene riconosciuta tramite il numero della patente.
Frequenza di occorrenza		Potenzialmente continuo
Varie		

3.4.3 CASO D'USO 3 : CREA PARCO AUTO

Nome del caso d'uso		Crea Parco Auto
Ambito		Applicazione RentApp
Livello		Obiettivo Gestore
Attore primario		Gestore
Parti interessate		Il gestore vuole creare un nuovo parco auto.
Precondizioni		Il gestore deve essere correttamente autenticato all'interno del sistema
Garanzia di successo		Il nuovo parco auto del gestore viene correttamente aggiunto alla lista dei parchi auto del sistema. Il sistema mostra un messaggio di conferma.
Principale scenario di successo		<ol style="list-style-type: none"> 1. Il gestore sceglie di creare un nuovo parco auto 2. Il gestore inserisce i dati richiesti dal sistema. 3. Il sistema assegna un id sequenziale ai parchi auto, elabora la richiesta e memorizza i dati inseriti. 4. Il sistema mostra un messaggio di conferma e reindirizza alla dashboard il gestore.
Estensioni		Il gestore non inserisce tutti i campi richiesti 2.a Il sistema mostra un messaggio di errore
Requisiti particolari		Il sistema deve essere in grado sempre di gestire la creazione del parco auto in maniera affidabile.
Tecnologie e variazioni dei dati		
Frequenza di occorrenza		Occasionale
Varie		

3.4.4 CASO D'USO 4 : INSERIMENTO NUOVO VEICOLO

Nome del caso d'uso	Inserimento nuovo veicolo
Ambito	Applicazione RentApp
Livello	Obiettivo gestore
Attore primario	Gestore
Parti interessate	Il gestore vuole inserire un nuovo veicolo all'interno del parco auto.
Precondizioni	Il gestore deve essere autenticato all'interno del sistema
Garanzia di successo	Il nuovo veicolo viene correttamente aggiunto alla lista delle auto del parco auto del gestore.
Principale scenario di successo	<ol style="list-style-type: none"> 1. Il gestore sceglie di inserire un nuovo veicolo 2. Il gestore inserisce i dati richiesti dal sistema. 3. Il sistema elabora la richiesta e memorizza i dati inseriti. 4. Il sistema reindirizza il gestore alla dashboard e mostra un messaggio di conferma.
Estensioni	<p>Se il gestore inserisci nei dati richiesti una targa già presente nel sistema o non inserisce tutti i campi richiesti</p> <p>3a. Il sistema risponde con un messaggio d'errore</p>
Requisiti particolari	Il sistema deve essere in grado di gestire con l'80 % successo l'inserimento del nuovo veicolo nel parco auto
Tecnologie e variazioni dei dati	
Frequenza di occorrenza	Ricorrente
Varie	

3.4.5 CASO D'USO 5 : VISUALIZZA LISTA PRENOTAZIONI

Nome del caso d'uso	Visualizza lista prenotazioni
Ambito	Applicazione RentApp
Livello	Obiettivo del gestore
Attore primario	Gestore
Parti interessate	Il Gestore vuole visualizzare la lista delle prenotazioni dei propri parchi auto
Precondizioni	Il Gestore deve essere autenticato all'interno del sistema
Garanzia di successo	Vengono mostrate tutte le prenotazioni attive di tutti i parchi auto relativi al gestore
Principale scenario di successo	<ol style="list-style-type: none"> 1. Il gestore sceglie di visualizzare la lista delle prenotazioni dei propri parchi auto 2. Il sistema elabora la richiesta 3. Il sistema mostra al gestore la lista delle prenotazioni dei parchi auto
Estensioni	Il gestore non ha una lista di prenotazioni di un parco auto 3a. Il sistema risponde con una lista vuota
Requisiti particolari	Il sistema deve essere in grado di gestire la visualizzazione delle liste dei parchi auto di un gestore.
Tecnologie e variazioni dei dati	
Frequenza di occorrenza	Potenzialmente continuo
Varie	

3.4.6 CASO D'USO 6 : PRENOTA VEICOLO

Nome del caso d'uso		Prenota veicolo
Ambito		Applicazione RentApp
Livello		Obiettivo cliente
Attore primario		Cliente
Parti interessate		Cliente vuole prenotare un veicolo Servizio esterno di pagamento
Precondizioni		Il cliente deve essere autenticato all'interno del sistema e deve aver effettuato una ricerca in base alle proprie esigenze.
Garanzia di successo		La prenotazione del veicolo viene effettuata con successo.
Principale scenario di successo		<ol style="list-style-type: none"> 1. L'utente sceglie di prenotare un veicolo 2. Il sistema chiede il servizio di pagamento ad un sistema esterno. 3. Il sistema crea e salva la prenotazione del cliente. 4. Il sistema genera un resoconto di prenotazione da mostrare all'utente.
Estensioni		<p>Se il veicolo che intende prenotare non è più disponibile</p> <p>1.a Viene generato un messaggio di errore. 1.b l'utente viene reindirizzato alla pagina preposta alla ricerca.</p> <p>Se il pagamento non va a buon fine</p> <p>2.a viene mostrato un messaggio di errore</p>
Requisiti particolari		Il sistema deve essere in grado di gestire la prenotazione di un veicolo da parte di un cliente e interfacciarsi con il sistema esterno di pagamento.
Tecnologie e variazioni dei dati		
Frequenza di occorrenza		Frequente
Varie		

3.4.7 CASO D'USO 7 : RICERCA VEICOLO

Nome del caso d'uso		Ricerca Veicolo
Ambito		Applicazione RentApp
Livello		Obiettivo cliente
Attore primario		Cliente
Parti interessate		Cliente vuole ricercare un veicolo da prenotare
Precondizioni		Il cliente deve essere autenticato all'interno del sistema.
Garanzia di successo		Viene mostrata una lista di veicoli disponibili
Principale scenario di successo		<ol style="list-style-type: none"> 1. L'utente sceglie di prenotare un veicolo 2. L'utente inserisce i dati relativi alla ricerca richiesti dal sistema 3. Il sistema elabora la richiesta e restituisce una lista di veicoli disponibili per la prenotazione
Estensioni		<p>Se la ricerca non produce alcun risultato</p> <p>3.a Viene mostrato un messaggio di avvertimento</p>
Requisiti particolari		Il sistema deve essere in grado di gestire la ricerca da parte di un cliente restituendo tutti i veicoli disponibili per il luogo e periodo selezionato.
Tecnologie e variazioni dei dati		
Frequenza di occorrenza		Frequente
Varie		

3.4.8 CASO D'USO 8 : VISUALIZZA LISTA PRENOTAZIONI CLIENTE

Nome del caso d'uso	Visualizza lista prenotazioni Cliente
Ambito	Applicazione RentApp
Livello	Obiettivo del Cliente
Attore primario	Cliente
Parti interessate	Il Cliente vuole visualizzare lo storico delle proprie prenotazioni
Precondizioni	Il Cliente deve essere autenticato all'interno del sistema
Garanzia di successo	Vengono mostrate tutte le prenotazioni relative al Cliente che invoca la funzionalità di sistema
Principale scenario di successo	<ol style="list-style-type: none">1. Il Cliente sceglie di visualizzare lo storico delle proprie prenotazioni2. Il sistema elabora la richiesta3. Il sistema mostra al cliente la lista delle prenotazioni
Estensioni	Il cliente non ha una lista di prenotazioni 3a. Il sistema risponde con una lista vuota
Requisiti particolari	Il sistema deve essere in grado di riconoscere l'utente che invoca la funzione restituendo solo le prenotazioni relative a quell'utente.
Tecnologie e variazioni dei dati	
Frequenza di occorrenza	Potenzialmente continuo
Varie	

3.4.9 CASO D'USO 9 : LOGOUT

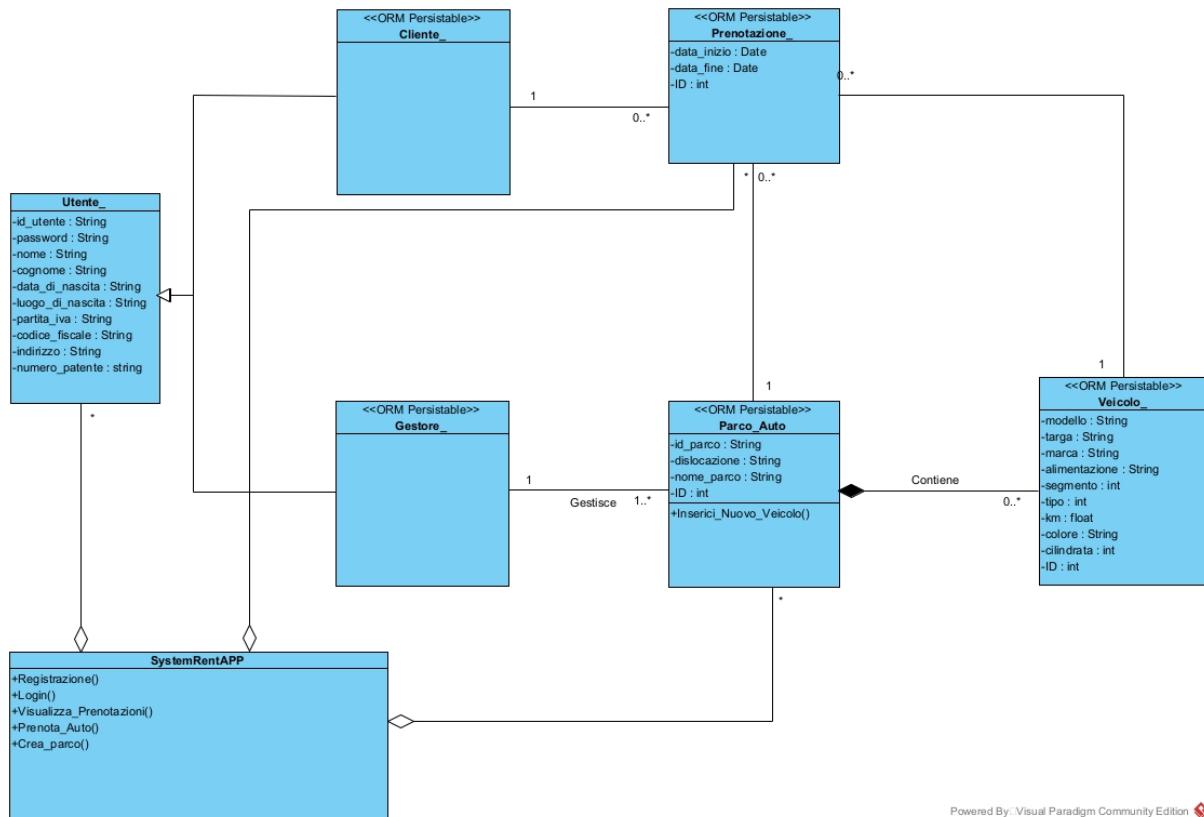
Nome del caso d'uso	Effettua Logout
Ambito	Applicazione RentApp
Livello	Obiettivo utente
Attore primario	Utente
Parti interessate	Utente vuole disconnettersi dal sistema.
Precondizioni	L'utente deve essere registrato all'interno del sistema e correttamente loggato.
Garanzia di successo	Il sistema mostra la corretta interfaccia
Principale scenario di successo	<ol style="list-style-type: none">1. L'utente Preme su Disconnetti all'interno della dashboard2. Il sistema rimuove i token dal database e restituisce un messaggio di conferma3. L'utente viene rimandato al login
Estensioni	Se l'utente non è loggato il sistema ignora la richiesta
Requisiti particolari	Il sistema deve essere in grado sempre di cancellare i token con successo se presenti.
Tecnologie e variazioni dei dati	
Frequenza di occorrenza	Potenzialmente continuo
Varie	

3.5 TABELLA OBIETTIVI ATTORI

Attore	OBIETTIVO
<i>Utente non Registrato</i>	Registrati all'interno del sistema
<i>Utente</i>	Esegui l'autenticazione al sistema (Login) Esegui la disconnessione dal sistema (Logout)
<i>Gestore</i>	Crea Parco Auto Inserisci nuovo Veicolo in Parco Auto Visualizza lista Prenotazioni
<i>Cliente</i>	Ricerca Veicolo Prenota Veicolo Visualizza il proprio storico Prenotazioni
<i>Sistema Esterno di Pagamento</i>	Gestisci la Transazione di Pagamento

3.6 SYSTEM DOMAIN MODEL

Al fine di mettere a fuoco i concetti fondamentali di un sistema e definire un vocabolario specifico per il sistema è stato sviluppato un *System Domain Model* o **SDM**. Tale modello fornisce a tutti i membri del team una base comune di concetti su cui ragionare e una terminologia condivisa rigorosa e specifica dando una descrizione statica del sistema.

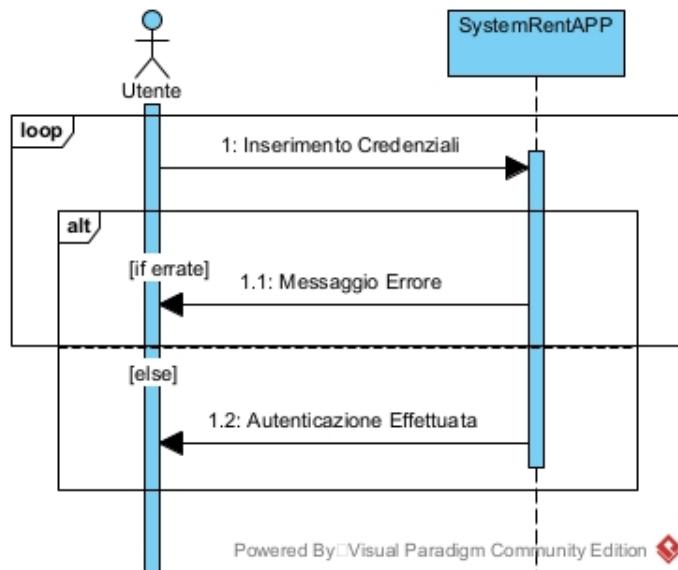


3.7 SEQUENCE DIAGRAM DI ANALISI

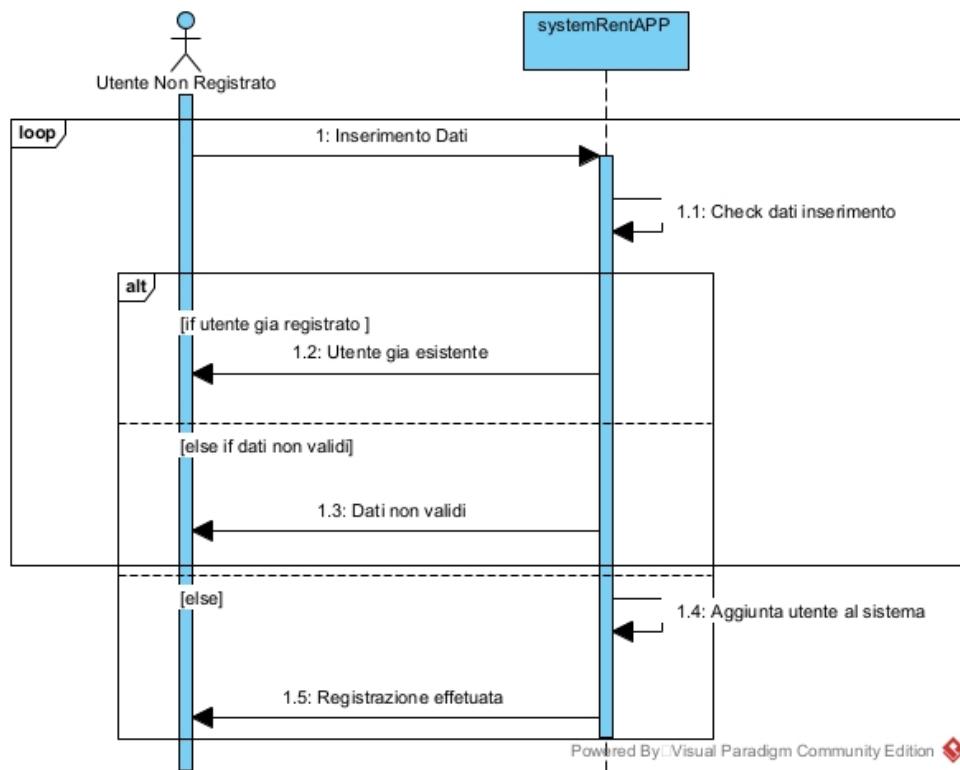
I Sequence Diagram enfatizzano la temporizzazione e l'ordine dei messaggi scambiati tra le istanze del sistema, cioè mettono in luce una determinata sequenza di azioni in cui tutte le scelte sono state già effettuate.

Di seguito i Sequence Diagram di Analisi delle funzionalità identificate per il sistema.

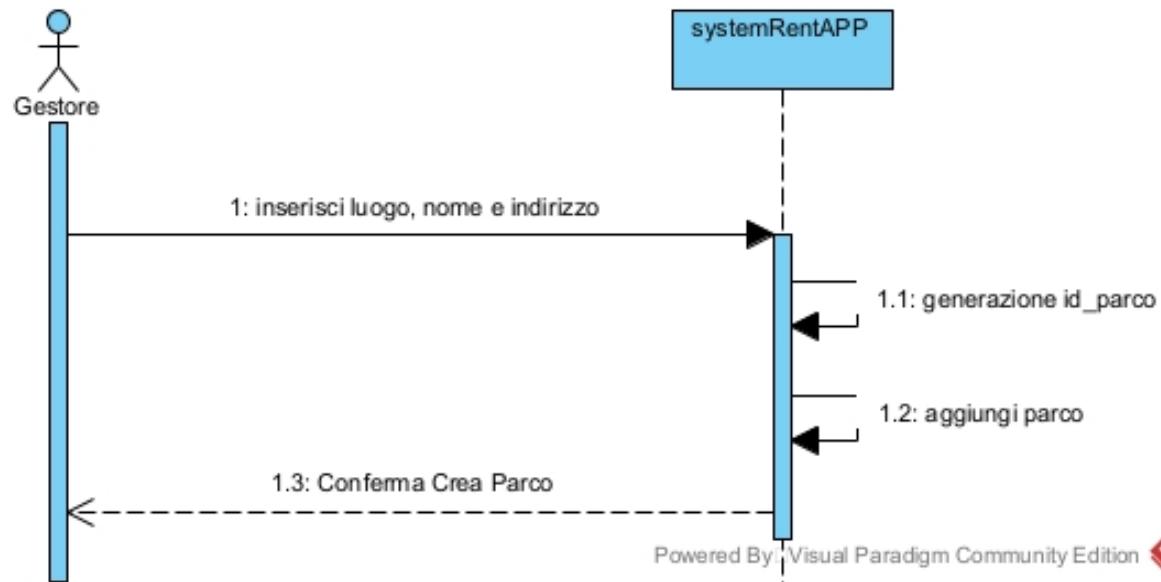
3.7.1 EFFETTUARE LOGIN - SEQUENCE DIAGRAM DI ANALISI



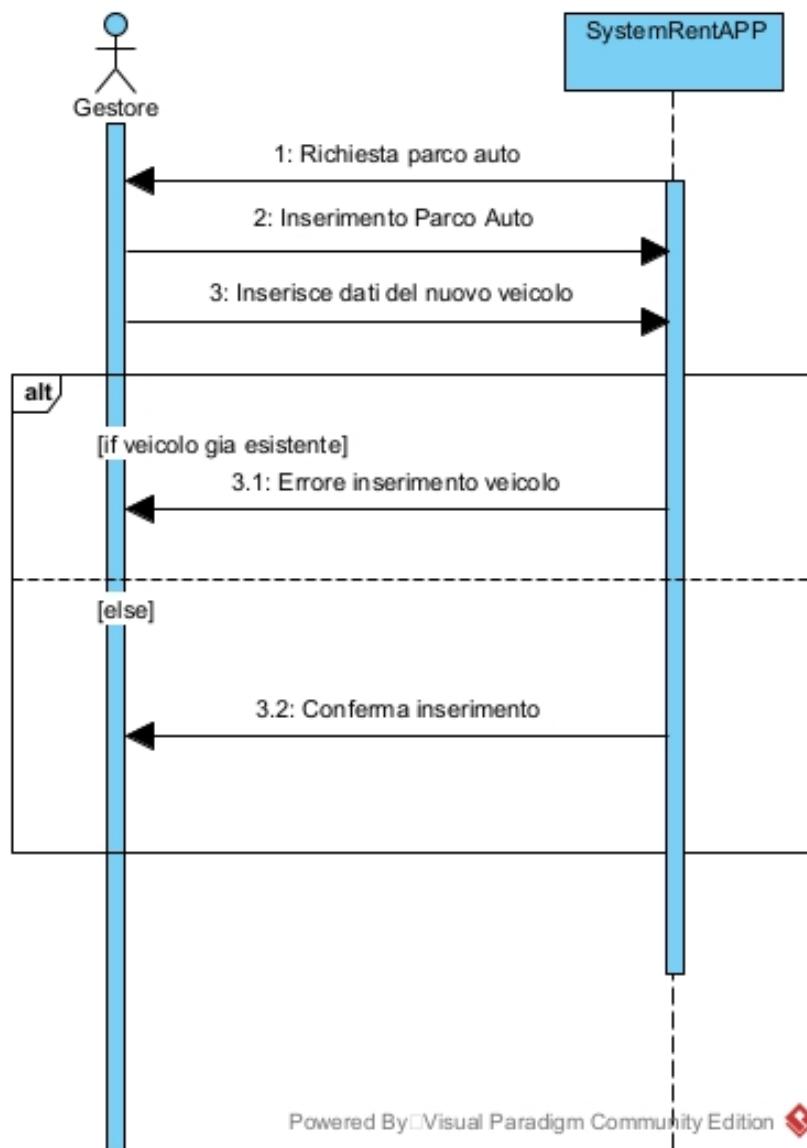
3.7.2 REGISTRAZIONE - SEQUENCE DIAGRAM DI ANALISI



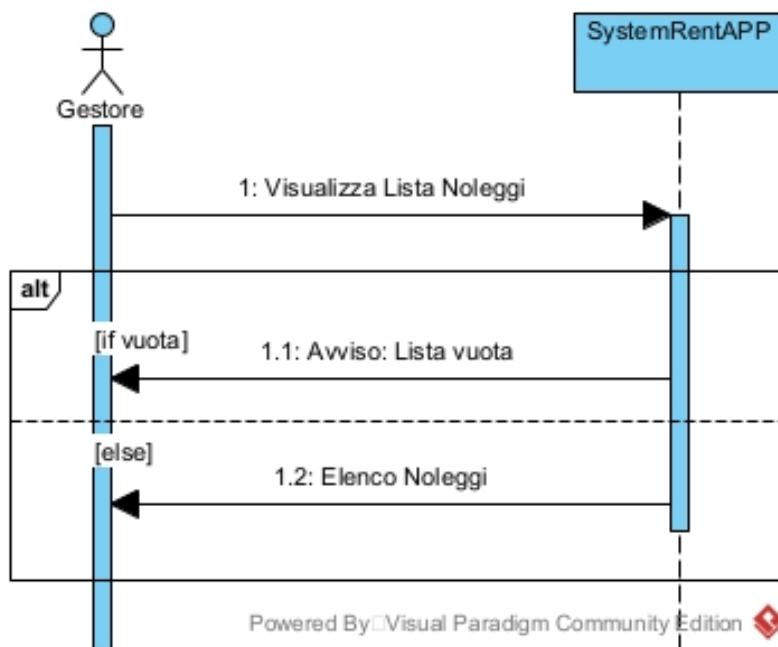
3.7.3 CREA PARCO AUTO - SEQUENCE DIAGRAM DI ANALISI



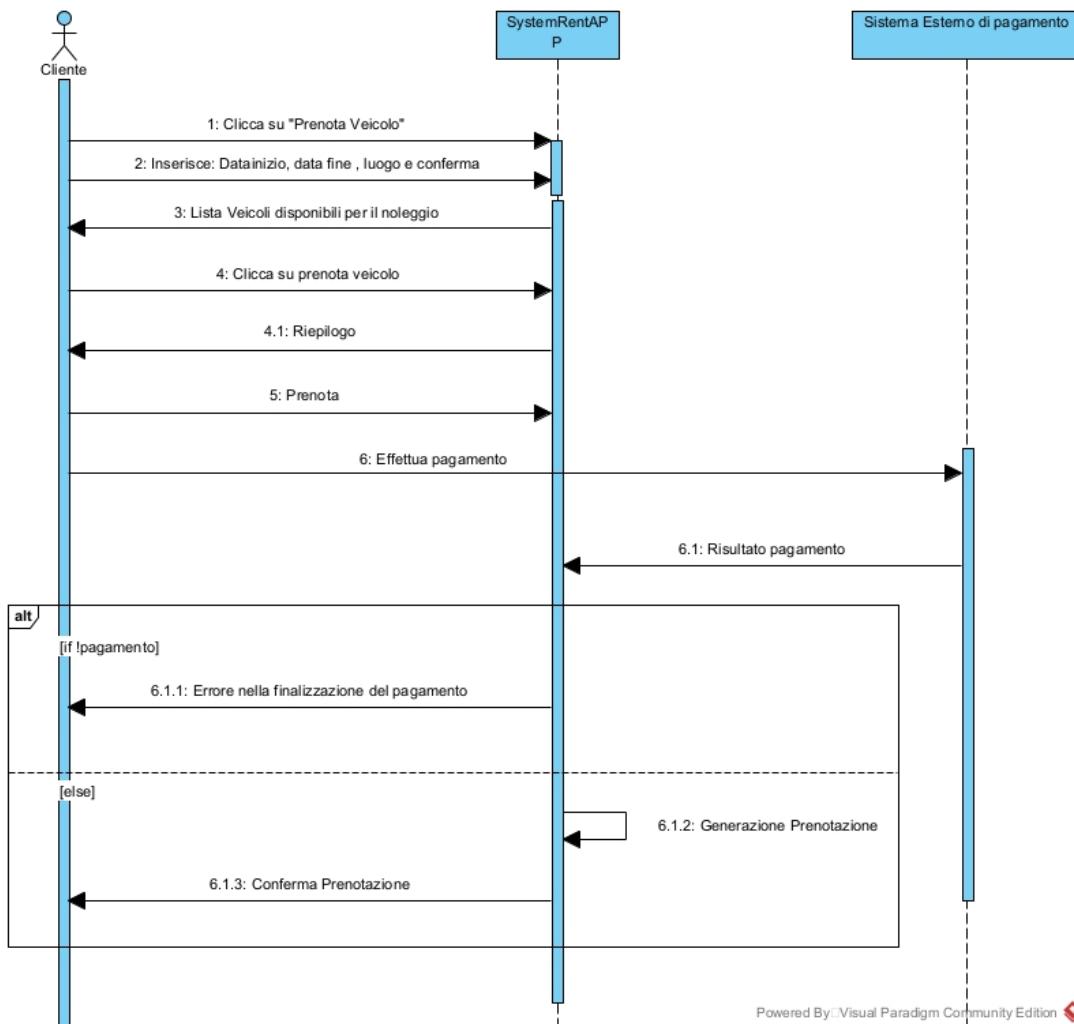
3.7.4 INSERISCI NUOVO VEICOLO - SEQUENCE DIAGRAM DI ANALISI



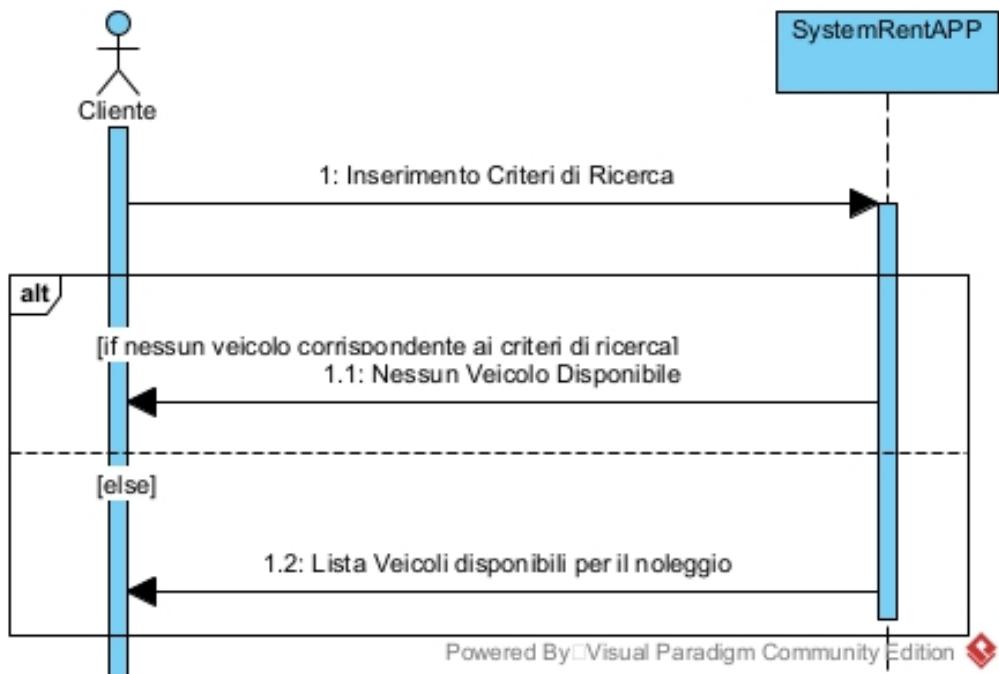
3.7.5 VISUALIZZA LISTA PRENOTAZIONI - SEQUENCE DIAGRAM DI ANALISI



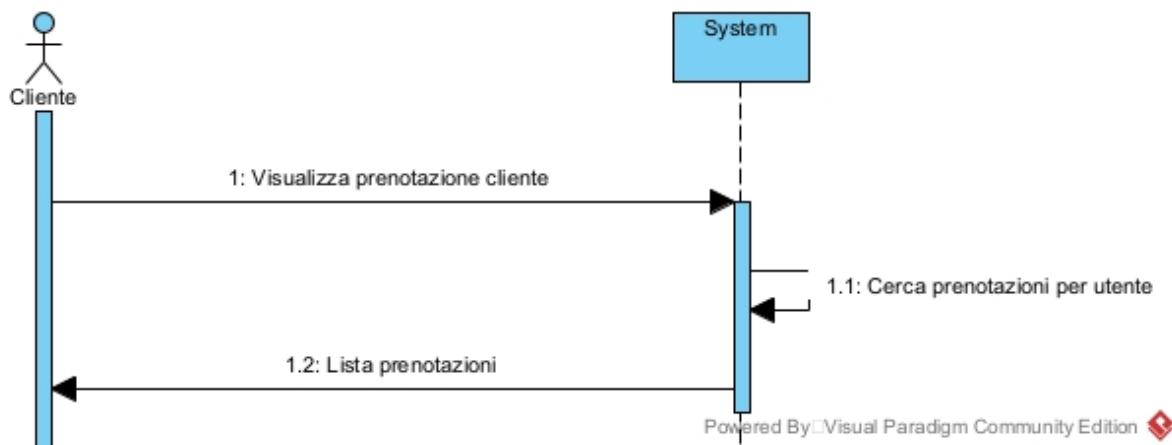
3.7.6 PRENOTA VEICOLO - SEQUENCE DIAGRAM DI ANALISI



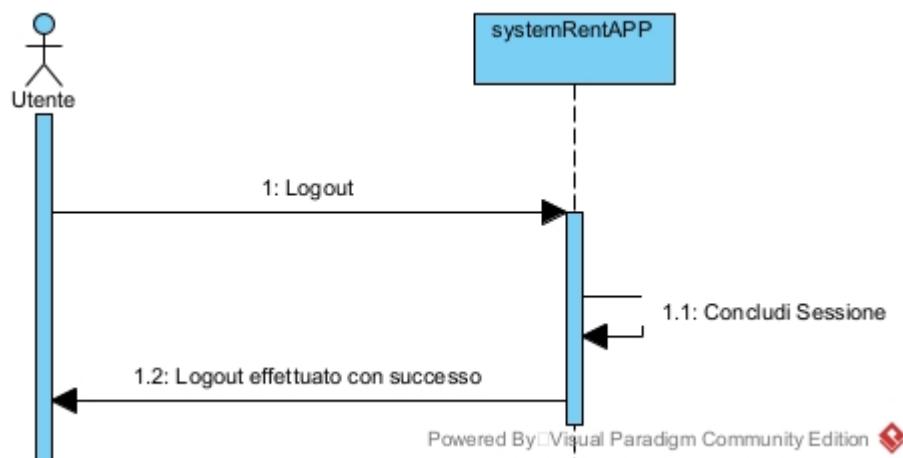
3.7.7 RICERCA VEICOLO - SEQUENCE DIAGRAM DI ANALISI



3.7.8 VISUALIZZA LISTA PRENOTAZIONI CLIENTE - SEQUENCE DIAGRAM DI ANALISI



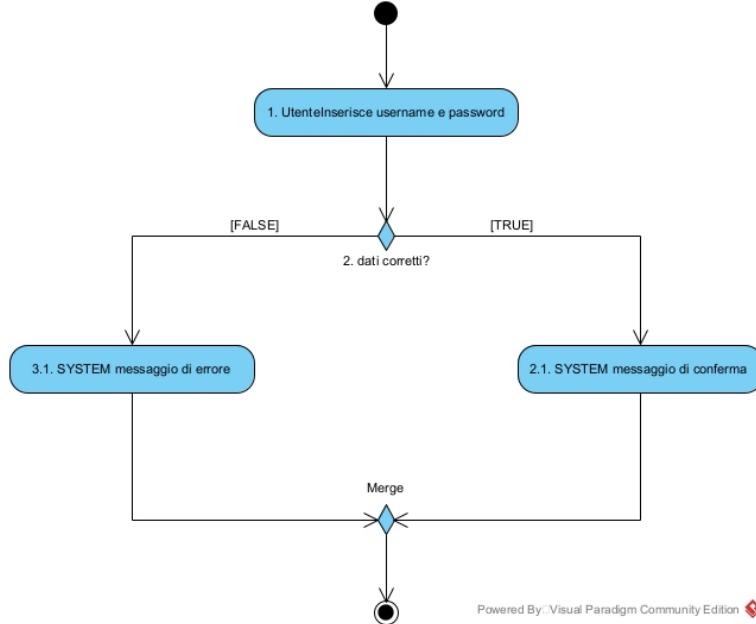
3.7.9 LOGOUT- SEQUENCE DIAGRAM DI ANALISI



3.8 ACTIVITY DIAGRAM DI ANALISI

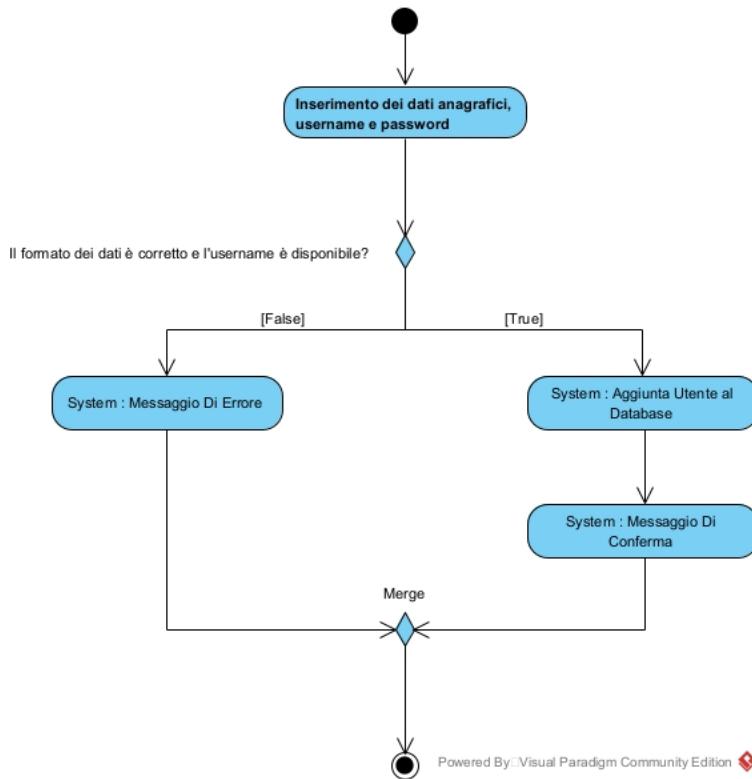
Gli Activity Diagram sono un tipo di diagramma che permette di descrivere un processo attraverso dei grafi in cui i nodi rappresentano le attività e gli archi l'ordine con cui vengono eseguite.

3.8.1 EFFETTUA LOGIN - ACTIVITY DIAGRAM DI ANALISI



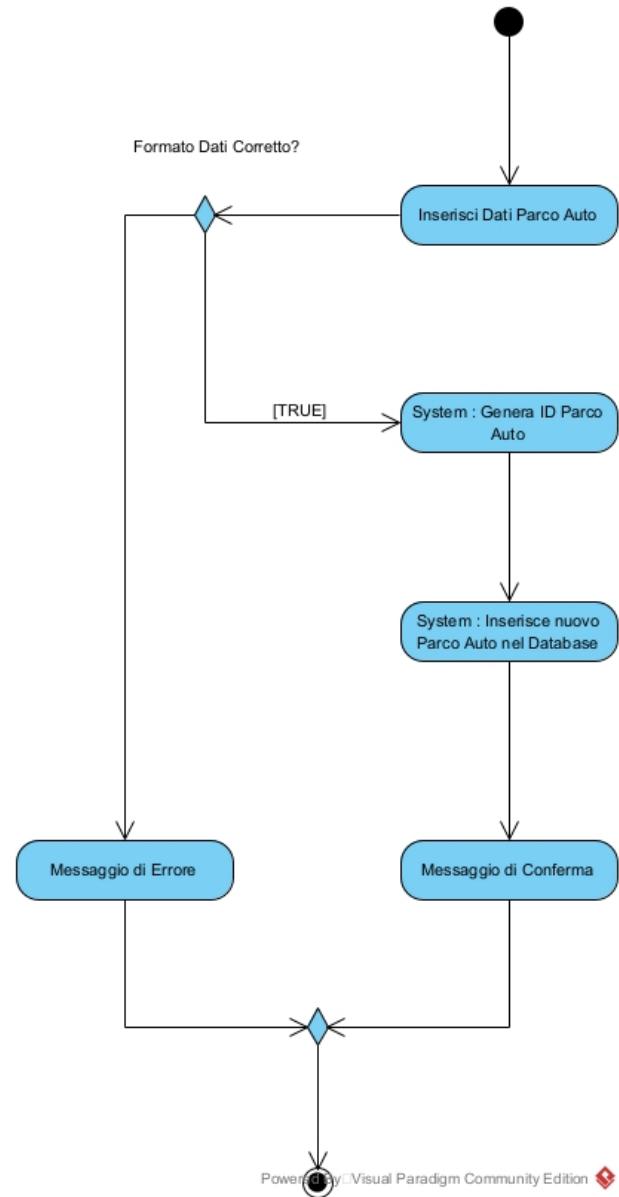
Powered By: Visual Paradigm Community Edition

3.8.2 REGISTRAZIONE - ACTIVITY DIAGRAM DI ANALISI

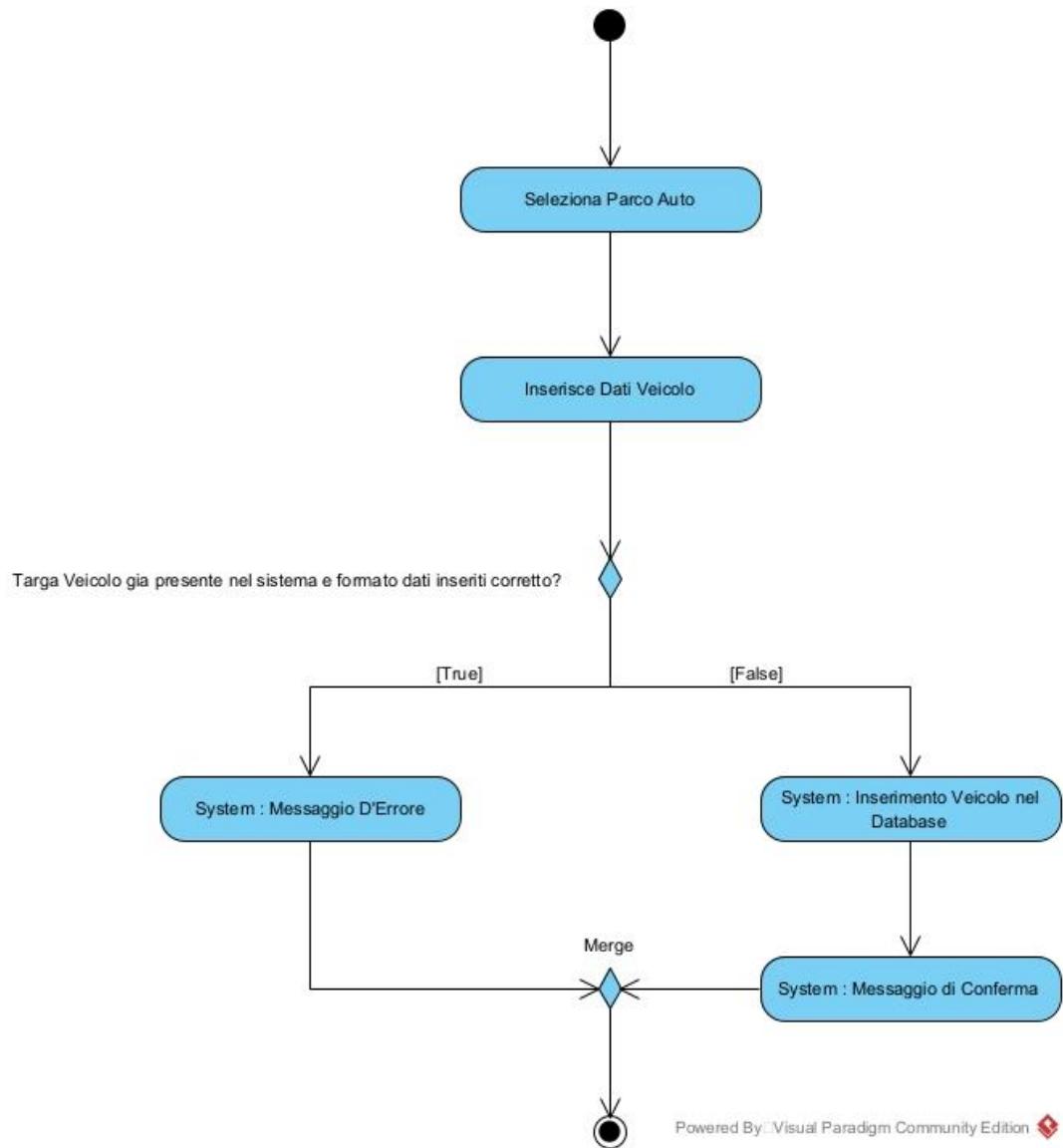


Powered By: Visual Paradigm Community Edition

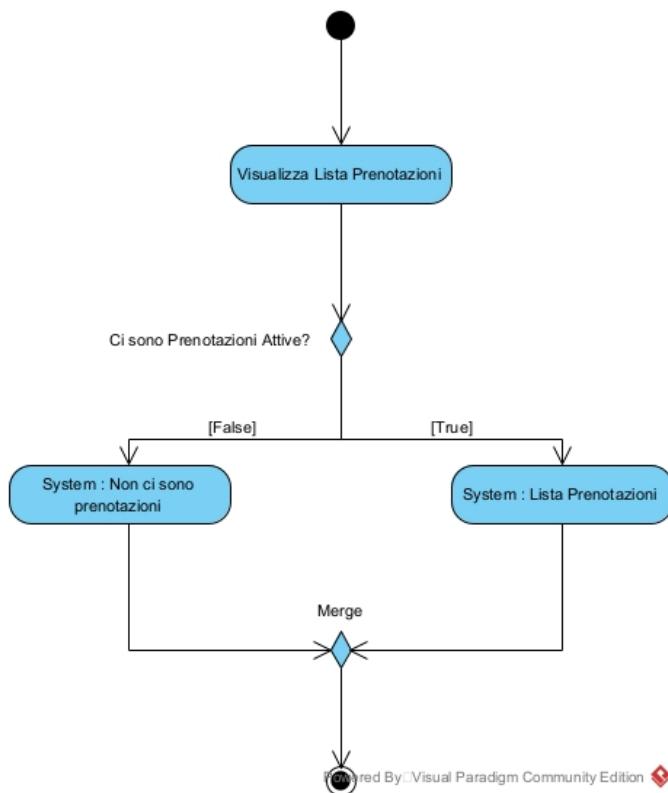
3.8.3 CREA PARCO AUTO - ACTIVITY DIAGRAM DI ANALISI



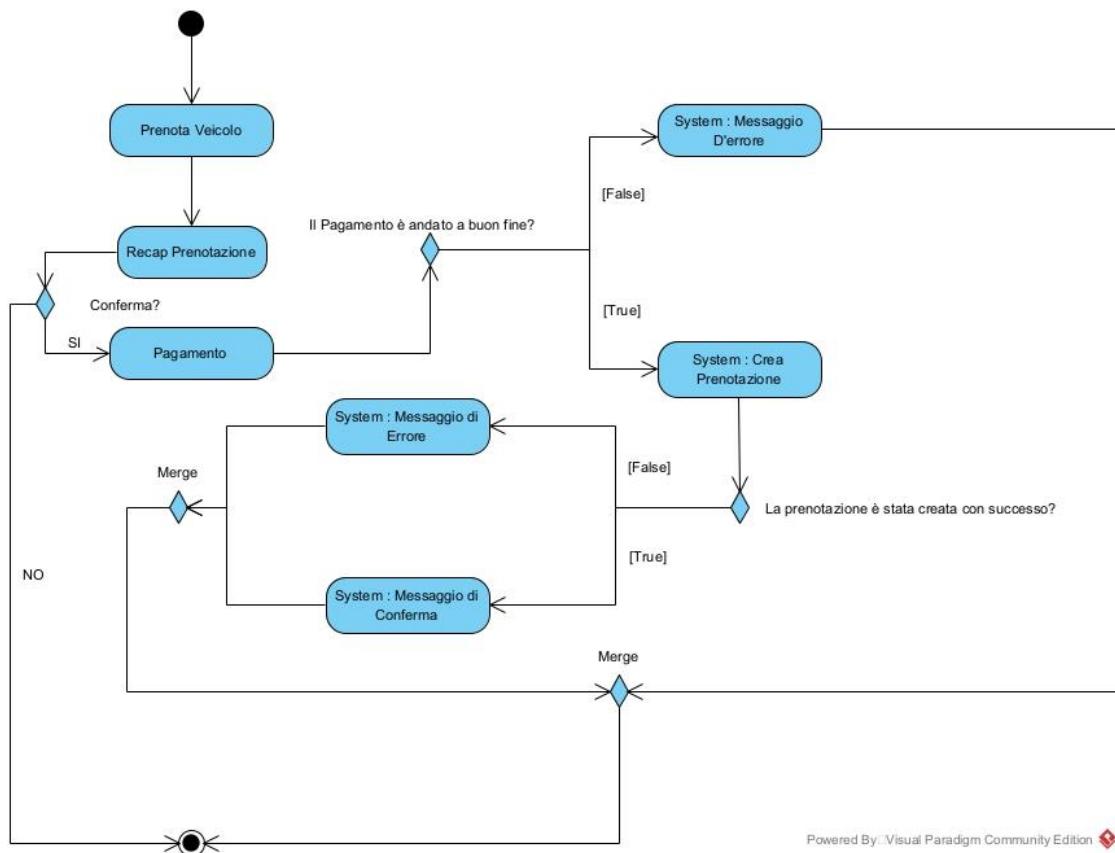
3.8.4 INSERISCI NUOVO VEICOLO - ACTIVITY DIAGRAM DI ANALISI



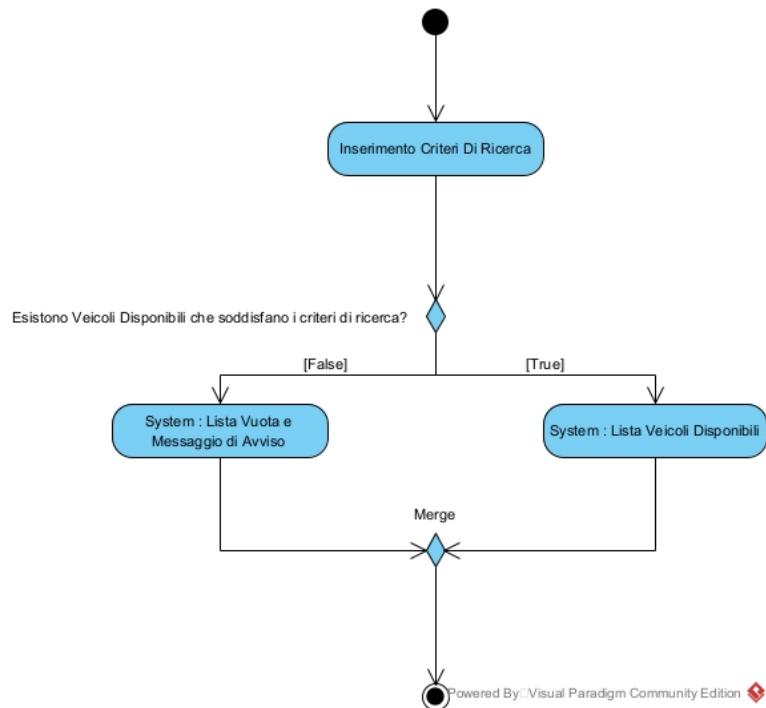
3.8.5 VISUALIZZA LISTA PRENOTAZIONI - ACTIVITY DIAGRAM DI ANALISI



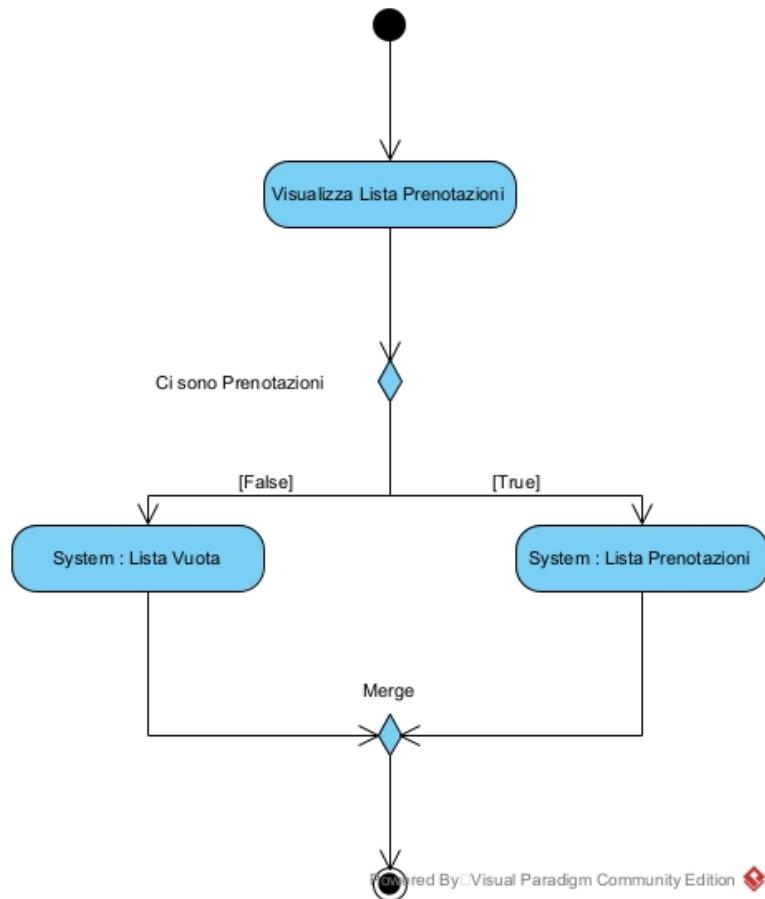
3.8.6 PRENOTA VEICOLO - ACTIVITY DIAGRAM DI ANALISI



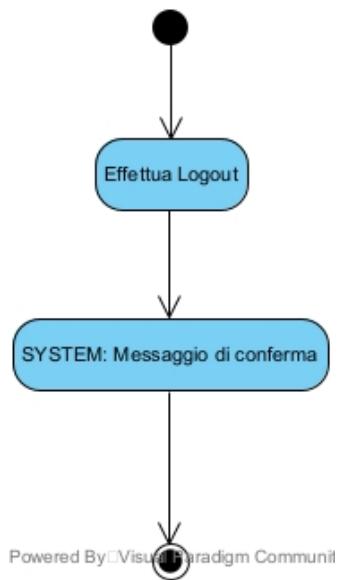
3.8.7 RICERCA VEICOLO - ACTIVITY DIAGRAM DI ANALISI



3.8.8 VISUALIZZA LISTA PRENOTAZIONI CLIENTE - ACTIVITY DIAGRAM DI ANALISI



3.8.9 LOGOUT - ACTIVITY DIAGRAM DI ANALISI



CAPITOLO 4 : ARCHITETTURA SOFTWARE

All'interno di un sistema software è importante avere una vista di come sarà l'architettura di quest'ultimo. In particolare, una architettura software è definita dai suoi componenti, dalle relazioni reciproche tra i componenti e con l'ambiente e dai principi che ne governano la progettazione e l'evoluzione.

Infatti, descrivere l'architettura software di un sistema significa elencarne le sotto parti costituenti ed illustrarne i rapporti inter-funzionali.

Per la descrizione dell'architettura utilizzata ci si è avvalso di uno stile architettonico di tipo Model View Controller o **MVC**. L'MVC è uno dei pattern più utilizzato sia nello sviluppo web che in applicazioni java. L'adozione di tale pattern architettonico è stata dettata dalla necessità della nostra web app di visualizzare dati generici tramite l'utilizzo di rappresentazioni diverse per lo stesso dato. Si pensi ad esempio alle entità "prenotazioni" visualizzate secondo una logica differente dalle due tipologie di utenti che utilizzano il software.

L'MVC consente la realizzazione di un'architettura che permette la separazione netta dei componenti di presentazione dei dati dai componenti che gestiscono i dati stessi.

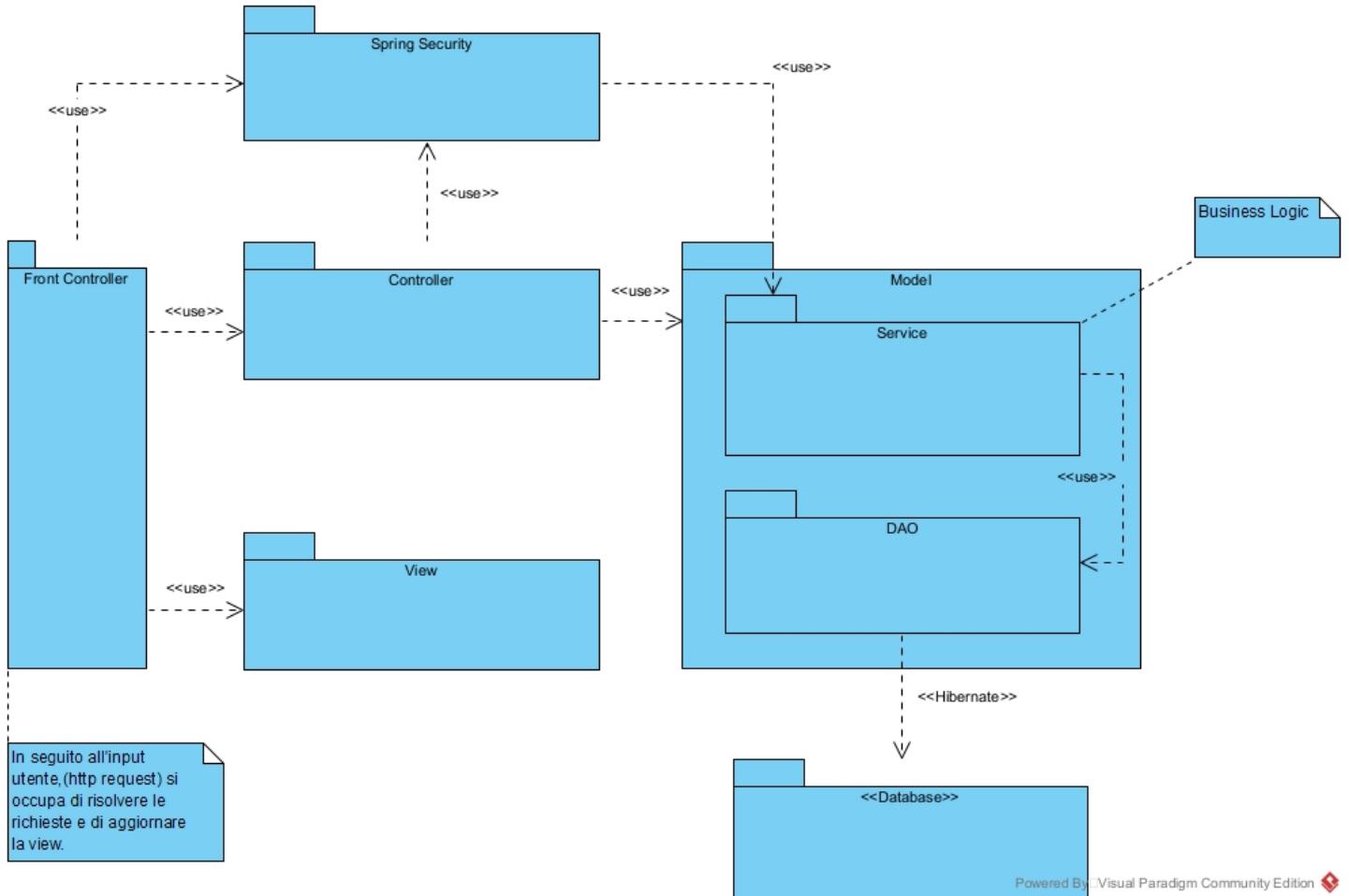
L'architettura in una sua versione basilare prevede l'utilizzo di tre principali componenti : Model View e Controller.

Per lo sviluppo della nostra App, è stato scelto di utilizzare una variante del suddetto modello chiamata **Spring MVC** accessoriata da una componente aggiuntiva chiamata **Spring Security**. Per maggiori dettagli su questo pattern architettonico si rimanda al capitolo trattante lo sviluppo in cui si è fatta trattazione del framework.

In particolare, Spring MVC per l'aggiornamento delle viste visualizzate dall'utente si basa su un processo sincrono in cui utilizza una strategia di "push" con la quale nelle notifiche invia anche i dati da visualizzare al fine di evitare l'utilizzo del pattern observer per l'aggiornamento asincrono delle viste.

Di seguito le viste di alto livello dell'architettura del sistema RentApp.

4.1 VISTA ARCHITETTURALE DI ALTO LIVELLO



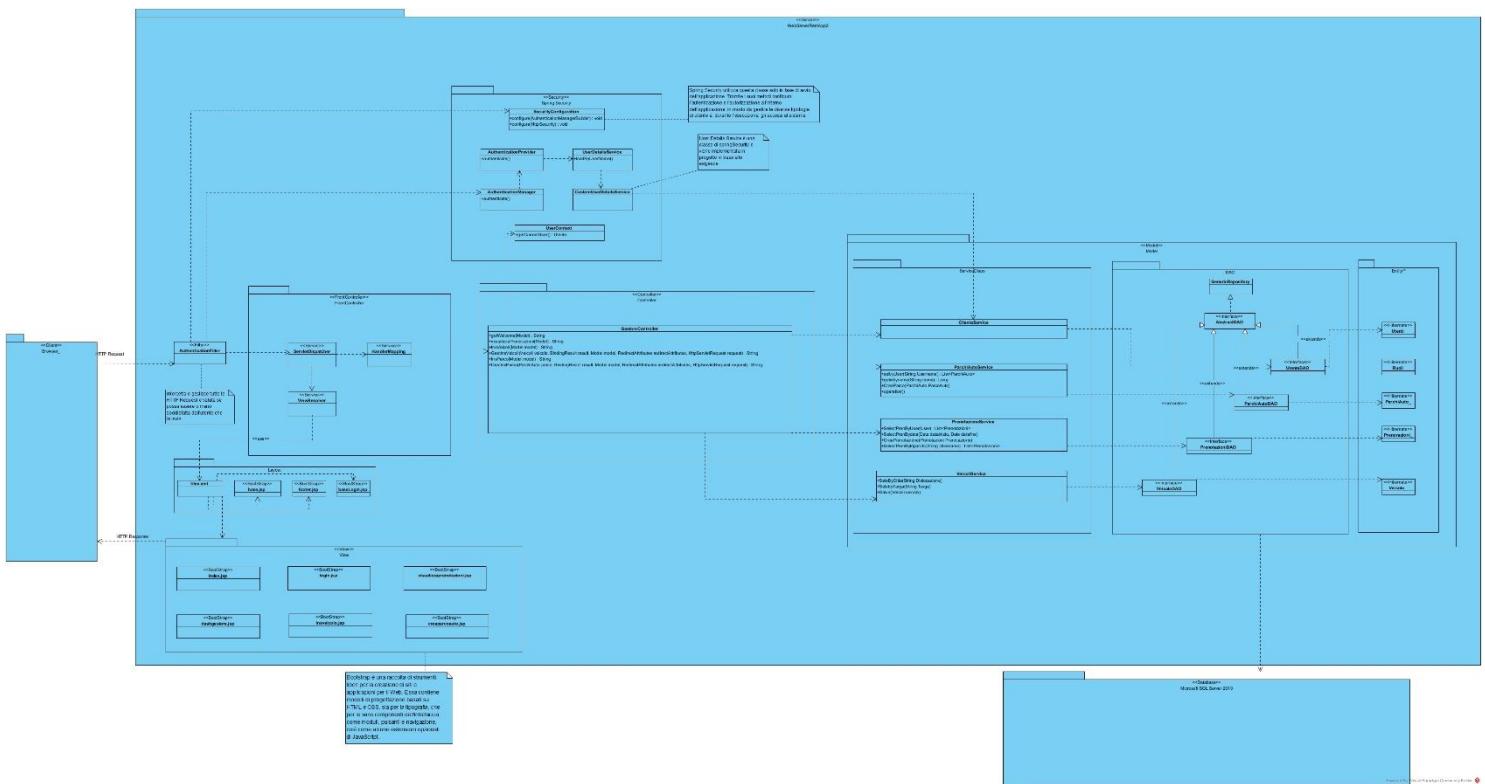
4.2 VISTE ARCHITETTURALE DI DETTAGLIO

Per quanto riguarda la specializzazione del modello precedente , per una questione di leggibilità il team ha deciso di sviluppare due viste architetturali per il gestore e per il cliente precisando che entrambe le viste proposte sono delle prospettive differenti della medesima vista architetturale.

Per una visione migliore dei due diagrammi inseriti si rimanda al file ".vpp" allegato al documento.

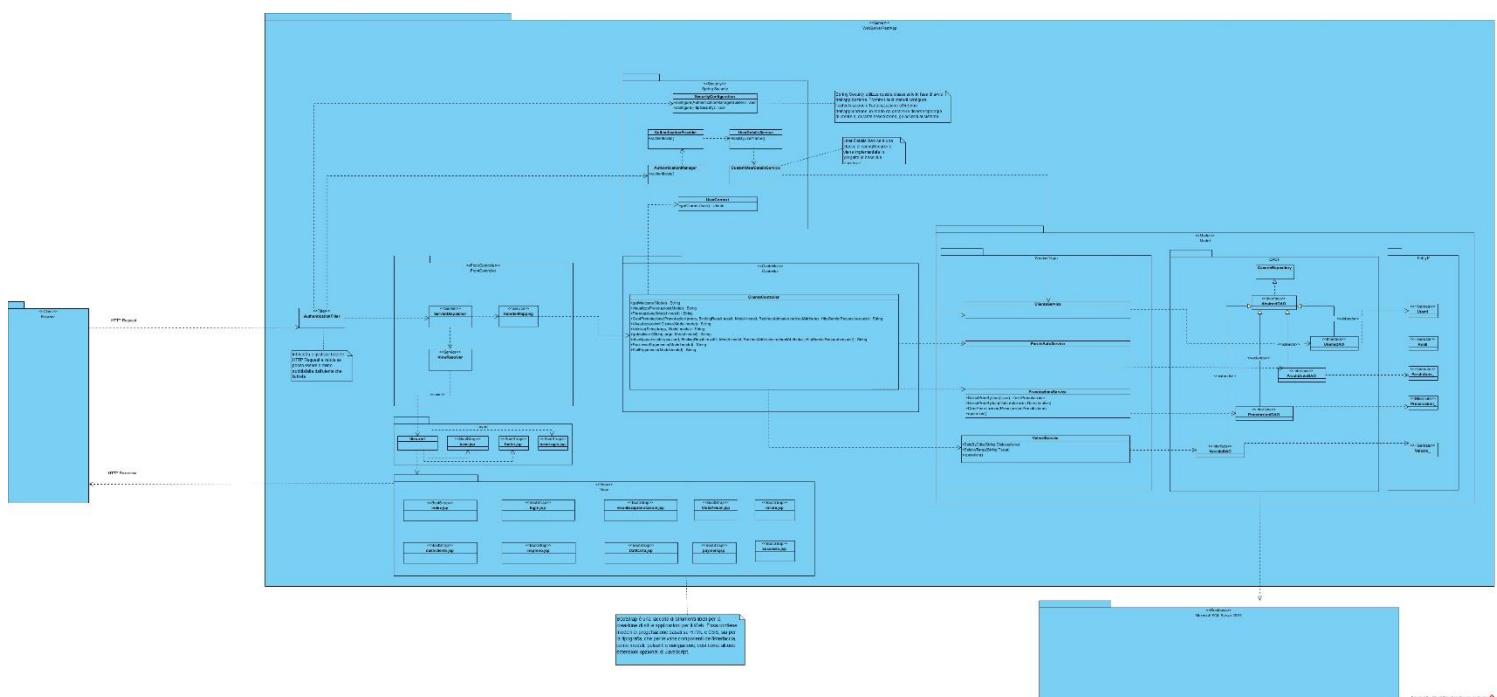
4.2.1 VISTA ARCHITETTURALE DI DETTAGLIO (GESTORE)

Nella vista sono stati inserite solo le funzionalità richieste dall'attore Gestore.



4.2.2 VISTA ARCHITETTURALE DI DETTAGLIO (CLIENTE)

Nella vista sono stati inserite solo le funzionalità richieste dall'attore Cliente.

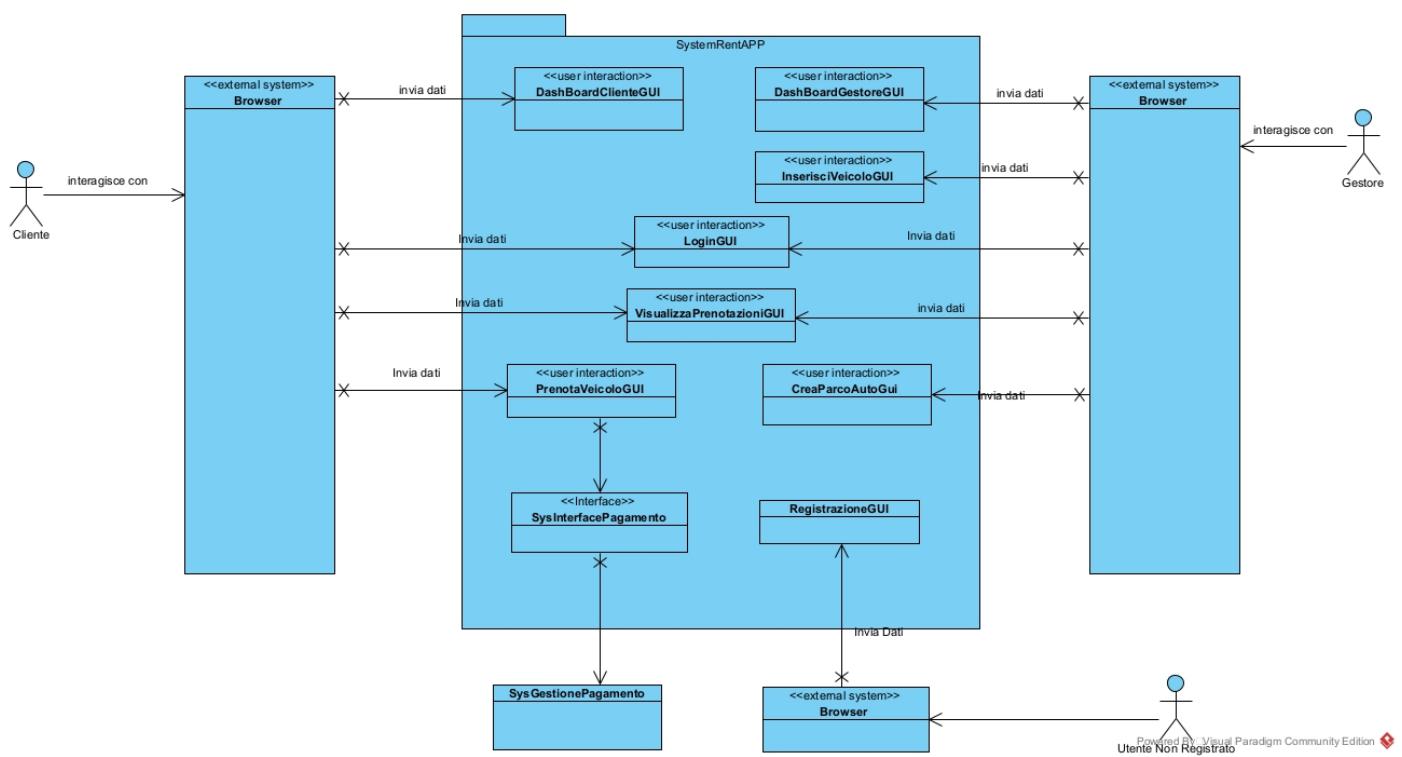


4.3 DIAGRAMMA DI CONTESTO

Lo scopo principale del diagramma di contesto è quello di mostrare il contesto di una View.

Il contesto viene rappresentato dall'ambiente con cui il sistema o una sua specifica parte interagisce e le entità del contesto possono essere rappresentate da persone o oggetti fisici.

Di seguito è mostrato il diagramma di contesto con il quale vengono identificate le viste del sistema che permettono a ciascuna classe di utenti di interagire con il sistema.

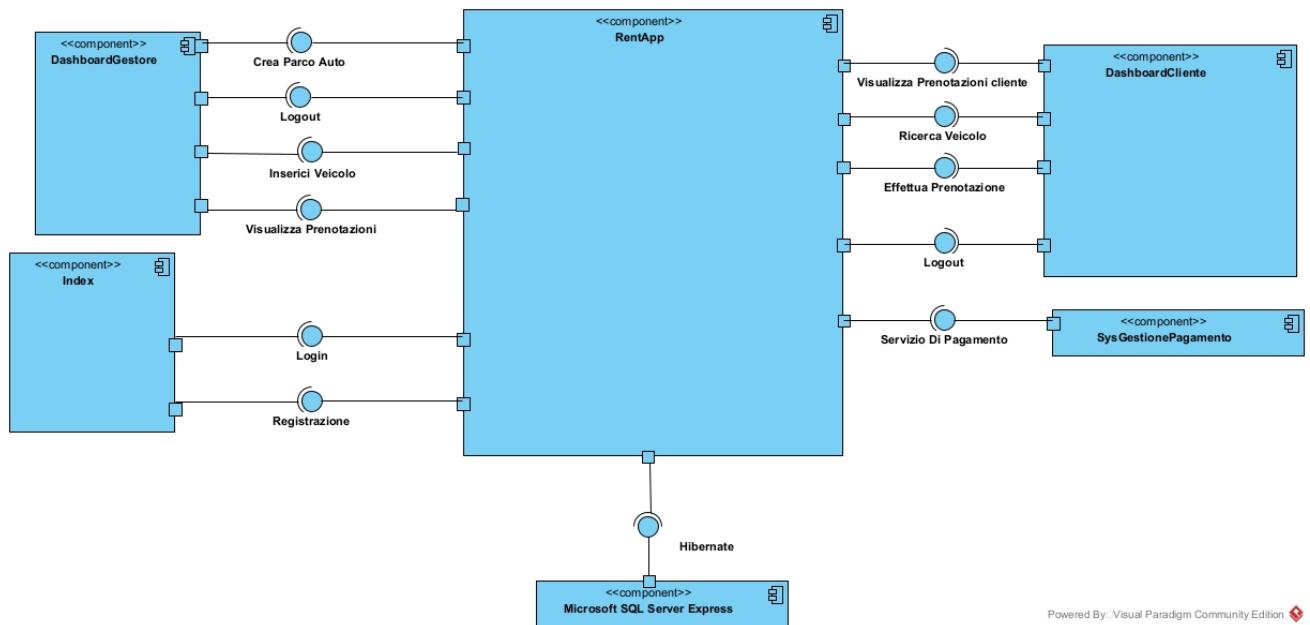


4.4 DIAGRAMMA DELLE COMPONENTI

RentApp si presenta come una componente software alla quale altre componenti possono richiedere delle funzionalità se compatibili, cioè se offrono e richiedono le interfacce alla web app.

Nel dettaglio distinguiamo queste componenti come:

- Una componente che permette di gestire la persistenza dei dati del sistema (Microsoft SQL Server 2019 Express)
- Tre componenti, una per ogni tipologia di attore e una condivisa tra “Utente” e “Utente non registrato”, con le relative interfacce con il sistema.
- Una componente per il sistema esterno di pagamento alla quale RentApp chiede un’interfaccia di pagamento.



4.5 SEQUENCE DIAGRAM DI DETTAGLIO

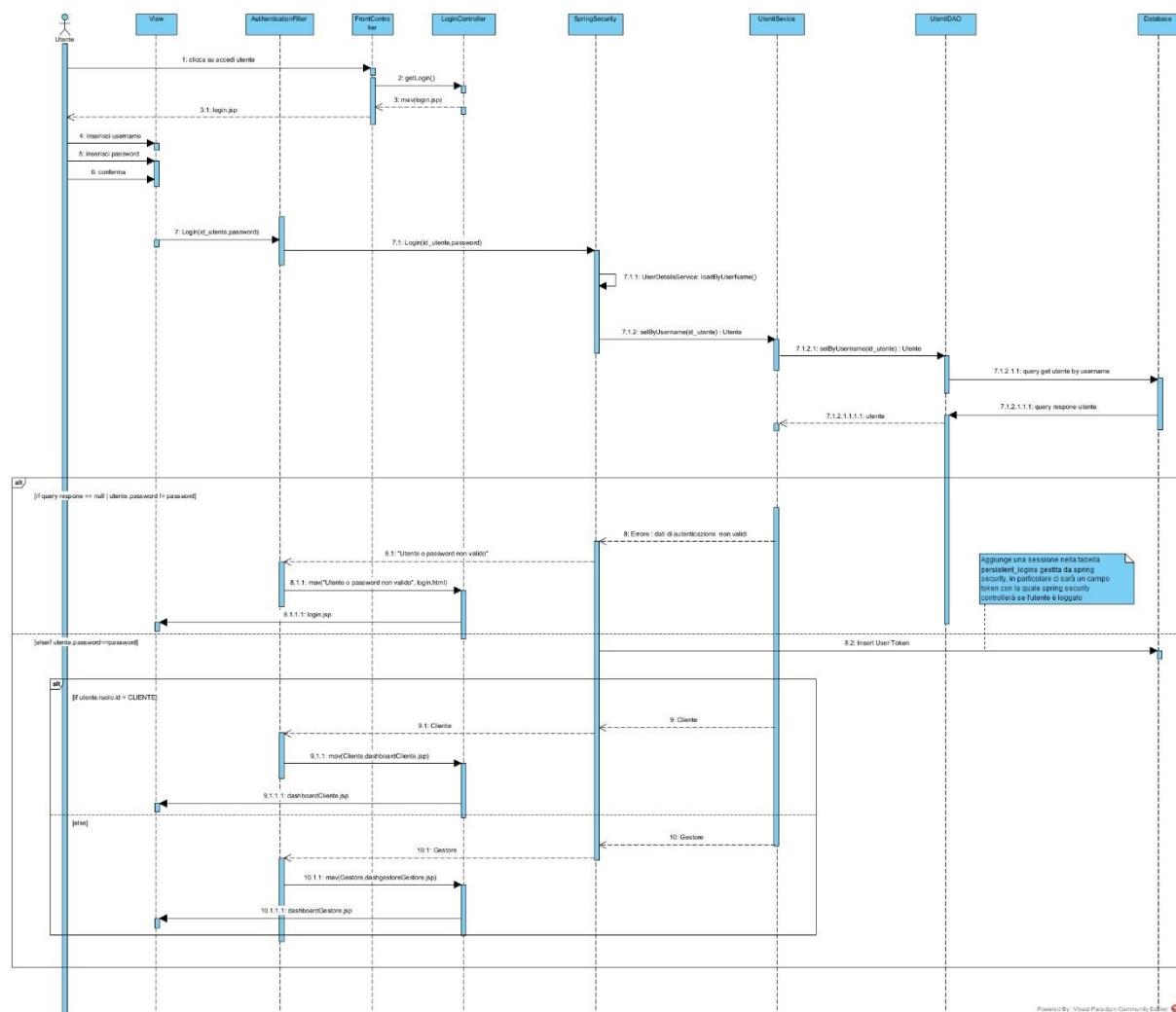
Un diagramma di sequenza di dettaglio può essere usato per definire gli input e gli output del sistema da realizzare. Esso rappresenta un particolare scenario di caso d'uso, dove per scenario si intende una determinata sequenza di azioni in cui tutte le scelte sono già state effettuate.

Il diagramma di sequenza di dettaglio descrive le relazioni che intercorrono in termini di messaggi tra Attori, Oggetti di business, Oggetti o Entità del sistema che si sta rappresentando. In UP questi diagrammi vanno associati solo ai casi d'uso e agli scenari che sono oggetto della specifica iterazione.

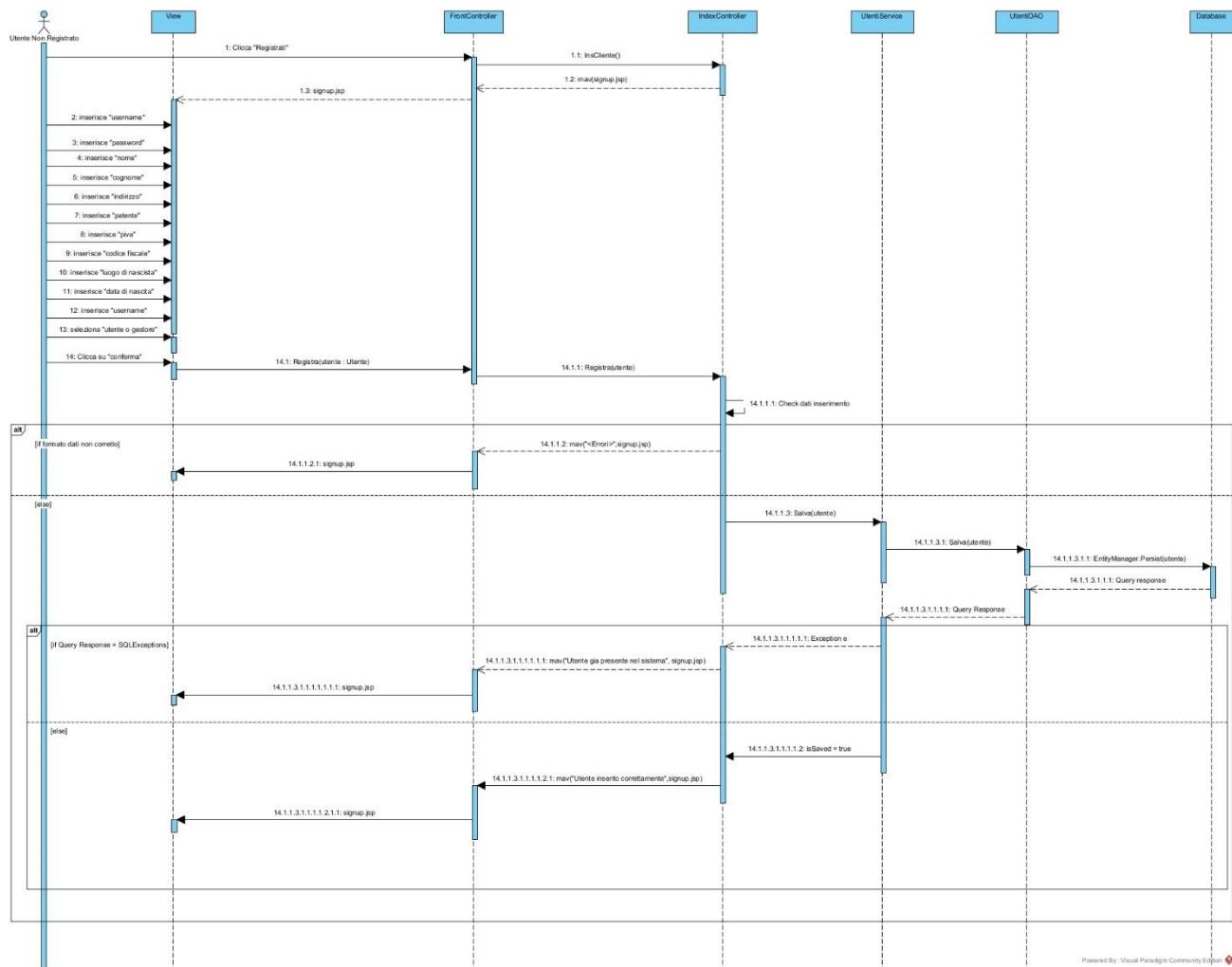
Devono essere definiti in fase di elaborazione per determinare gli eventi e le operazioni del sistema e per scrivere i contratti delle operazioni di sistema.

Per una visualizzazione più accurata si rimanda al “.vpp” fornito in allegato al documento.

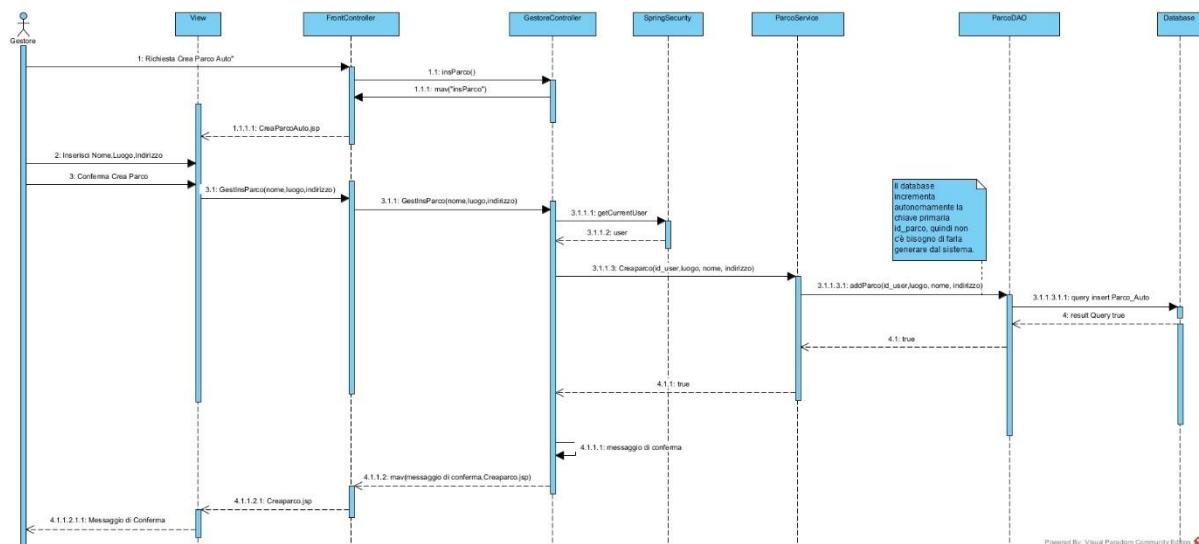
4.5.1 EFFETTUA LOGIN - SEQUENCE DIAGRAM DI DETTAGLIO



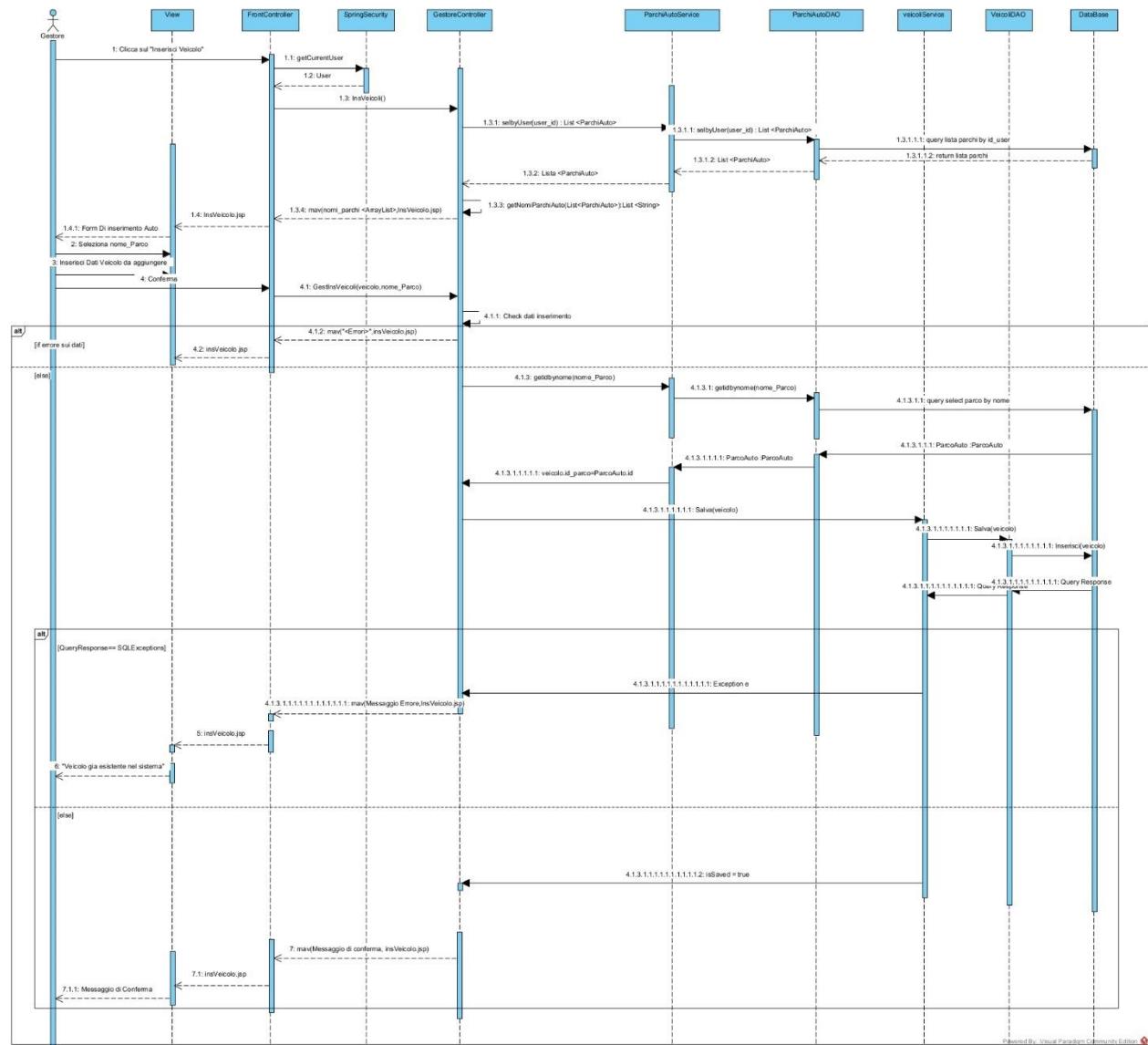
4.5.2 REGISTRAZIONE - SEQUENCE DIAGRAM DI DETTAGLIO



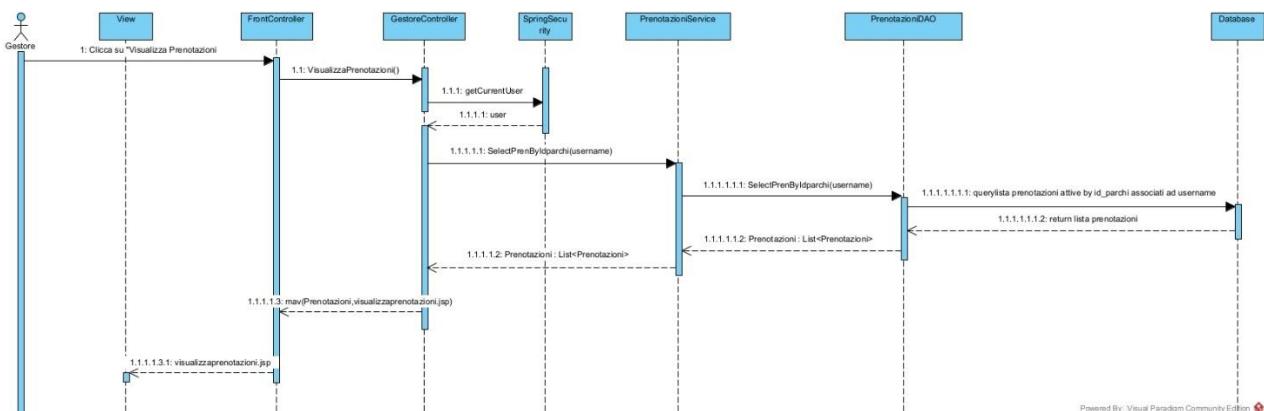
4.5.3 CREA PARCO AUTO - SEQUENCE DIAGRAM DI DETTAGLIO



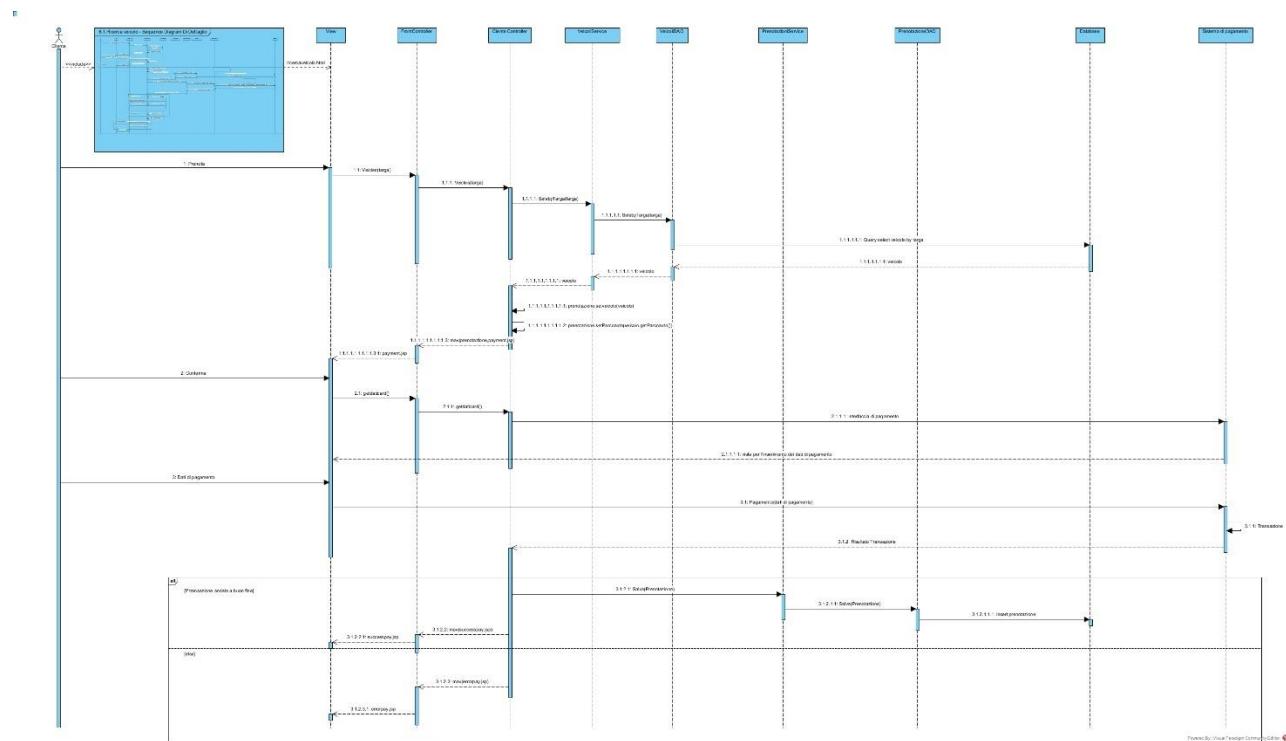
4.5.4 INSERISCI NUOVO VEICOLO - SEQUENCE DIAGRAM DI DETTAGLIO



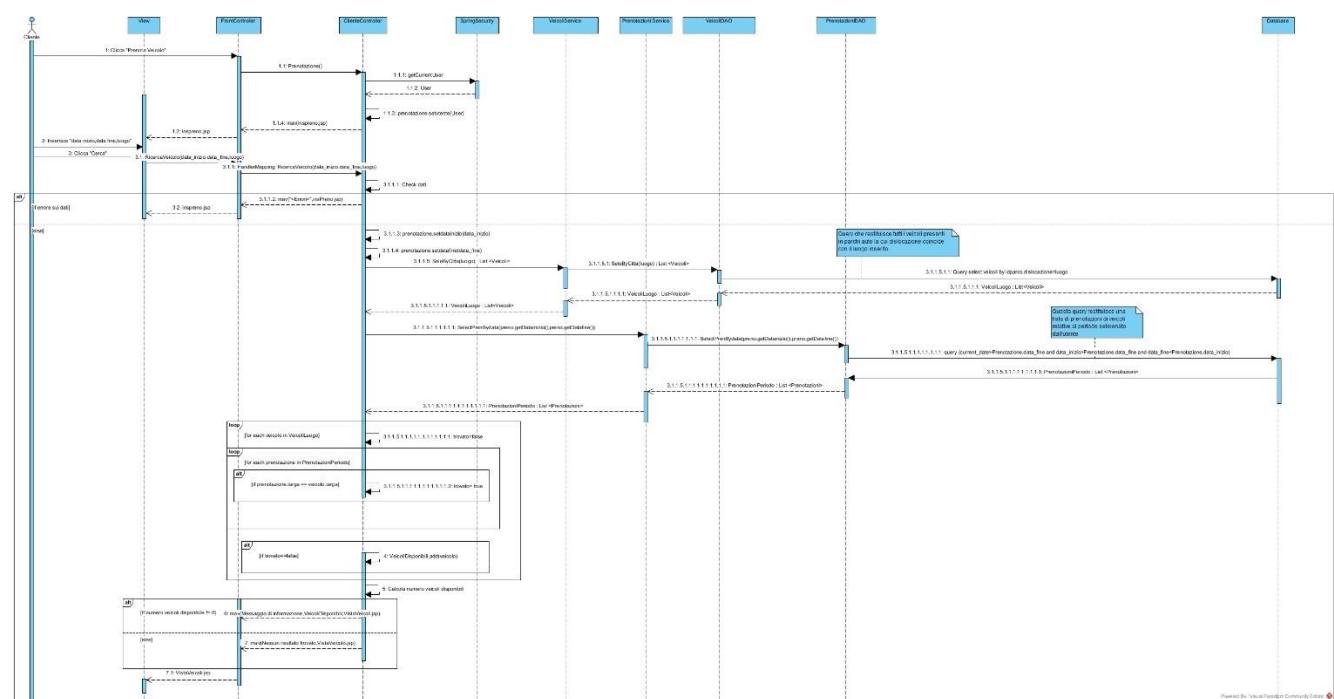
4.5.5 VISUALIZZA LISTA PRENOTAZIONI - SEQUENCE DIAGRAM DI DETTAGLIO



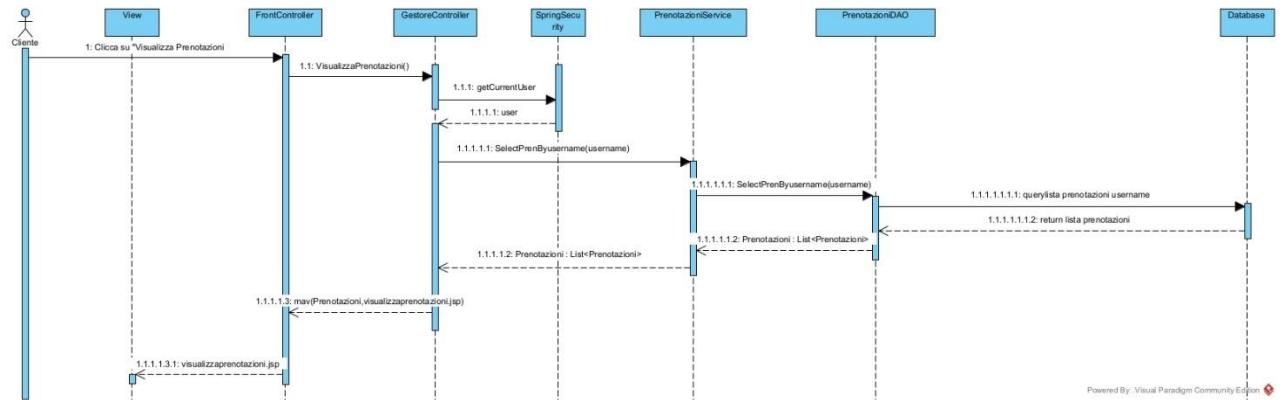
4.5.6 PRENOTA VEICOLO - SEQUENCE DIAGRAM DI DETTAGLIO



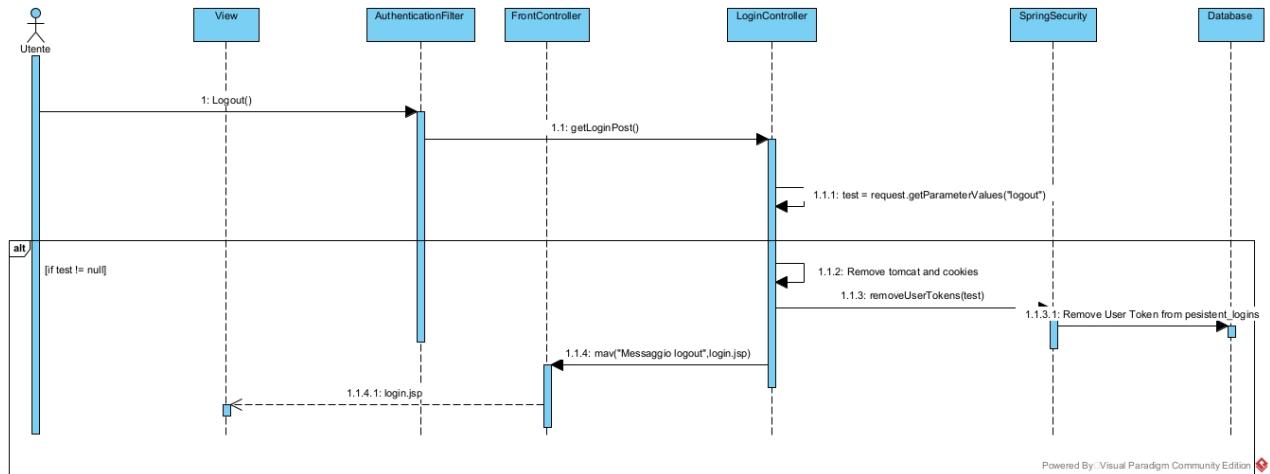
4.5.7 RICERCA VEICOLO - SEQUENCE DIAGRAM DI DETTAGLIO



4.5.8 VISUALIZZA LISTA PRENOTAZIONI CLIENTE - SEQUENCE DIAGRAM DI DETTAGLIO



4.5.9 LOGOUT - SEQUENCE DIAGRAM DI DETTAGLIO



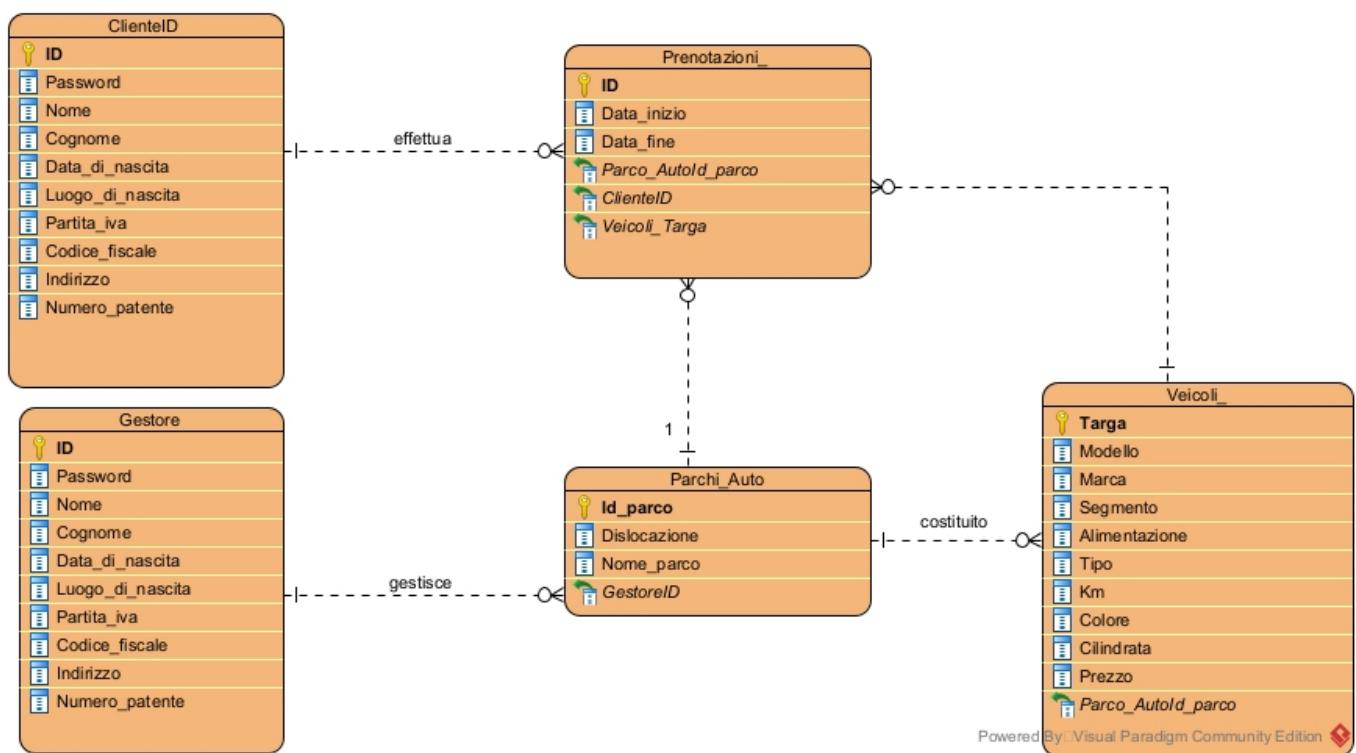
CAPITOLO 5 : PERSISTENZA DEI DATI

Al fine di garantire la persistenza dei dati utilizzati dalla web app, il team ha optato per l'utilizzo del software **Microsoft SQL Server 2019 Express** per la creazione di un database relazionale.

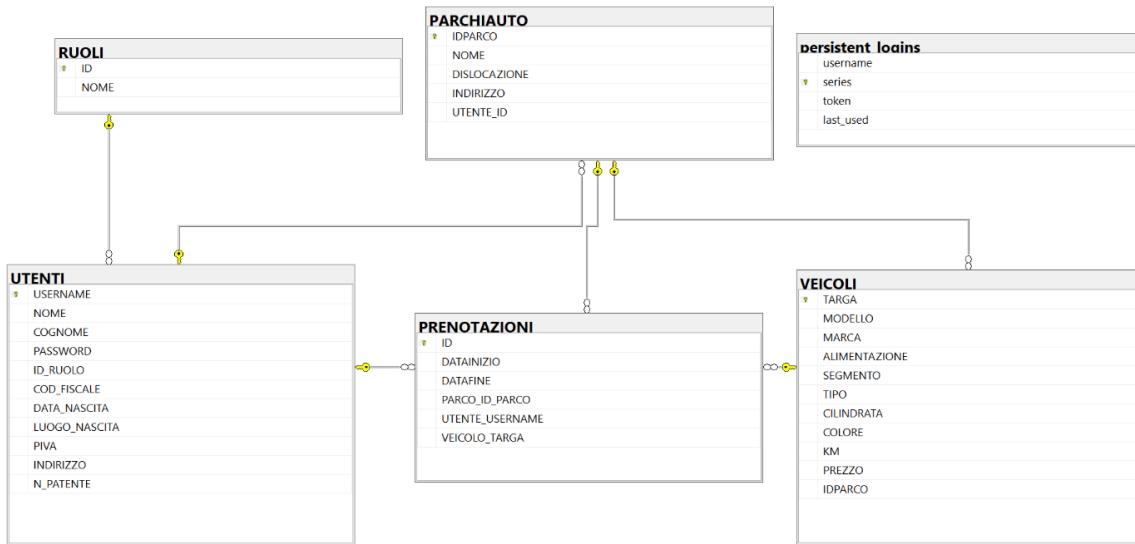
Nello specifico, per facilitare la creazione del database è stato utilizzato il tool **Microsoft SQL Server Management Studio 18** che ha permesso la realizzazione del suddetto in maniera semplice, ricorrendo a strumenti grafici che hanno permettono di raggiungere lo scopo avendo una conoscenza del linguaggio SQL basilare.

Dapprincipio per avere una corretta strutturazione del database il team ha ideato e realizzato uno schema E-R (*Entità-Relazioni*) che ha messo in evidenza alcune scelte realizzative e vincoli da rispettare tra le varie tabelle della struttura dati.

Di seguito si riporta l'ideazione prescrittiva del database:



In seguito, nella fase di creazione del database, per evitare ridondanza di informazioni che avrebbero portato ad un appesantimento di quest'ultimo, che a sua volta si sarebbe ripercossa sul software che lo utilizza limitandone così le prestazioni, il team ha deciso di apportare le seguenti modifiche:



Nello specifico possiamo osservare che:

- Per facilitare e rendere più snello il processo di login e registrazione è stata introdotta una tabella utente che sostituisce quella gestore e cliente nella quale è stato inserito un campo inherente all'id del ruolo dell'utente che lo specializza nelle due classi che utilizzeranno il sistema.
- Una tabella Ruoli la cui chiave primaria è chiave esterna per la tabella utente in modo da evitare ridondanza di dati.
- Una terza tabella fittizia è stata aggiunta per usufruire delle funzionalità di **Spring Security**. In particolare, tale tabella seppur interna alla struttura dati non è collegata al resto della struttura in quanto tutte le informazioni contenute al suo interno sono inerenti solo alle sessioni attive e vengono cancellate in caso di logout.

5.1 DEFINIZIONE DELLE RELAZIONI TRA ENTITÀ

Di seguito vengono elencate le relazioni tra le varie entità:

1. **UTENTI – RUOLI** : Ad ogni utente è associato un solo ruolo , ogni ruolo è associato a più utenti.
2. **UTENTI – PRENOTAZIONI** : Ogni utente è associato a più prenotazioni, una prenotazione è riferita da un unico utente.
3. **UTENTI – PARCHIAUTO** : Un utente può avere più parchi auto, un parco auto è riferito ad un solo utente.
4. **PRENOTAZIONI – VEICOLI** : Una prenotazione è riferita ad un solo veicolo, un veicolo è presente in più prenotazioni.
5. **PRENOTAZIONI – PARCHIAUTO** : Una prenotazione è riferita ad un solo parco auto, un parco auto può avere più prenotazioni.
6. **VEICOLI – PARCHIAUTO** : Un veicolo è riferito ad un solo parco auto, un parco auto può avere più veicoli.

I riferimenti tra le tabelle sono stati mappati nelle entità del software utilizzando Hibernate che mette a disposizione una serie di tag da applicare sulle chiavi esterne che permette di stanziare le relazioni fra le varie entità in base alla struttura del database di riferimento.

Il framework Hibernate verrà trattato nel capitolo riguardante lo sviluppo.

CAPITOLO 6 : IMPLEMENTAZIONE

Al fine di garantire il soddisfacimento dei requisiti funzionali e non funzionali e per facilitare il processo di sviluppo, il team ha deciso di utilizzare una serie di framework open source per la produzione del prodotto finale.

Il processo di sviluppo è stato effettuato, coerentemente al metodo Agile, in diverse iterazioni in ognuna della quali è stato rilasciato un software funzionante e potenzialmente pronto all'uso. Nel corso delle iterazioni il software è stato raffinato aggiungendo nuove funzionalità e migliorando quelle già esistenti.

Prima di elencare tutte le iterazioni e le release del software, il team ha ritenuto opportuno inserire nel contesto una rapida panoramica delle tecnologie adoperate.

Un framework, nella produzione del software, è una struttura di supporto su cui un software può essere organizzato e progettato. Con l'aumento della produzione di software e della complessità delle interfacce grafiche sono nati sempre più frequentemente framework con lo scopo di facilitare il più possibile la vita allo sviluppatore, da un lato per evitare la creazione ex novo di interfacce spesso molto simili da un'applicazione all'altra, ma anche e soprattutto per assisterlo nella progettazione dell'applicazione. Un framework è solitamente definito da una serie di classi astratte che saranno implementate dall'applicazione da realizzare; in questo modo il programmatore ha a disposizione una ben definita struttura.

Il Pattern architettonico scelto è stato il Model-View-Controller, diventato una pietra miliare per le applicazioni Web-based, poiché riesce brillantemente a separare tutta la logica business e la rappresentazione delle informazioni dalle interazioni utente con quest'ultime.

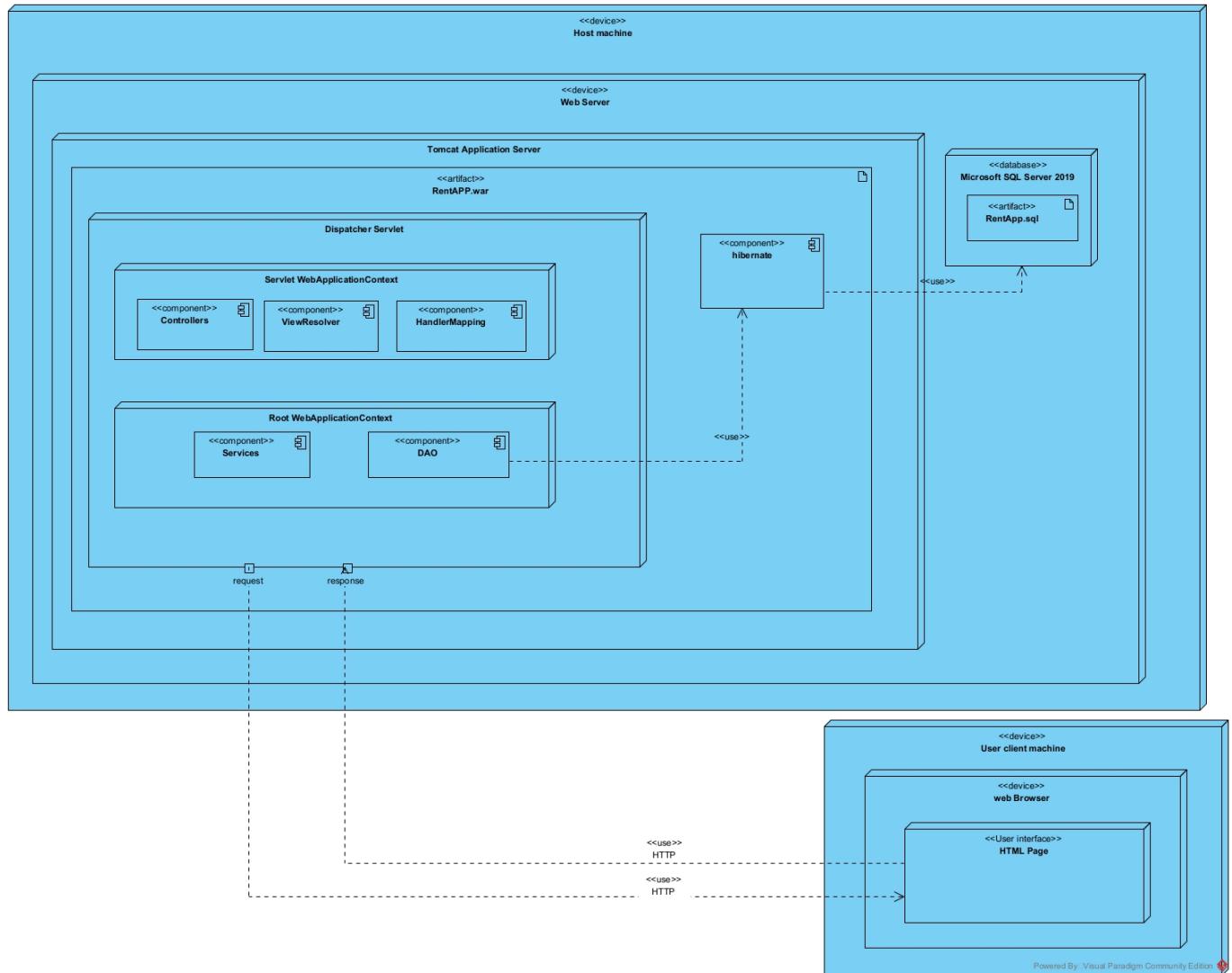
Tale modello è costituito sostanzialmente da 3 strati: la parte business formata da Model e Controller, e l'interfaccia utente (View). Nel dettaglio:

- Model Implementa la logica di business, fornendo i metodi utili per l'accesso ai dati dell'applicazione ed ha la responsabilità della gestione del database.
- Controller Implementa la logica di controllo, riceve i comandi dell'utente (in genere attraverso lo strato View) e li attua modificando lo stato degli altri due componenti.
- View si riferisce alla parte dell'applicazione che gestisce le visualizzazioni delle schermate da presentare all'utente. In tale sezione deve essere ridotta al minimo la logica di business, dovendosi occupare solamente della creazione dell'interfaccia grafica da presentare all'utente.

È importante far notare che sia lo strato View che lo strato Controller dipendono direttamente dal Model, il quale non dipende dagli altri. Questo è uno dei fattori più importanti di questa architettura, poiché permette al modello di essere implementato e testato indipendentemente dallo strato di visualizzazione. Grazie a questi accorgimenti, molteplici sono i benefici derivati: grazie a tale approccio, è possibile implementare viste multiple, permettendo l'esposizione degli stessi dati allo stesso istante in modi diversi.

6.1 DEPLOYMENT DIAGRAM

Al fine di avere una panoramica più chiara della costituzione del software e come esso viene distribuito sulle risorse hardware si propone il seguente diagramma di deploy:



Il sistema software è dispiegato su due nodi:

- **Nodo Server:** dove è in esecuzione l'applicazione web Spring su web server Tomcat e il Database Microsoft SQL Server 2019 , e che implementa logica di controllo, elaborazione e persistenza
- **Nodo Client:** dove è in esecuzione l'interfaccia web su web browser che implementa unicamente la logica di presentazione.

6.2 SPRING MVC

“Spring è un framework open source nato con l'intento di gestire la complessità nello sviluppo di applicazioni enterprise.” .

Spring si basa sul principio di **Dependency Injection** tramite il quale, contrariamente a quanto accade utilizzando una libreria, in cui è il codice sviluppato a richiamare gli elementi definiti nella stessa, il codice viene richiamato dai componenti del framework.

La DI prevede che tutti gli oggetti all'interno dell'applicazione accettino le dipendenze, ovvero gli oggetti di cui hanno bisogno, tramite costruttori o metodi setter. Non sono quindi gli stessi oggetti a creare le proprie dipendenze, ma esse vengono iniettate dall'esterno.

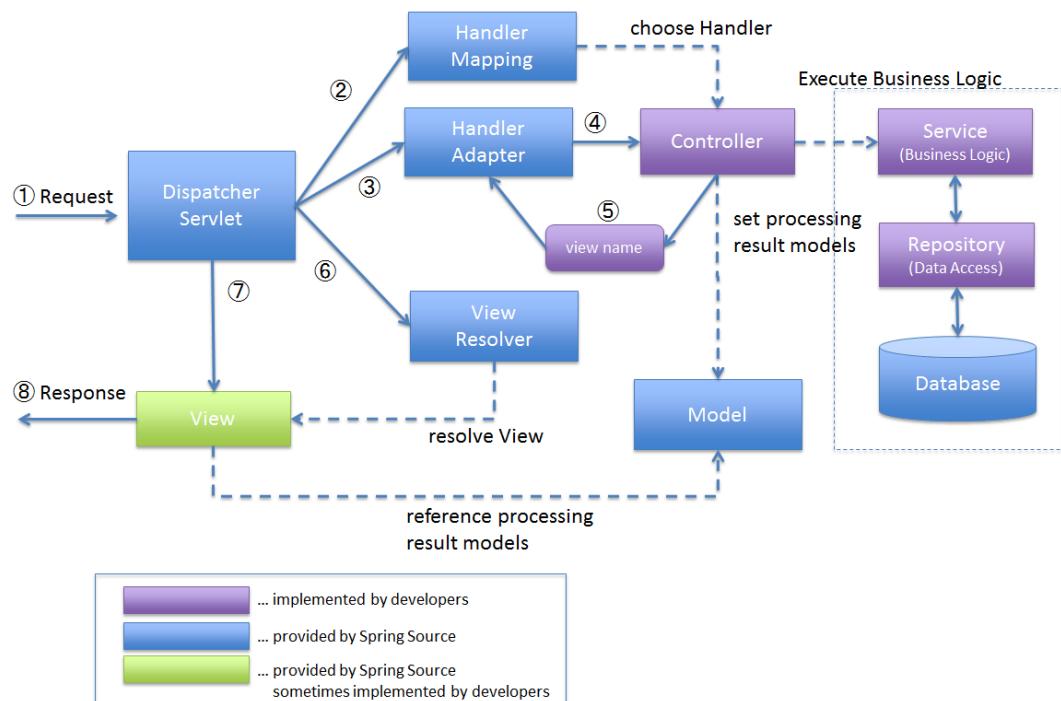
Inoltre, si occupa di istanziare gli oggetti (Beans) dichiarati nel progetto e di reperire ed iniettare tutte le dipendenze ad essi associate. Tali dipendenze possono essere i componenti del framework o altri Beans dichiarati nel contesto applicativo.

I vari Beans possono essere annotati come Componenti(Component), oppure specializzate come Controller, Service o Repository in base al layer in cui giacciono.

Tra le varie funzionalità offerte da Spring, **SpringMVC** permette lo sviluppo di applicazioni che seguono un pattern architetturale MVC.

Tutto si basa attorno ad un *Front Controller* o “*Dispatcher Servlet*”, che costituisce l'unica vera variante rispetto al modello “classico”. Tale componente ha il compito di inoltrare le richieste ottenute dalle HTTP request all'interfaccia *Handler Mapping*, la quale è specializzata nell'interpretare le richieste e decidere a quale controller devono essere inoltrate. Una volta inoltrate le richieste ai Controller, dopo un elaborazione, questi restituiscono i risultati al Dispatcher Servlet , che attraverso un modulo chiamato *View Resolver* fornisce la specifica View al client.

Nell'immagine che segue viene mostrato un diagramma che mostra un flusso di funzionamento del sistema:



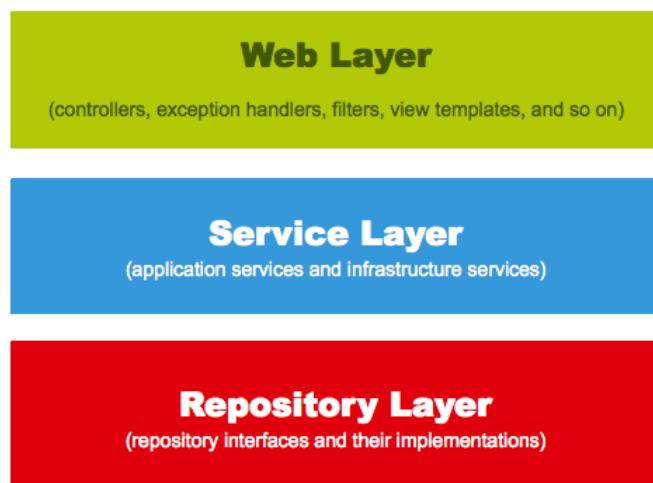
I moduli più importanti di Spring MVC dunque sono:

- HandlerMapping Seleziona quale Controller dovrà gestire la richiesta in arrivo secondo i suoi attributi.
- HandlerAdapter Permette l'esecuzione della logica del Controller selezionato dal HandlerMapping.
- Controller Nel mezzo tra Model e View, amministra le richieste in arrivo e reindirizza le risposte al View.
- View È responsabile di far dialogare l'applicazione con l'utente, gestendo le risposte che possono provenire dal Controller direttamente o dal Model.

Nel dettaglio l'immagine mostra il seguente flusso di operazione:

1. L'utente effettua una richiesta al server per qualche risorsa
2. La DispatcherServlet consulta l'HandlerMapping per trovare il controller opportuno
3. La DispatcherServlet invoca l'HandlerAdapter
4. Tramite l'HandlerAdapter viene chiamata la logica del controller
5. Il Controller processa la richiesta e ritorna un oggetto ModelAndView alla DispatcherServlet sempre tramite l'HandlerAdapter
6. La DispatcherServlet manda il nome della view al View Resolver per trovare la pagina da visualizzare
7. La DispatcherServlet passa l'oggetto del model alla View per restituire il risultato.
8. Il risultato viene restituito all'utente

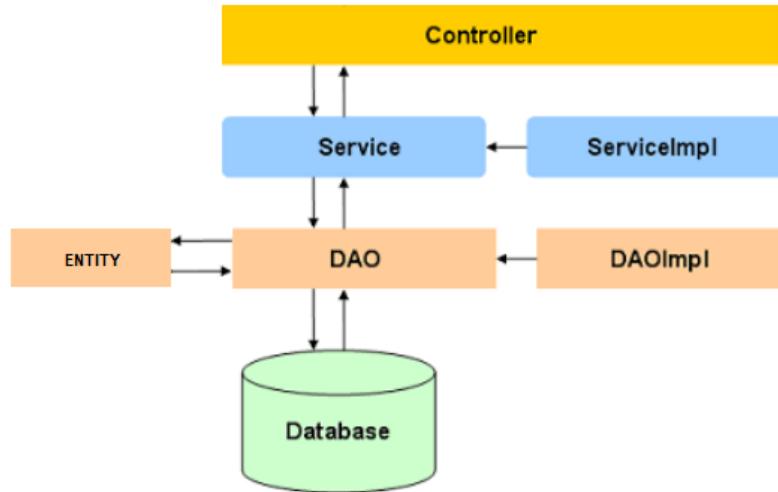
È possibile rappresentare l'intera architettura Spring in un modello three layer:



- **Web Layer:** rappresenta il livello in cui risiede il processing delle informazioni e i meccanismi di restituzione delle viste. Includerà tra l'altro i controller indicati da spring con la notazione "@Controller" e le viste.
- **Service Layer:** è un livello intermedio che contiene parte della logica di business e rappresenta uno strato di collegamento tra i controller e le interfacce del database. Fanno parte di questo layer i Service indicati da spring con la notazione "@Service".
- **Repository Layer:** fornisce il punto d'accesso al database , contiene tutte le repository o DAO che forniscono un'astrazione del database sottostante e consentono al software di comunicare con esso fornendogli delle opportune interfacce. Spring etichetta i DAO con la notazione "@Repository". Il collegamento tra i DAO e il database è stato realizzato nel nostro caso utilizzando il Middleware "Hibernate" di cui tratteremo a breve.

6.2.1 UTILIZZO DI SPRING MVC

Come anticipato nel paragrafo precedente, SpringMVC è un framework a strati. Coerentemente al framework adoperato, la nostra applicazione, anche al fine di garantire una più semplice modificabilità e riuso, è stata implementata in maniera modulare a livelli.



È possibile individuare i Seguenti moduli principali:

- **Controller**
- **Service**
- **DAO**
- **Entity**

Le entità altro non sono che le classi create e mappate sul database attraverso l'annotazione @Entity utilizzando Hibernate.

Le Entity presenti nel nostro software sono:

- *ParchiAuto*
- *Prenotazioni*
- *Ruoli*
- *Utenti*
- *Veicoli*

Le classi DAO o *Data Access Objects* hanno il compito di interagire con il database. All'interno di quest'ultime oltre alle Query specifiche progettate per adempiere alle funzionalità offerte dal sistema , devono essere sempre presenti quattro metodi di base chiamati **CRUD** che permettono di effettuare le operazioni di base sul database:

- **Create** (INSERT)
- **Read** (SELECT)
- **Update** (UPDATE)
- **Delete** (DELETE)

Tutte le Query da noi pensate e sviluppate sono state scritte, al fine di evitare attacchi di Query Injection possibili con le *Stored Procedures*, utilizzando i CriteriaBuilder o il linguaggio JPQL.

Ad ogni classe Entity corrispondono due classi DAO, l'interfaccia e l'implementazione.

L'implementazione della classe DAO è preceduta dall'annotazione di Spring @Repository, ciò segnala a Spring l'interfaccia DAO che esegue le operazioni CRUD sulla base di dati.

Le classi Service vengono utilizzate per evitare ai controller che utilizzeranno i DAO l'uso diretto di questi.

Il motivo di tale scelta ricade nella ormai consueta metodologia di divisione in livelli per rendere ogni strato del software indipendente, quanto più possibile, dagli altri.

Per ogni classe DAO del sistema ci sarà dunque una classe intermedia Service, identificata da Spring con il tag @Service, che metterà a disposizione del controller la stessa interfaccia messagli a disposizione dal DAO. Tale scelta, dunque permette di ridurre notevolmente l'accoppiamento tra i moduli del sistema.

Il livello Controller si prende carico di effettuare i controlli sui dati che si vuole inserire nel database in modo da rispettare tutti i vincoli richiesti. Per identificare questa classe inseriamo l'annotazione di Spring @Controller.

Il team ha optato per lo sviluppo di 4 controller indipendenti:

- *Index controller*: Gestisce le richieste alla welcome page e si occupa di gestire le registrazioni di nuovi utenti.
- *Login Controller*: si occupa della gestione delle richieste utente relative a login e logout.
- *Cliente Controller*: gestisce tutte le richieste che un utente può effettuare coprendo così la visualizzazione dello storico prenotazioni del cliente, la ricerca e la prenotazione di un veicolo (nella fase di testing e sviluppo il servizio di pagamento è stato realizzato tramite stub, le richieste allo stub vengono gestite stesso da questo controller).
- *Gestore Controller*: si occupa di gestire tutte le richieste possibili da parte di un gestore coprendo così la richiesta di visualizzazione di prenotazioni attive, la creazione di un nuovo parco auto e l'inserimento di un veicolo nel sistema.

Di seguito una tabella che mostra le richieste e le responsabilità dei vari controller:

Richiesta	URL	Controller	Descrizione
GET	/index	IndexController	Restituisce l'index, ovvero la pagina di welcome.
	/signup	IndexController	Restituisce la pagina di registrazione
POST	/signup	IndexController	Provvede al passaggio dei dati inseriti nel form della registrazione al database.
	/login/form	LoginController	Restituisce il form di login.
POST	/login/form	LoginController	Provvede al passaggio dei dati inseriti e quindi le credenziali, al database per verificare l'accesso.
	/login/form?logout	LoginController	Provvede e effettua il logout e a terminare la sessione dell'utente.
GET	/dashcliente/	ClientiController	Restituisce la dashboard del cliente se quest'ultimo ha effettuato l'accesso.
GET	/dashcliente/visualizzaprenotazioni	ClientiController	Restituisce la pagina con l'elenco delle prenotazioni effettuate dal cliente.
GET	/dashcliente/prenotazione	ClientiController	Restituisce il form di inserimento di una nuova prenotazione.

Richiesta	URL	Controller	Descrizione
POST	/dashcliente/prenotazione	ClientiController	Istanzia un nuovo oggetto di tipo prenotazione con i dati compilati nei form.
	/dashcliente/cercaLuogo	ClientiController	Restituisce l'elenco dei veicoli disponibili per l'utente in base alle date e alla dislocazione inserita.
	/dashcliente/pagamento/{targa}	ClientiController	Restituisce il riepilogo della prenotazione effettuata.
	/pagamento/{targa}/pay	ClientiController	Restituisce il form di inserimento dei dati di pagamento.
	/pagamento/{targa}/pay	ClientiController	Passa i dati del pagamento al database per verificarne la correttezza.
	**/Successopagamento	ClientiController	Restituisce una pagina dove è scritto che il pagamento è stato effettuato con successo.
	**/Errorepagamento	ClientiController	Restituisce una pagina dove è scritto che il pagamento non è andato a buon fine.
	/dashgestore	GestoriController	Restituisce la dashboard gestore se l'utente ha effettuato il login.
	/dashgestore /aggiungi	GestoriController	Restituisce il form di inserimento di un nuovo veicolo all'interno di un parco auto.
	/dashgestore /aggiungi	GestoriController	Passa i dati dell'inserimento veicolo al database creando un nuovo oggetto veicolo.
	/dashgestore /creaparco	GestoriController	Restituisce il form di creazione di un nuovo parcoauto.
	/dashgestore /creaparco	GestoriController	Passa i dati dei form di inserimento di creazione del parco al database istanziando un nuovo oggetto parco auto.
GET	/dashgestore/visualizzaprenotazioni	GestoriController	Restituisce l'elenco delle prenotazioni a carico di ogni parco auto del gestore.

Per una trattazione più approfondita sull'argomento si rimanda alla documentazione presente sul sito ufficiale di Spring.

6.3 SPRING SECURITY

Spring Security è un framework di autenticazione e controllo degli accessi potente e altamente personalizzabile. È lo standard di fatto per la protezione delle applicazioni basate su Spring.

Spring Security è un framework che si concentra sulla fornitura di autenticazione e autorizzazione alle applicazioni Java. Come tutti i progetti Spring, il vero potere di Spring Security si trova nella facilità con cui può essere esteso per soddisfare i requisiti personalizzati

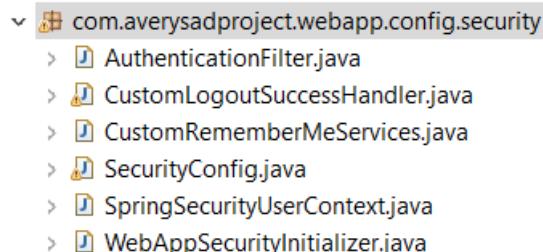
Esso mette a disposizione diverse funzionalità, tra cui la protezione dagli attacchi CSRF (Cross-site request forgery), ovvero gli attacchi che vengono fatti tramite comandi non autorizzati che minano i cambiamenti di sessioni e tecniche di attacco basate sugli header data, i form o i cookies.

Spring security gestisce anche il session fixation, ovvero i diversi accorgimenti che vengono fatti per impedire gli exploit sulle sessioni di un utente, l' HTTPS , ovvero il meccanismo di politica che protegge i siti web dagli attacchi di tipo man-in-the-middle, quelli sui protocolli e i cookie hijacking, il controllo sull'uso della cache e altri tipi di protezione.

6.3.1 UTILIZZO DI SPRING SECURITY

Le dipendenze di spring security vengono specificate nel pom.xml.

Il codice atto al funzionamento di Spring Security è interamente contenuto all'interno del package config.security:



Analizzando nel dettaglio i file di configurazione:

- **L'Authentication Filter** : tramite la funzione getAuthRequest è in grado di generare il token di autenticazione dove sono specificati username e password.
- **CustomLogoutSuccessHandler** : contiene l'handler in grado di gestire il logout di un utente tramite una funzione che intercetta la richiesta e rimuove il token di autenticazione per quel determinato utente.
- **CustomRememberMeServices** contiene la funzione in grado di garantire la persistenza del login utente.
- **SpringSecurityUserContext** contiene la funzione **getCurrentUser** che ritorna il nome dell'utente attualmente connesso o **“null”** se l'utente risulta anonimo e quindi non ancora registrato

- **SecurityConfig** : è sicuramente il più interessante dal punto di vista implementativo, questo si occupa infatti della configurazione di tutti gli elementi di base necessari al funzionamento del framework di sicurezza.
- Oltre che le funzioni di configurazione, è presente una chiamata **BCryptPasswordEncoder** che è in grado di cifrare la password inserita dall'utente in maniera tale che essa venga cifrata anche all'interno del database in modo da proteggere l'account dell'utente da possibili fughe di dati:

Le password vengono salvate nel database con il seguente formato
`{id}encodedPassword`, dove l'id che non può essere nullo permette al Password Encoder di identificare la tecnica di cifratura della password.
 Nel nostro progetto è stato utilizzato un algoritmo di hashing denominato bcrypt, che risulta essere di tipo "one-way", ovvero appartenente alla famiglia di funzioni computazionalmente semplici su ogni tipo di input ma difficili da "invertire" partendo dall'immagine di un input casuale.

Tramite la funzione **configure** è possibile definire quali siano i path delle risorse accessibili da un determinato ruolo. Nel caso di questo progetto sono stati definiti due ruoli, quello di cliente e quello di gestore. In tabella vi sono le directory principali a cui hanno accesso esclusivo i vari ruoli.

<i>Directory Path</i>	<i>Ruoli permessi</i>
<code>rentapp/dashcliente/**</code>	<code>ROLE_GESTORE</code>
<code>rentapp/dashgestore/**</code>	<code>ROLE_CLIENTE</code>
<code>rentapp/login/**</code>	<code>PERMITALL</code>
<code>rentapp/resources</code>	<code>PERMITALL</code>
<code>rentapp/</code>	<code>ROLE_ANONYMOUS,ROLE_GESTORE,ROLE_CLIENTE</code>

Inoltre, nella stessa Security Config è presente la funzione `persistentTokenRepository()` responsabile di salvare nella tabella `persistent_logins` le informazioni relative alla sessione.

Per poter funzionare Spring Security necessita di un proprio Service chiamato **"customUserDetailsService"**, tale service si interfaccia con il service relativo alla tabella utente al fine di costruire la sessione utente.

L'unica funzione che contiene è `loadUserByUsername`, funzione implementata dall'interfaccia `UserDetailsService` messa a disposizione da spring.

All'interno di questa funzione viene stanziato un service utente al fine di effettuare una Query sul database per ottenere l'utente a partire dall'username fornito dall'utilizzatore del sistema.

Viene poi costruito un `UserBuilder` che conterrà username, password e ruolo dell'utente.

Se la password contenuta nell'`UserBuilder` coinciderà con quella passata dall'utilizzatore in fase di login, il login sarà effettuato con successo salvando un token nel database.

Mediante questo token spring security ad ogni richiesta inviata dall'utilizzatore riconoscerà la sessione e permetterà o meno una determinata azione all'utente.

Per una trattazione più approfondita sull'argomento si rimanda alla documentazione presente sul sito ufficiale di Spring.

6.4 HIBERNATE

Il ruolo che svolge un ORM è quello di associare a ogni operazione e elemento usato nella gestione del database degli oggetti con adeguate proprietà e metodi, astraendo l'utilizzo del database dal DBMS specifico.

La necessità di un ORM nasce dall'intrinseca differenza tra il modello relazionale e quello ad oggetti; quest'ultimo, infatti, ha concetti come ereditarietà, polimorfismo, relazioni bidirezionali ed altre che non hanno una controparte nel mondo relazionale dei database. Si viene quindi a creare la relazione tra oggetto Java e tabella SQL.

L'esempio più noto di ORM per il linguaggio Java è **Hibernate**.

Hibernate è una piattaforma middleware open source, fornisce supporto non solo per quanto riguarda la mappatura di un modello di dominio orientato agli oggetti in un classico database relazionale, ma anche per quanto riguarda il reperimento dei dati da esso, ovvero la gestione delle query.

È stato utilizzato nel nostro progetto per la gestione e la comunicazione con il database SQL Server Express 2019. È necessario caricare le dipendenze per poter abilitare Hibernate sia nel file application.properties sia nel file pom.xml per affinché esse siano utilizzate dalla nostra web app. La persistenza è un concetto fondamentale nello sviluppo di applicazioni software.

Per la documentazione riguardante il funzionamento dettagliato di Hibernate, si rimanda il lettore al sito ufficiale in quanto la trattazione esula dallo scopo di questa documentazione.

6.5 ITERAZIONI DI IMPLEMENTAZIONE

ID	Implementazione	Inizio	Fine	Durata	Status
	Iterazione 1	16-06-2021	30-06-2021	15d	Completata
1	Mockup e creazione delle pagine html	16-06-2020	30-07-2021	15d	Parzialmente Completata
2	Creazione dei file di configurazione	16-06-2021	16-06-2021	1d	Completata
3	Configurazione Database e Creazione di Service e DAO	16-06-2021	16-06-2021	1d	Completata
4	Implementazione caso d'uso "Login"	17-06-2021	30-07-2021	15d	Parzialmente funzionante
5	Implementazione caso d'uso "Inserimento nuovo veicolo"	19-06-2021	19-06-2021	1d	Completata
6	Implementazione caso d'uso "Ricerca Veicolo"	20-06-2021	25-06-2021	6d	Completata
7	Implementazione caso d'uso "Prenota Veicolo"	23-06-2021	26-06-2021	4d	Completata
8	Testing and Debugging	27-06-2021	30-06-2021	4d	Completata
	Iterazione 2	01-07-2021	07-07-2021	7d	Completata
9	Mockup e creazione delle pagine html	01-07-2021	06-07-2021	6d	Parzialmente Completata
10	Implementazione caso d'uso "Crea parco auto"	01-07-2021	02-07-2021	2d	Completata
10	Implementazione caso d'uso "Login"	01-07-2021	07-07-2021	5d	Completata
11	Implementazione caso d'uso "Registrazione"	02-07-2021	04-07-2021	3d	Completata
12	Implementazione caso d'uso "Visualizza prenotazioni"	05-07-2021	06-07-2021	2d	Completata
11	Testing and Debugging	06-07-2021	07-07-2021	2d	Completata

	Iterazione 3	08-07-2021	21-07-2021	14d	Completata
13	Revisione “Login” per aggiunta di funzionalità necessarie al “Logout”	08-07-2021	15-07-2021	8d	Revisionata
14	Implementazione caso d’uso “Logout”	08-07-2021	15 -07-2021	8d	Completata
15	Implementazione caso d’uso “Visualizza prenotazioni cliente”	16-07-2021	19-07-2021	4d	Completata
16	Testing and Debugging	19-07-2021	21-07-2021	3d	Completata

Per quanto riguarda la consistenza dei dati, le *constraints* sull'inserimento delle variabili sono state ottenute con l'impiego della libreria *javax.validation.constraints*. e *hibernate validator engine*
I tipi di vincoli importati sono stati :

@Id: specifica la primary key di una tabella.
@GeneratedValue: implementa una strategia di creazione degli Id, come ad esempio l'auto incremento.
@Column: tramite Hibernate associa una classe alla colonna di una tabella di un modello ER.
@NotEmpty: vincola l'elemento etichettato a non essere vuoto.
@NotNull: l'elemento etichettato deve essere non nullo (viene da noi utilizzato per i tipi interi).
@OneToOne: permette di specificare una relazione con molteplicità uno a molti.
@ManyToOne: permette di specificare una relazione con molteplicità molti a uno.
@Valid: applicata sui metodi, attiverà la verifica e la validazione dei dati inseriti nei form delle View.
@JoinColumn: etichetta un elemento come chiave esterna di un modello ER per una relazione tra le entità.
La suddetta notazione necessita di un *FetchType*, ovvero il metodo di estrazione dei dati dalla tabella. Esso può essere di tipo *LAZY* se all'occorrenza di una chiamata i valori vengono estratti dalle tabelle chiamate e quindi il fetch avviene "on-demand" ovvero solo quando i valori sono necessari, oppure di tipo *EAGER* se le risorse vengono estratte dalle tabelle a prescindere se esse verranno utilizzate o meno.
@Temporal viene utilizzato per far riferimento a un oggetto appartenente alla classe *java.util.date* o *java.util.calendar*.
@Size: etichetta usata per specificare la grandezza massima e minima di una variabile.
Ad esempio, la variabile "*codfis*" che rappresenta il codice fiscale di un utente, deve per forza contenere 16 caratteri, altrimenti viene restituito un errore.

I *messaggi di errore* vengono specificati all'interno delle etichette. In particolare, la chiamata viene fatta nella dichiarazione della suddetta specificando *message="NomeMessaggioDiErroreOWarning"*, la corrispondenza di questi messaggi viene poi specificata in un apposito file denominato *messages.properties* in cui sono presenti i valori da restituire all'interfaccia, il testo degli elementi button, gli alert e i warning.

Questi ultimi vengono restituiti all'interno dell'interfaccia tramite il comando <spring:message code "codice-di-errore-specificato-nei-message.properties"/>.

Per quanto riguarda invece le eccezioni e gli errori, queste vengono gestite all'interno degli appositi controller, tramite comandi di tipo try{ code }catch(exception){code}.

In merito all'inserimento dei dati all'interno dei form creati, dati che poi vengono salvati sul database relazionale, se non imponiamo nessun livello di controllo su ciò che l'utente inserisce, in linea teorica un utente potrebbe inserire qualsiasi valore nei campi, anche codice potenzialmente pericoloso che potrebbe andare a danneggiare la nostra applicazione.

È quindi fondamentale inserire un sistema di validazione:

La *Bean Validation* è un sistema di validazione che si basa su delle specifiche notazioni che vengono prelevate da Hibernate.

Per poter attivare la Bean Validation è stato necessario apportare delle modifiche al progetto.

Nel file pom.xml è stata aggiunta una dipendenza che attiva tali strumenti (*Hibernate Validator*).

Per poter attivare la validazione è stato necessario istanziare un nuovo Bean nominato Validator che sovraintende alla validazione dei dati di tipo *LocalValidatorFactoryBean*, all'interno del *WebApplicationContextConfig.java*, dove è stata settata l'origine dei messaggi che verranno visualizzati a video.

```

@Bean(name = "validator")
public LocalValidatorFactoryBean validator()
{
    LocalValidatorFactoryBean bean = new LocalValidatorFactoryBean();
    bean.setValidationMessageSource(messageSource());

    return bean;
}

```

Quando la web app incorre in un errore di validazione, i relativi messaggi saranno prelevati dalla *MessageSource*, un altro Bean dove è presente la regola che definisce il prelievo dei messaggi dal file *messages.properties* nella quale saranno dichiarati.

```

@Bean
public MessageSource messageSource()
{
    ResourceBundleMessageSource resource = new ResourceBundleMessageSource();
    resource.setBasename("messages");

    return resource;
}

```

```

insveicolo.form.btnAdd.label = Aggiungi
insveicolo.form.btnAbort.label = Dashboard
insveicolo.form.alert.label = Veicolo Aggiunto con Successo
insveicolo.form.warning.label = Si è verificato un errore
insarticolo.form.btnAdd.label = Conferma
Success.titolo = P
datip.carta = Numero carta:
datip.csv = CV:
datip.data= Data di scadenza:
datip.titolare = Titolare Carta:
inscliente.form.alert1.label = Username già esistente
NotNull.ParchiAuto.nome.validation = Inserire Nome Parco Auto
NotNull.ParchiAuto.indirizzo.validation = Inserire Indirizzo
NotNull.ParchiAuto.dislocazione.validation = Campo Dislocazione non può essere vuoto
Prenotazioni.datafine.validation = Data Fine non può essere vuota
successo.form.btnAbort.label = Dashboard
creaparcoauto.form.titolo.label = Crea Nuovo Parco Auto
creaparcoauto.form.alert.label = Creato Parco Auto con successo

```

L'ultima modifica è stata apportata al file *WebApplicationContextConfig.java*.

È stato eseguito l'Override del Validator con il metodo *getValidator*, con il quale viene restituito semplicemente il Validator appena creato, in assenza di questo metodo la visualizzazione dei messaggi non avverrebbe.

```

@PostMapping(value = "/creaparco")
public String GestInsParco(@Valid @ModelAttribute("newParco" ) ParchiAuto parco, BindingResult result, Model model,
                           RedirectAttributes redirectAttributes, HttpServletRequest request)
{
    if (result.hasErrors())
    {
        return "insParco";
    }
    ...
}

```

Ad esempio, nel metodo “*GestInsParco*” che invia materialmente i dati al nostro server c’è bisogno di effettuare un controllo, a tal fine viene utilizzato il *Bindingresult*. Laddove il risultato del nostro binding, quindi dell’invio dei dati, abbia degli errori e quest’ultimi siano dovuti ad un mancato rispetto della BeanValidation, allora il sistema restituirà nuovamente la schermata per l’inserimento dei dati accompagnata da un apposito messaggio d’errore.

CAPITOLO 8 : TESTING

Il testing è un'attività fondamentale del ciclo di sviluppo del sistema software e deve essere eseguita alla fine di ciascuna iterazione per verificare la presenza di errori o imperfezioni nel codice che potrebbero portare ad un sistema poco funzionale e poco affidabile.

In maniera continuativa, durante la fase di implementazione di ogni iterazione, è stato effettuato un **testing black box** sulle funzionalità via via implementate e rese eseguibili, utilizzando ove necessario , dei metodi stub, e tenendo conto degli output attesi facendo riferimento ai requisiti definiti ad inizio iterazione.

ID Test	Iterazione	Azione	Risultati attesi	Risultati ottenuti	Passato
1	1.1	Collegamento a http://localhost:8080/rentapp/	Viene restituita pagina di welcome	Viene restituita pagina di welcome	Si
1	1.2	Collegamento a http://localhost:8080/rentapp/index	Viene restituita pagina di welcome	Viene restituita pagina di welcome	Si
2	2.1	Click sul bottone Registrati	Viene restituito form registrazione	Viene restituito form registrazione	Si
3	3.1	Effettuo registrazione con campi vuoti	Impossibilità di portare a commissione la richiesta a causa della mancanza di informazioni	-Il nome deve essere compreso tra 2 e 50 caratteri - Il Cognome deve essere compreso fra 2 e 50 caratteri - L'indirizzo deve essere compreso fra 10 e 80 caratteri -La Partita IVA deve essere composta da 11 caratteri -Il luogo di nascita deve essere compreso tra 2 e 50 caratteri -Il nome deve essere compreso tra 2 e 50 caratteri	Si
3	3.2	Effettuo registrazione con username già esistente	Username già esistente	SQL Exception : Violazione dei vincoli di chiave primaria	No

3	3.3	Effettuo registrazione con username già esistente	Username già esistente	Assenza messaggio testuale di errore	No
3	3.4	Effettuo registrazione con username già esistente	Username già esistente	Corretta visualizzazione del messaggio	Si
3	3.5	Effettuo registrazione come gestore con campi corretti	Avvenuta registrazione	Utente salvato con successo	Si
3	3.6	Effettuo registrazione come utente con campi corretti	Avvenuta registrazione	Utente salvato con successo	Si
4	4.1	Accesso come gestore	Viene restituita la dashboard gestore	Benvenuto @nomeutente questa è la tua dashboard gestore	Si
4	4.2	Accesso come gestore sbagliando credenziali	Errore sulle credenziali	Nome utente o password errata	Si
4	4.3	Accesso con campo password vuoto	Errore sulle credenziali	Invito a riempire il campo lasciato vuoto	Si
5	5.1	Gestore clicca su visualizza prenotazioni	Viene restituito elenco vuoto delle prenotazioni	Viene restituito elenco vuoto delle prenotazioni	Si
5	5.2	Gestore clicca su visualizza prenotazioni,dopo che un utente ha prenotato un suo veicolo	Viene restituito elenco con le prenotazioni, dopo che un utente ha prenotato un suo veicolo	Viene restituito elenco con le prenotazioni, dopo che un utente ha prenotato un suo veicolo	Si
6	6.1	Gestore clicca su crea parco auto	Viene restituito form creazione parco auto	Viene restituito form creazione parco auto	Si

7	7.1	Creo parco con campi vuoti	Errore sui campi	Campo Dislocazione non può essere vuoto Inserire Nome Parco Auto Inserire Indirizzo	Si
7	7.2	Creo parco con dati corretti	Parco creato con successo	Creata Parco Auto con successo	Si
8	8.1	Gestore clicca su inserisci veicolo	Viene restituito form inserimento veicolo	Viene restituito form inserimento veicolo	Si
9	9.1	Inserisco veicolo con campi vuoti	Errore sui campi	Il modello non può essere vuoto, inserisci il modello La targa deve essere composta da 7 Caratteri Il segmento non può essere vuoto,inserisci segmento Il campo colore non può essere vuoto, inserisci colore La marca non può essere vuota, inserisci la marca La targa non può essere vuota,inserisci la targa Il tipo non può essere vuoto, inserisci tipo	Si
9	9.2	Inserisco veicolo con targa non valida	Errore sul campo targa	La targa deve essere composta da 7 Caratteri	Si
9	9.3	Inserisco veicolo correttamente	Considerazione: Devo ricordare il parco auto in cui inserire il veicolo	Veicolo Aggiunto con Successo,N.B: il sistema restituisce i nomi dei parchi in possesso del gestore in un menu a tendina	Si
9	9.4	Inserisco veicolo con la stessa targa di quello inserito precedentemente	Impossibilità di portare a termine la richiesta a causa del vincolo di chiave primaria	L'eccezione viene gestita, e a video viene mostrato un messaggio che riporta che la targa è già esistente	Si

9	10.1	Logout utente	Logout effettuato	Logout effettuato	Si
10	11.1	Accesso come cliente	Viene restituita la dashboard cliente	Viene restituita la dashboard cliente	Si
11	12.1	Accesso come cliente sbagliando credenziali	Errore sulle credenziali	Nome utente o password errata	Si
12	12.2	Accesso con campo password vuoto	Errore sui campi	Invito a riempire i campi	Si
12	13.1	Cliente clicca su visualizza prenotazioni	Viene restituito elenco vuoto delle prenotazioni	Viene restituito elenco vuoto delle prenotazioni	Si
13	13.2	Cliente clicca su visualizza prenotazioni,dopo averne effettuata una	Viene restituito l'elenco delle prenotazioni da lui effettuate	Viene restituito l'elenco delle prenotazioni da lui effettuate	Si
13	14.1	Cliente clicca su prenota veicolo	Viene restituito l'apposito form	Viene restituito l'apposito form	Si
14	15.1	Cliente inserisce date valide e dislocazione nota	Viene restituito elenco veicoli disponibili	Viene restituito elenco veicoli disponibili	Si
15	15.2	Cliente inserisce data fine prenotazione anteriore a quella di inizio	Errore sulle date	ATTENZIONE: Inserire le date in maniera corretta	Si
15	15.3	Cliente inserisce una data di inizio minore della data corrente	Errore sull'inserimento delle date	ATTENZIONE: Inserire le date in maniera corretta	Si
15	15.4	Cliente inserisce date corrette e dislocazione non	Messaggio in cui viene specificato che la dislocazione non esiste o che nessun parco auto al momento sul	Elenco vuoto dei veicoli disponibili, con calcolo dei giorni di effettiva prenotazione. Non mina alla	No

		valida (ad esempio Isola chenoncè)	database è virtualizzato nella data dislocazione	consistenza del sistema, ma poteva essere gestito	
15	15.5	Cliente inserisce date corrette e dislocazione sconosciuta	Nessun veicolo disponibile per date e luogo indicati	Il messaggio viene visualizzato correttamente	Si
15	16.1	Cliente prenota veicolo dopo che gli è stato restituito nell'elenco	Riepilogo dati e pagamento	Riepilogo dati veicolo	Si
16	17.1	Cliente dopo il riepilogo procede al pagamento	Form inserimento dati pagamento	Form inserimento dati pagamento	Si
17	18.1	Cliente inserisce i suoi dati di pagamento	Pagamento avvenuto con successo	Errore nella visualizzazione del form di pagamento, Metodo GET non supportato	No
18	18.2	Cliente inserisce i suoi dati di pagamento	Pagamento avvenuto con successo	Pagamento fallito	No
18	18.3	Cliente inserisce i dati di pagamento (Stub)	Pagamento avvenuto con successo	Pagamento avvenuto con successo	Si
18	18.4	Cliente inserisce dati pagamento errati	Pagamento fallito	Pagamento fallito	Si
18	19.1	Cliente prova a prenotare il veicolo appena prenotato/altro cliente prova a prenotare il veicolo nelle stesse date	Il veicolo non compare nell'elenco	Il veicolo non compare nell'elenco	Si
19	20.1	Cliente clicca su Effettua segnalazione	Viene restituita la vista Coming soon	Coming Soon	Si

20	21	Gestore inserisce veicolo con cilindrata come non intero	Errore su cilindrata che deve essere un intero	Errore java conversion	No
21	22.1	Cliente prova ad accedere come gestore	Errore	Accesso Negato! Autenticarsi con un utente diverso	Si
22	22.2	Gestore prova ad accedere come cliente	Errore	Accesso Negato! Autenticarsi con un utente diverso	Si
22	22.3	Utente admin prova ad accedere come cliente o gestore	Login effettuato con Successo	Login effettuato	Si

CAPITOLO 9 : INSTALLAZIONE E CONGIFURAZIONE

In questo capitolo verrà fatta una breve panoramica sull'istallazione dei software necessari affinché il prodotto da noi sviluppato funzioni correttamente.

9.1 APACHE TOMCAT

Come Application Web Server si è scelto di utilizzare Apache Tomcat 9.0. Per poter utilizzare il server, occorre innanzitutto scaricarlo dal sito ufficiale : <https://tomcat.apache.org/download-90.cgi>.

You are currently using **https://downloads.apache.org/**. If you encounter a problem with this mirror, please try another one from the mirrors list (that should be available).

Other mirrors: <https://downloads.apache.org/> ▾ [Change](#)

9.0.50

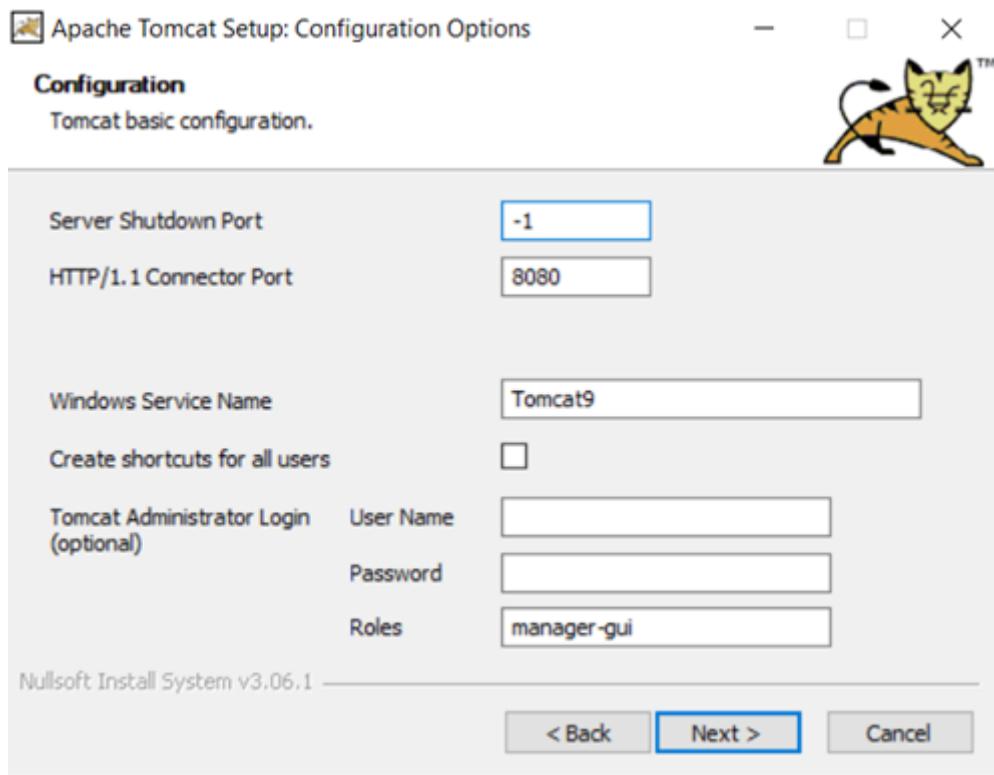
Please see the [README](#) file for packaging information. It explains what every distribution contains.

Binary Distributions

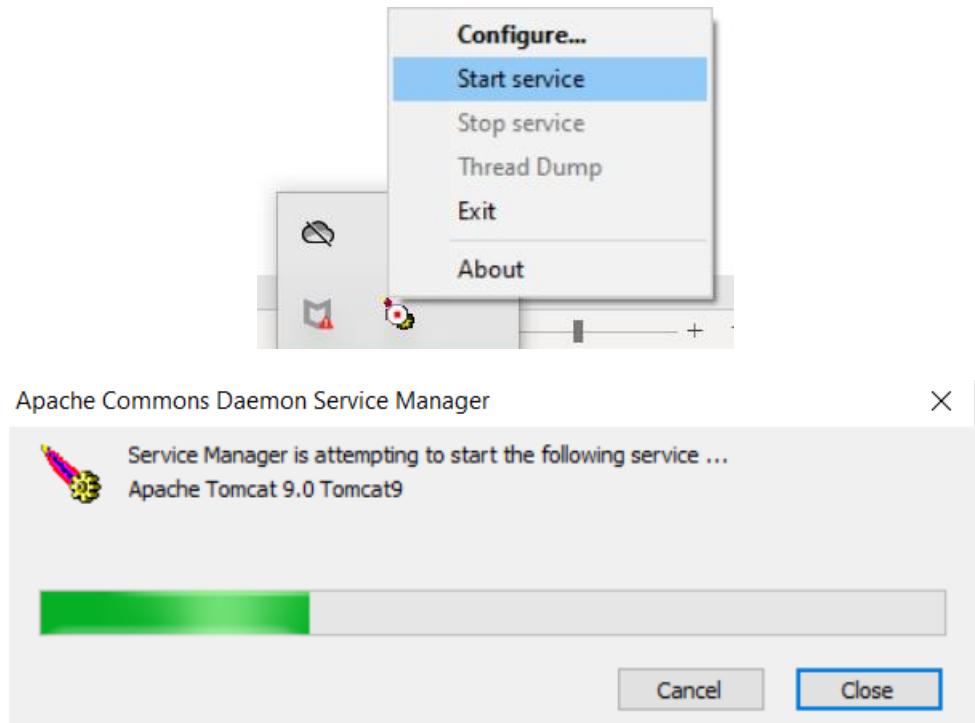
- Core:
 - [zip \(pgp, sha512\)](#)
 - [tar.gz \(pgp, sha512\)](#)
 - [32-bit Windows zip \(pgp, sha512\)](#)
 - [64-bit Windows zip \(pgp, sha512\)](#)
 - [32-bit/64-bit Windows Service Installer \(pgp, sha512\)](#)

Scaricare la versione da 32-bit/64-bit Windows Service Installer(pgp,sha512).

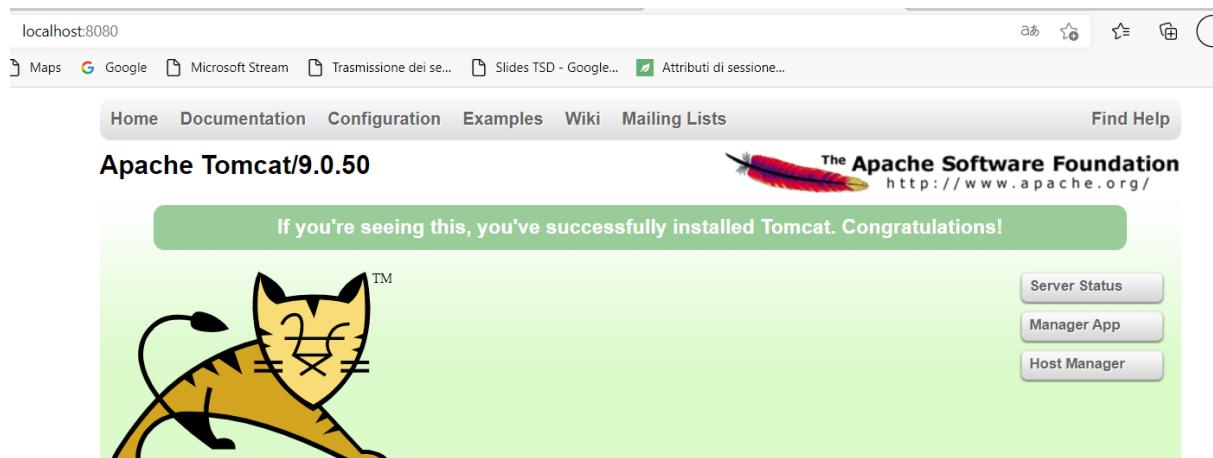
Dopodiché, occorre procedere all'installazione, bisogna specificare la porta,8080 di default, e un account per accedere alla gestione del server.



Installato il server, bisogna avviarlo dalla barra degli strumenti, e selezionare Start Service.



Una volta avviato il server, bisogna accedere sul sito tramite l'indirizzo localhost:8080/ e cliccare sulla sezione Manager App, specificando Nome Utente e password settate in fase di installazione.



E' possibile effettuare il deploy del progetto RentApp, nella sezione: WAR file to deploy.

Applications					
Path	Version	Display Name	Running	Sessions	Cor
/	None specified	Welcome to Tomcat	true	0	Sta
/docs	None specified	Tomcat Documentation	true	0	Sta
/host-manager	None specified	Tomcat Host Manager Application	true	0	Sta
/manager	None specified	Tomcat Manager Application	true	1	Sta
/rentapp	None specified		true	1	Sta

Deploy	
Deploy directory or WAR file located on server	
Context Path:	<input type="text"/>
Version (for parallel deployment):	<input type="text"/>
XML Configuration file path:	<input type="text"/>
WAR or Directory path:	<input type="text"/>
<input type="button" value="Deploy"/>	

WAR file to deploy	
Select WAR file to upload	<input type="button" value="Scegli il file"/> Nessun file scelto
<input type="button" value="Deploy"/>	

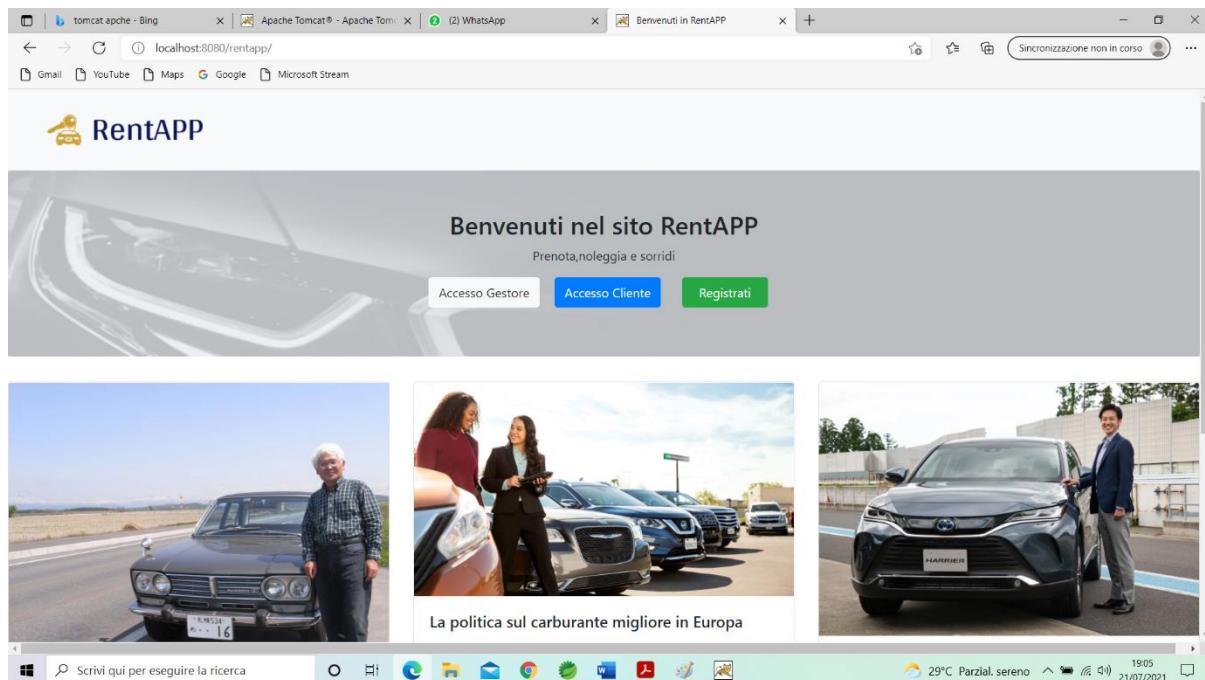
Per poter effettuare il deploy dell'applicazione sul server Tomcat 90. È necessario aggiungere queste modifiche aggiuntive al file pom.xml: il nome dell'applicativo nella sezione build, nella sezione properties la versione di Tomcat e infine nella sezione project il packaging war.

```
<packaging>war</packaging>

<finalName>rentapp</finalName>

<tomcat.version>9.0.38</tomcat.version>
```

Se la procedura è stata eseguita correttamente, sarà presente nella sezione Applications, il path per accedere alla web application RentApp. Il nome è stato specificato nel file pom.xml nella sezione build.



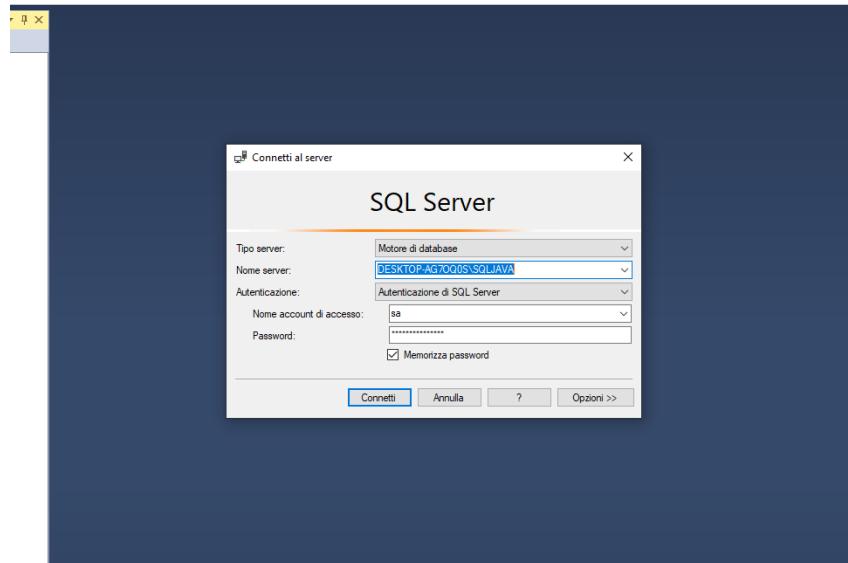
9.2 CONFIGURAZIONE DEL DATABASE

L'installazione di Tomcat non è l'unico requisito fondamentale per far funzionare la web app. In assenza del database relazionale ogni richiesta fatta alla web app che apparentemente sembrerà funzionare, comporterà un fallimento di comunicazione con il database in quanto sarà assente.

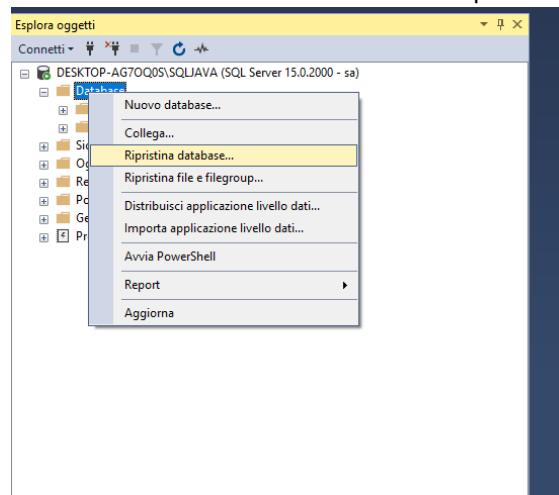
La configurazione del database è un processo semplice. Dopo aver scaricato e installato Microsoft SQL Server 2019 e Microsoft SQL Server Manager 2018 Express, basteranno pochi passi essenziali per configurare il database.

Di seguito sono riportati tutti i passi da seguire per la corretta configurazione.

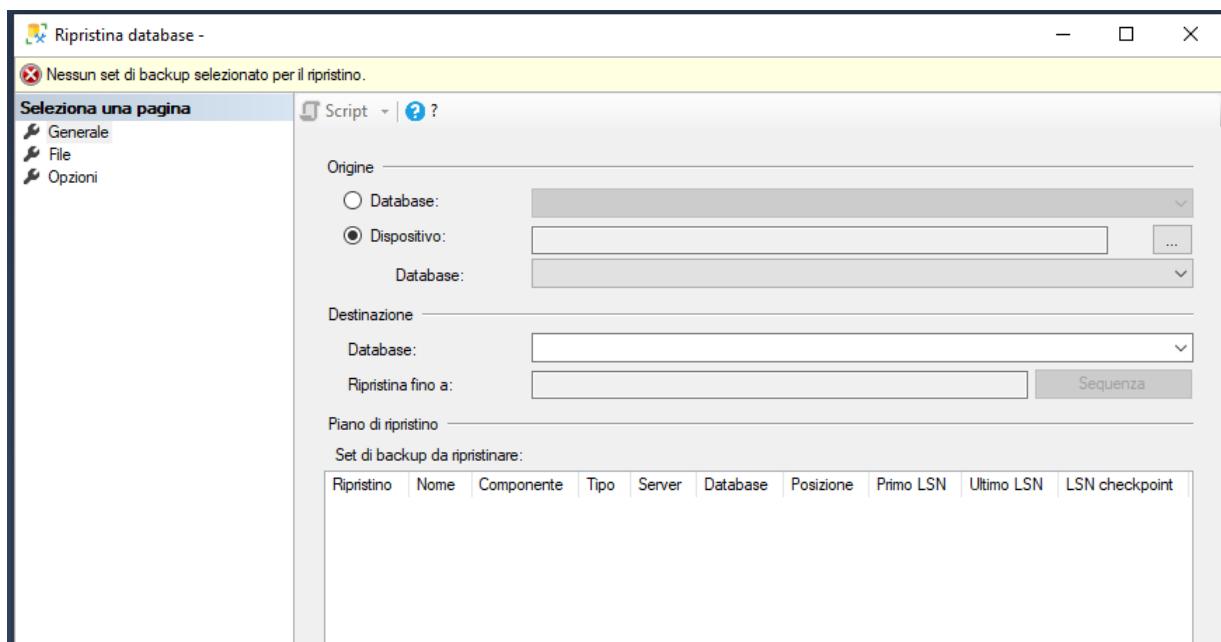
- Installare l'istanza SQL Server 2019 Express;
- Installare SQL Server Management Studio (SSMS)
- Accedere all' SSMS con le credenziali di system admin (sa) per la configurazione degli account e l'importazione del database:



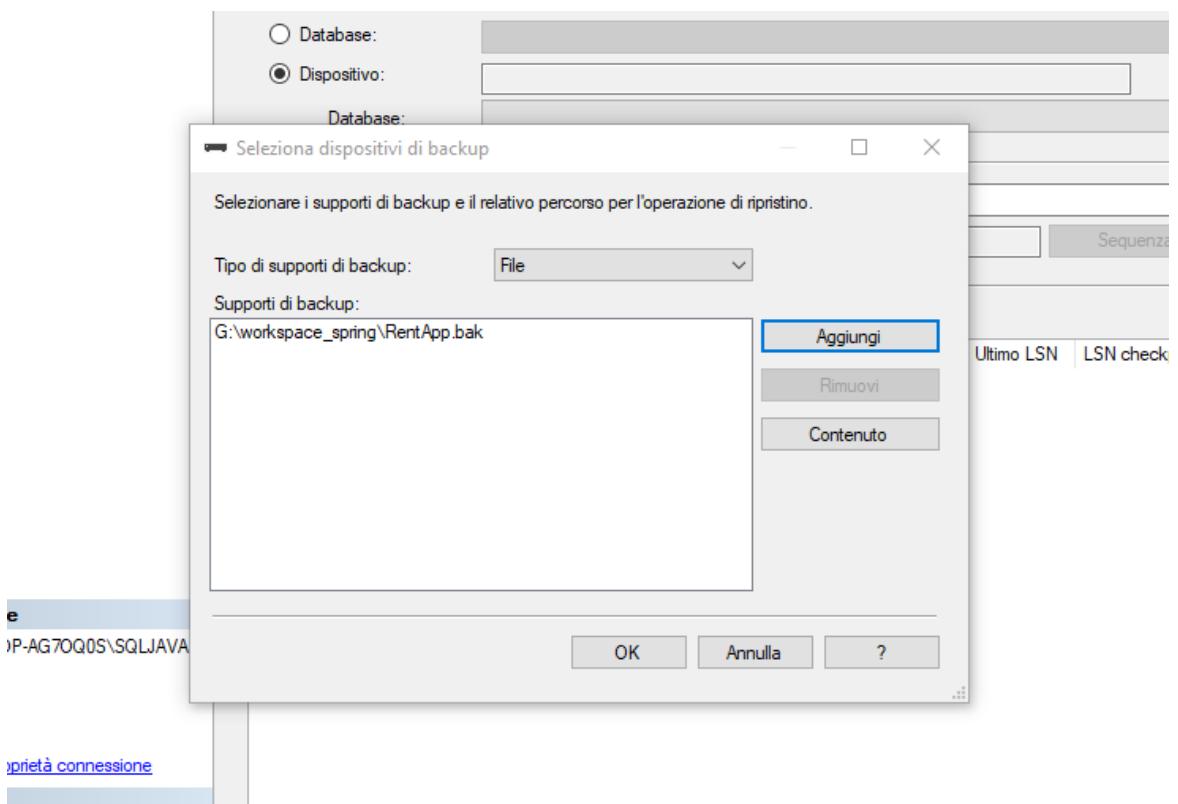
- Andare sulla voce database-> click tasto detro e cliccare su ripristina database:



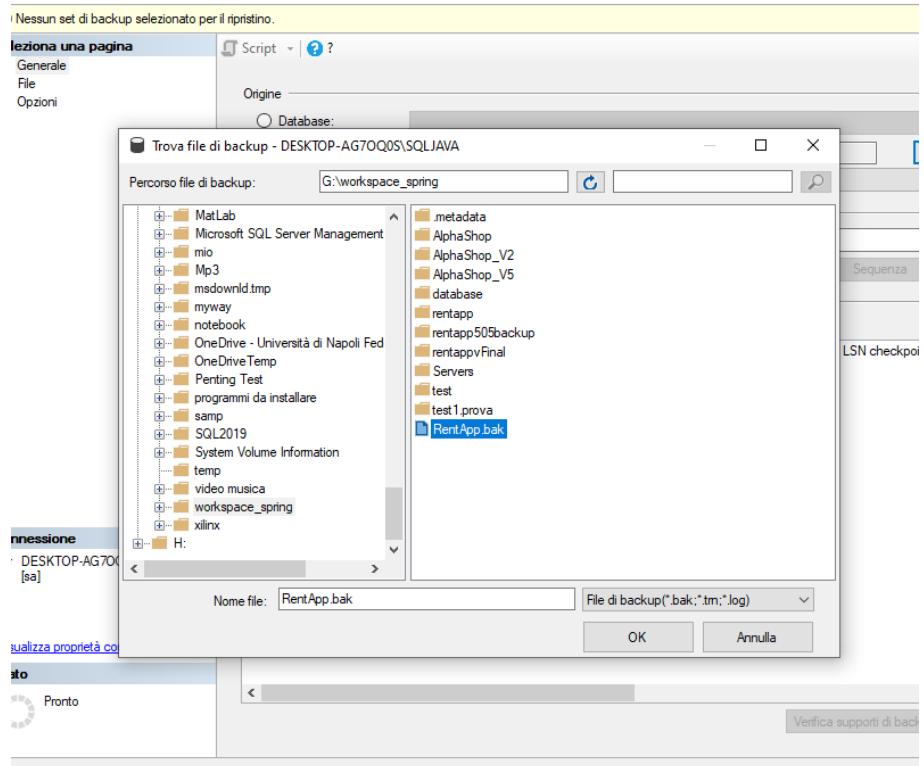
➤ Premere su dispositivo



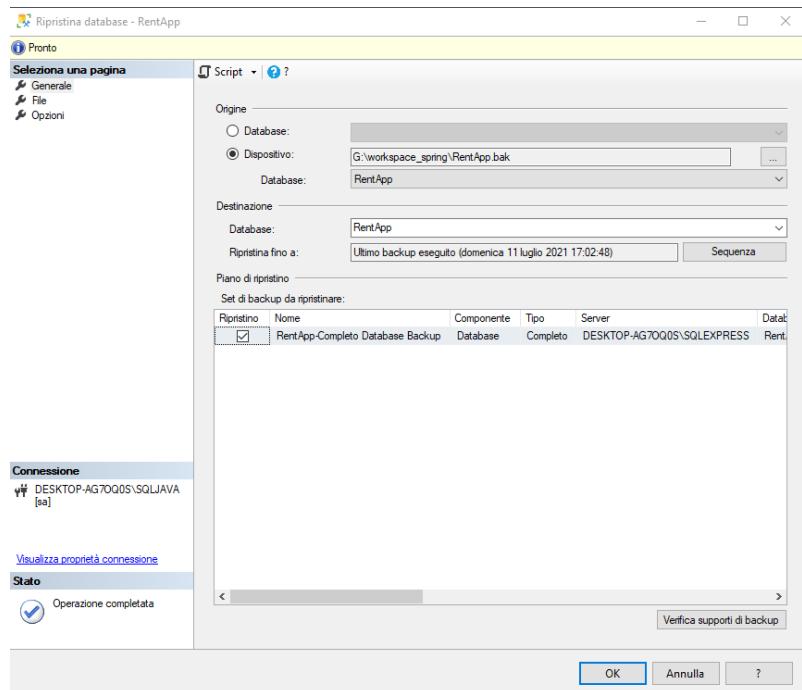
➤ Premere su aggiungi



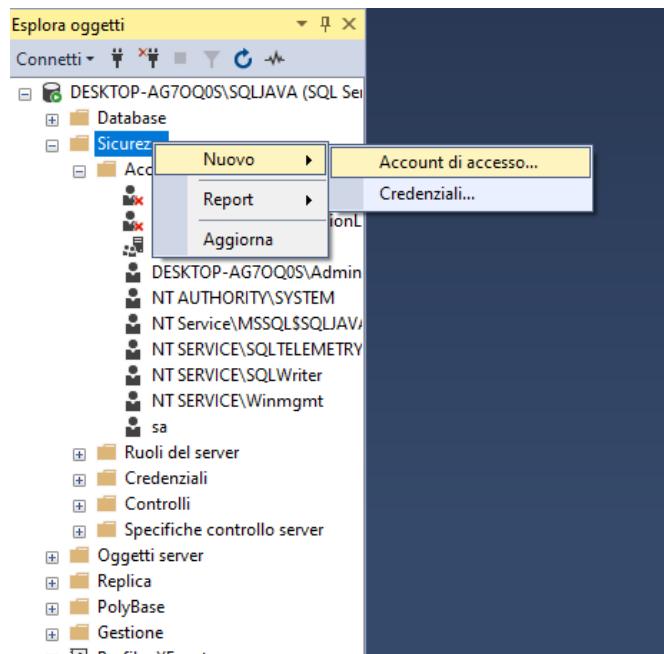
- Cercare la path dove è locato il modello del database e confermare



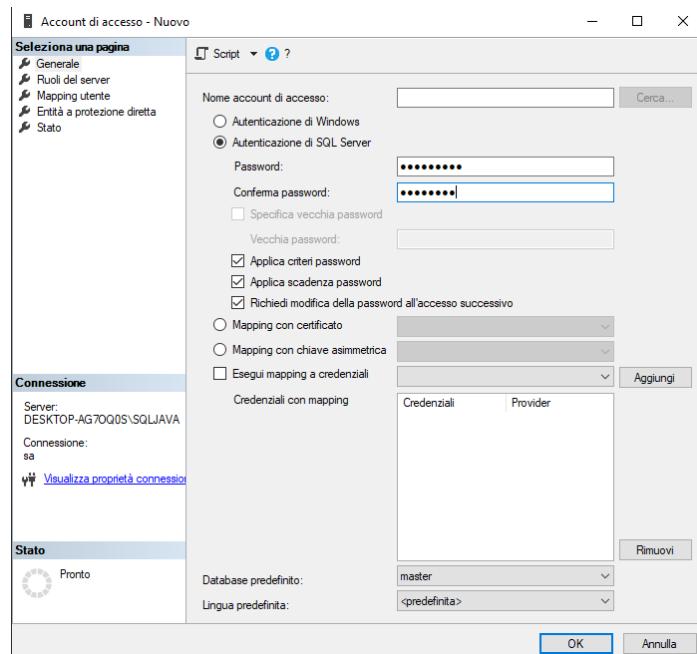
- Confermare:



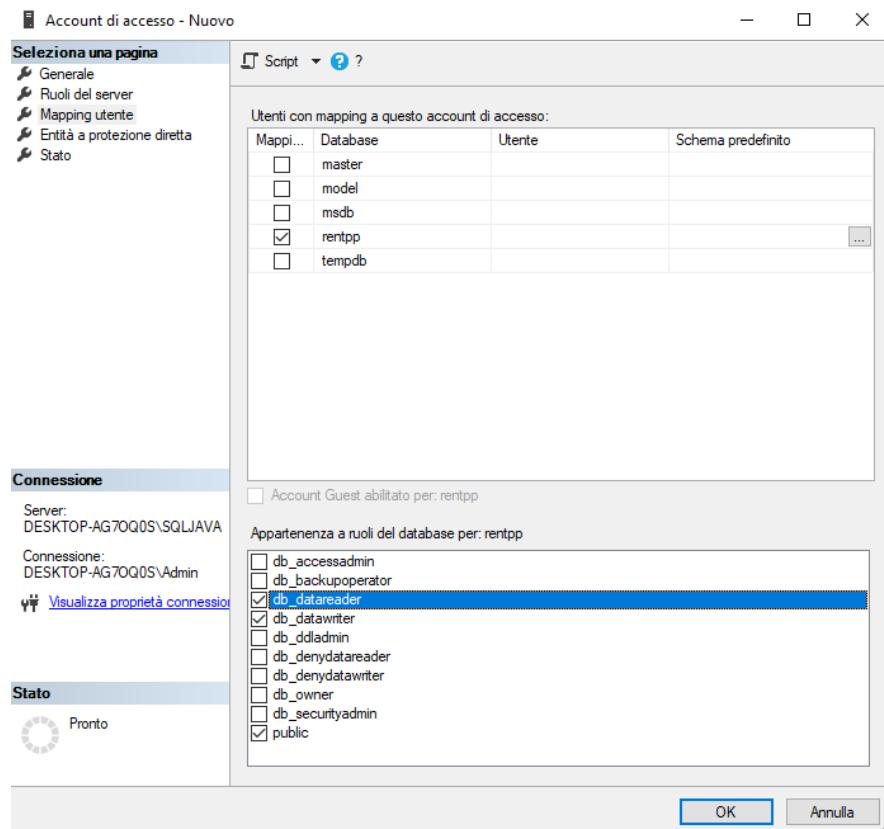
- Creazione di un utente webclient con password dedicata per l'accesso al database RentApp



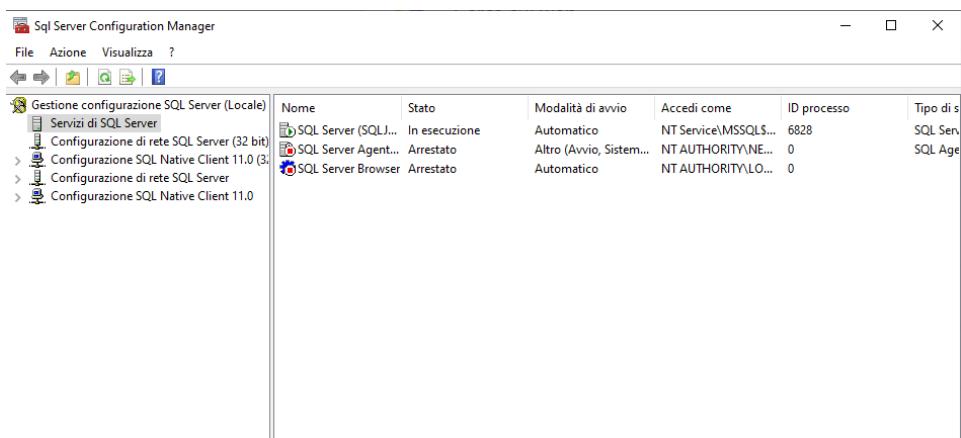
- Inserire il nome utente e la password da associare al database RentApp



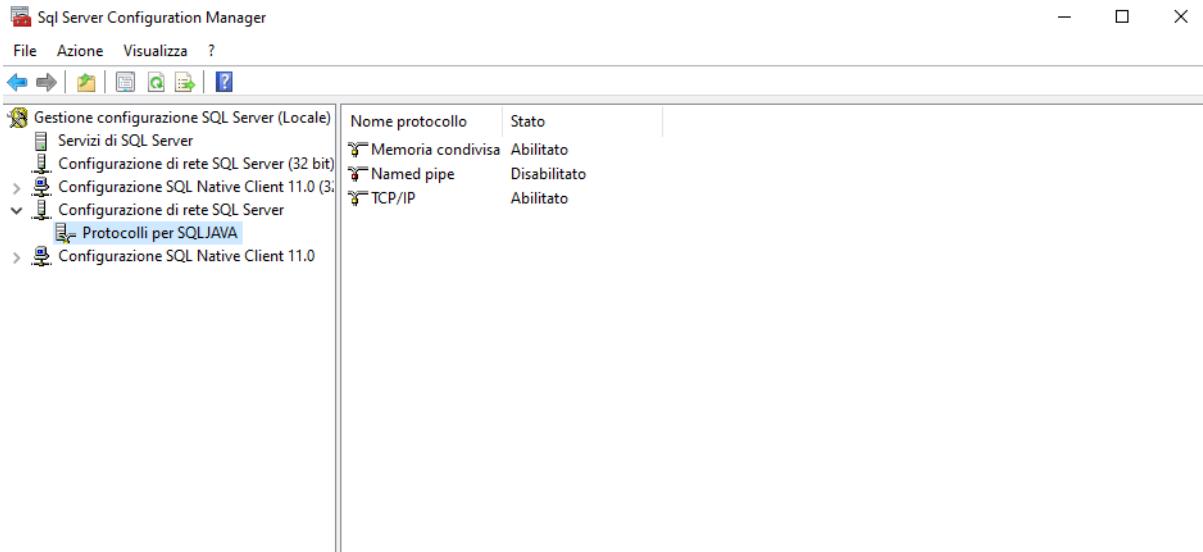
- Andare sulla voce “Mapping Utente” e selezionare il database “RentApp” ed associare in basso i permessi di lettura e scrittura, rispettivamente: “db_datareader, db_datawriter” e confermare



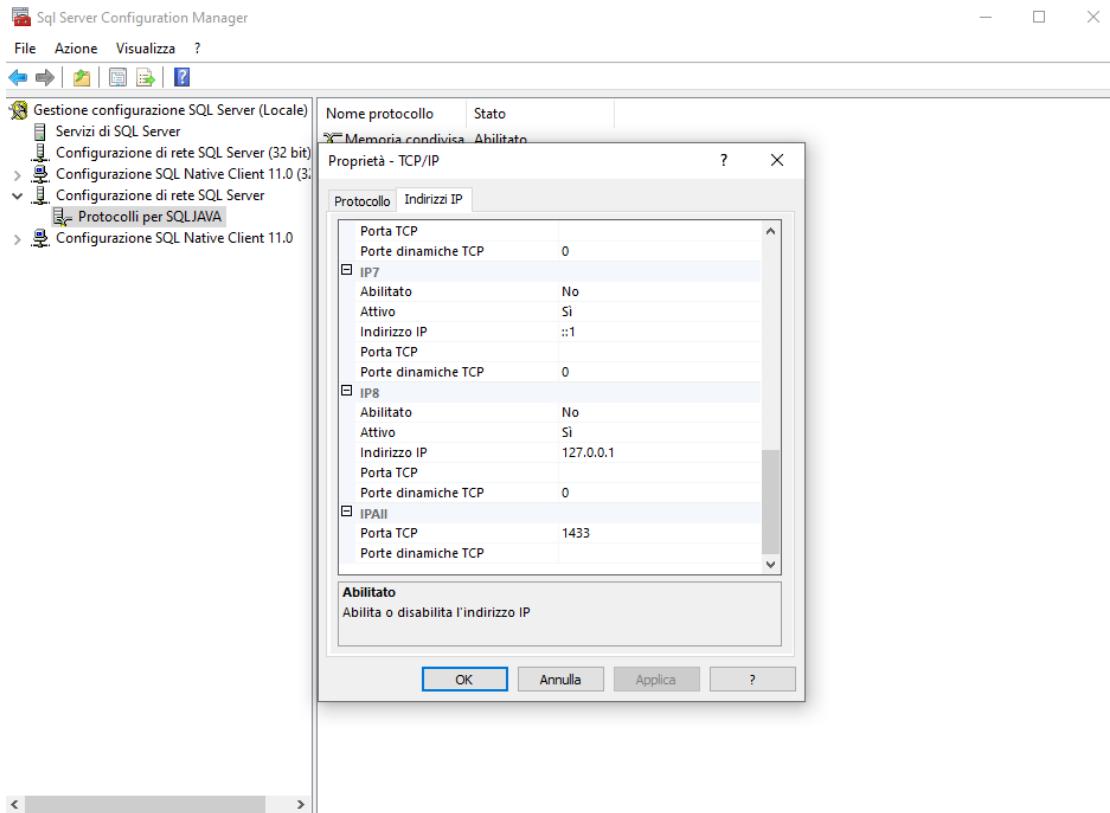
- Configurare le porte di accesso al database e i servizi che permettono la connessione, aprire il programma “Configurazione Sql Server Manager



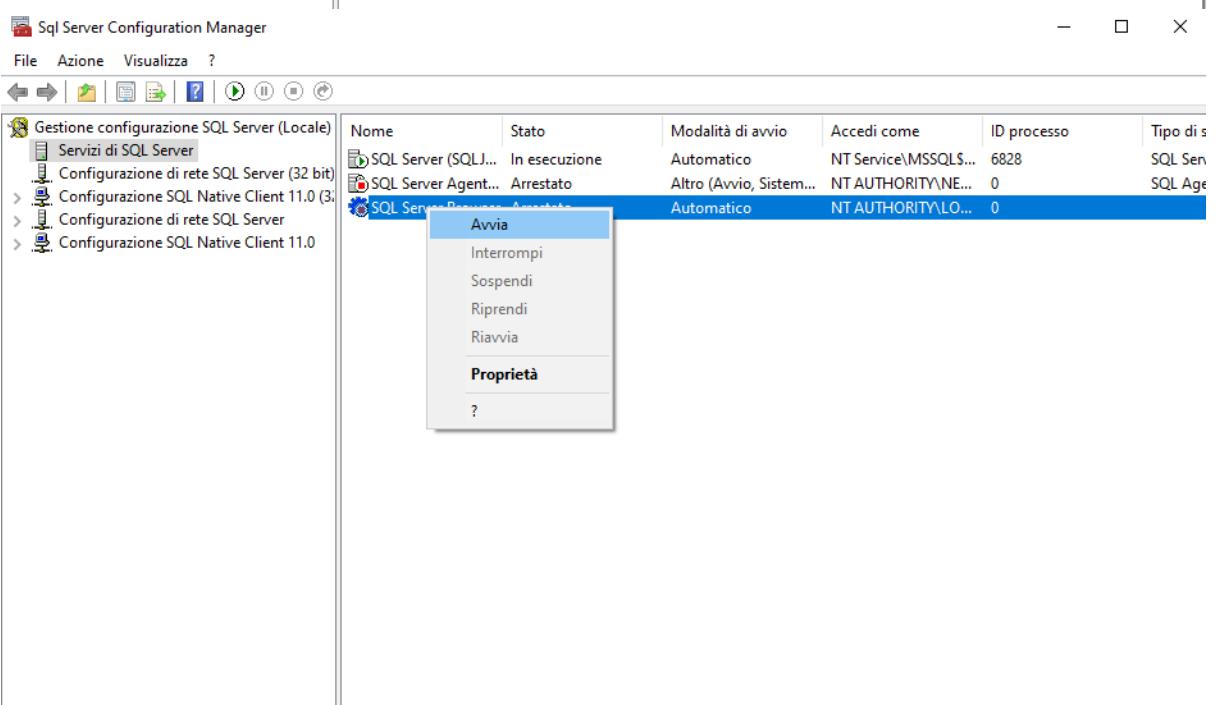
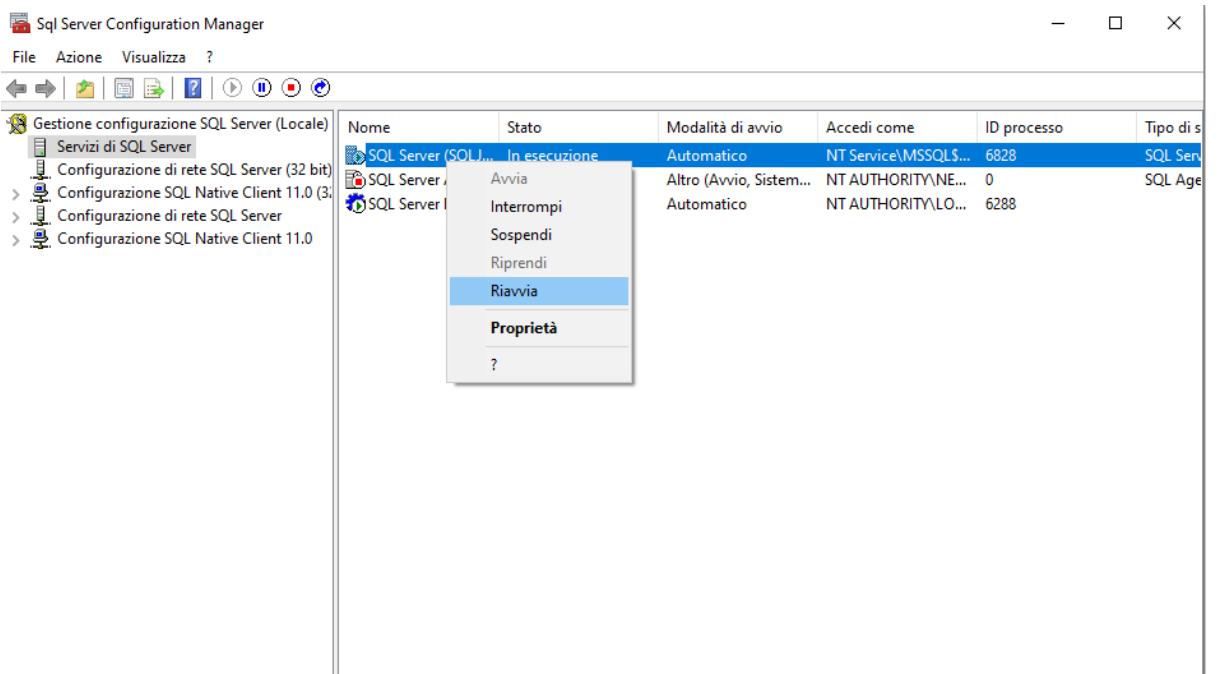
- Andare sulla voce di configurazione di rete SQL Server ed abilitare memoria condivisa:



- Premere successivamente su TCP/IP e successivamente sulla scheda indirizzi ip ed in fondo a tutto nella voce IPAll scrivere nella porta tcp il numero 1433 e successivamente confermare:



- Andare su Servizi di SQL Server e riavviare SQL Server e successivamente avviare SQL Browser



- Cambiare i settings nel file application.properties relativo al maven nel progetto inserendo l'url del proprio server SQL e l'username appena creato

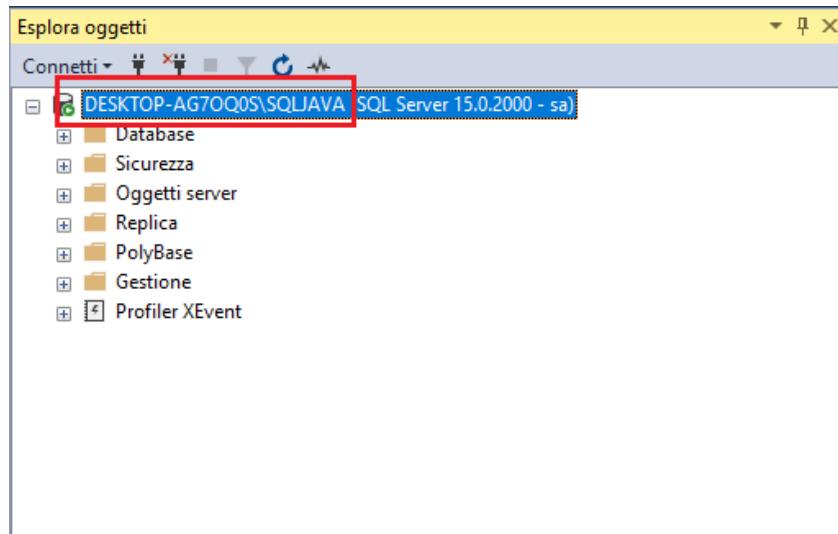


```

#JDBC
jdbc.driverClassName = com.microsoft.sqlserver.jdbc.SQLServerDriver
jdbc.url = jdbc:sqlserver://DESKTOP-AG70Q0S\SQLJAVA;database=RentApp;loginTimeout=30;
jdbc.username = WebClient
jdbc.password = root

```

Puoi trovare il nome del tuo server sull'SQL Server manager



Per una questione di compatibilità tra il nome del Web Server e la dicitura SQLEXPRESS va inserita "\\\" invece di "\\".

Se il processo di configurazione è andato a buon fine la web app riuscirà tramite Hibernate a comunicare correttamente con il Database.

CAPITOLO 10 : FUTUTURE IMPLEMENTAZIONI

Nelle successive iterazioni il team ha proposto svariati casi d'uso da progettare e implementare, di seguito si riportano i più significativi:

- “Visualizza parchi auto” : tramite questa funzionalità un gestore può visualizzare i veicoli presenti nei propri parchi auto.
- “Modifica parchi auto” : Tramite questa funzionalità un gestore può eliminare veicoli dai propri parchi auto o eliminare il parco auto con tutti i veicoli ad esso associati.
- “Segnala Guasto”: Tramite questa funzione un cliente può comunicare con un servizio assistenza per la risoluzione di guasti o per richiedere soccorso stradale.
- “Modifica profilo utente” : Tramite questo caso d'uso un utente può modificare alcuni dei propri dati personali.
- “Registra riconsegna” : Tramite questo caso d'uso un gestore può segnalare l'effettiva restituzione del veicolo all'autonoleggio.
- “Estendi periodo prenotazione”: Tramite questo caso d'uso un utente può estendere, nel caso in cui il veicolo non sia stato già prenotato, il periodo di noleggio di un veicolo.
- “Inserisci prenotazione offline” : Tramite questo caso d'uso un gestore può inserire una prenotazione per un proprio veicolo da parte di un cliente fisico che ha effettuato l'operazione all'autonoleggio, in modo da segnalare sul sito l'indisponibilità del veicolo.

Inoltre, il team ha previsto l'inserimento di nuove entità nel sistema come la tabella delle segnalazione dei guasti e la modifica di alcune di esse come lo stato del veicolo che identifica se effettivamente il veicolo è stato riconsegnato. In oltre data la sua modularità, il sistema si presta in maniera eccelsa ad interfacciarsi con sistemi esterni a cui può richiedere servizi per le future implementazioni.

Infine, nelle future implementazioni sarà inserito un nuovo attore “Addetto Servizio Clienti” il cui compito sarà quello di gestire le segnalazioni di guasti.