UNIVERSITÀ DI VERONA

DIPARTIMENTO DI INFORMATICA

Corso di Laurea Triennale in Informatica

CAUTO

Configuratore d'auto

Candidato:

Gennari Alessia (VR488137)

Moretto Mattia (VR489092)

Pasetto Michele (VR495361)

ANNO ACCADEMICO 2023/2024

Indice

Elenco delle figure

Capitolo 1

Desiderata

É richiesto lo sviluppo di un sistema informatico per la vendita online di automobili di un gruppo multi-concessionaria con la possibilità di configurare la versione di automobile desiderata.
Il gruppo vende auto di diversi modelli, raggruppati per marca. Per ogni modello di auto il sistema deve poter memorizzare:

- Un nome univoco
- Una descrizione
- Le dimensioni (altezza, lunghezza, larghezza, peso e volume del bagaglio)
- Possibili immagini che permettano di vedere l'auto da vari punti di vista con (potenzialmente diversi colori)

Il sistema dovrà registrare tutte le ruote a catalogo e definire possibili optional:

- Il colore
- Possibili aggiunte
  - Ruotino di scorta
  - Ruota di scorta
  - Vetro oscurato
  - Interni in pelle
  - Ruote con diametri maggiori di quello standard

Per ogni optional deve essere definito il prezzo e se applicabile o meno al modello scelto.
Per alcuni modelli deve poter essere applicato uno sconto che può variare da un mese all'altro e viene applicato in fase di costruzione del preventivo ed opportunamente indicato. Il cliente deve anche indicare, in caso di acquisto, dove intende ritirare l'auto.

Un generico utente anche senza autenticarsi deve poter configurare l'auto per la quale pensa di chiedere un preventivo. Successivamente nel momento in cui decide di richiedere un preventivo al gruppo è richiesto che l'utente si registri (nel caso in cui non lo fosse già) e poi effetturare l'accesso al sistema. Solo a seguito di ciò avrà quindi la possibilità di richiedere la valutazione della sua auto usata allegando delle fotografie.

Nel caso di valutazione dell'usato, il preventivo verrà finallizato da una persona del negozio, che provvederà a valutare l'usato. Il preventivo potrà essere prodotto anche come file pdf:

Il sistema memorizza i preventivi effettuati. Entro 20 giorni il potenziale acquirente può confermare il preventivo e pagare un acconto, per ordinare l'auto nella configurazione indicata. Il sistema provvederà a proporre una data di consegna (a un mese più 10 giorni per ogni optional richiesto) entro la quale avere l'auto ordinata.

Il gruppo ha diversi sedi in Italia dov'è possibile ritirare l'auto ordinata. Per ogni sede il sistema deve memorizzare nome, indirizzo completo e tutti gli ordini relativi a quella sede.

Il sistema deve permettere agli impiegati del gruppo di accedere tramite autenticazione al sistema e gestire i preventivi che richiedono la valutazione dell'usato. Gli impiegati potranno poi gestire gli ordini e avvisare i clienti quando l'auto ordinata è pronta per la consegna (dopo il pagamento dell'iporto dovuto).

La segreteria amministrativa del gruppo è responsabile dell'inserimento delle informazini su modelli e optional di ogni marca. Essa può accedere al sistema e visualizzare i preventivi per cliente, per marca e per negozio di consegna richiesto.

Il sistema deve inoltre permettere all'utente di configurare l'auto avendo un controllo del prezzo finale dell'auto ad ogni momento.

Capitolo 2

Specifiche Tecniche

2.1 Tecnologie utilizzate

Di seguito vengono elencate e descritte le principali tecnologie utilizzate per lo sviluppo del software per dare un idea generale di come è composto quanto richiesto. Queste particolari scelte progettuali sono dettate dalla confidenza e praticità nell'utilizzo da parte dei componenti del gruppo nell'utilizzarli anche in ambito lavorativo.

2.1.1 C++

Si tratta di un linguaggio di programmazione di alto livello orientato agli oggetti e fortemente tipizzato, è stato sviluppato come estensione del linguaggio C ed introdotto negli anni 80. C++ combina le caratteristiche della programmazione procedurale con quelle della programmazione orientata agli oggetti, permattendo la creazione di software più complessi, efficienti e modulari. É ampiamente utilizzato nello sviluppo di sistemi operativi, applicazioni desktop, giochi, software per dispositivi embedded e applicazioni ad alte prestazioni. Grazie alla sua flessibilità e potenza, C++ rimane uno dei linguaggi più influenti e utilizzati nel mondo della programmazione.

2.1.2 Angular

Angular è un framework open-source per lo sviluppo di applicazioni web, mantenuto da Google e basato su TypeScript, Angular facilita la creazione di applicazioni web dinamiche e reattive grazie alla sua architettura a componenti, che consente la suddivisione del codice in moduli riutilizzabili e facilmente gestibili. Tra le sue caratteristiche principali, Angular offre strumenti per il data binding, la gestione delle forme, l'iniezione di dipendenze e il routing, semplificando lo sviluppo di applicazioni complesse e scalabili. Grazie alla sua versalità e potenza Angular è una scelta popolare per la creazione di applicazioni web moderne, sia in ambito enterprise che consumer.

2.1.3 Json

JSON (JavaScript Object Notation) è un formato leggero per lo scambio di dati, facile da leggere e scrivere sia per le persone che per le macchine. Utilizzato principalmente per trasmettere dati tra un server e un client web, JSON rappresenta le informazioni in un formato testuale strutturato basato su coppia chiave-valore, simile ad un oggetto JavaScript. Grazie alla sua semplicità e compatibilità con la maggior parte dei linguaggi di programmazione, JSON è diventato uno standard ampiamente adottato per l'interscambio di dati nelle applicazioni web e nelle API. Nel nostro progetto è stato utilizzato come database.

2.1.4 Pistache

Pistache è un framework leggero e open-source scritto in C++ per lo sviluppo di server HTTP. Progettato per essere semplice ed efficiente, Pistache offre un architettura basata su eventi e supporta la creazione di server web ad alte prestazioni, in grado di gestire numerose connessioni simultanee. É particolarmente adatto per lo sviluppo di API RESTfull, grazie alla sua capacità di gestire richieste e risposte HTTP in modo intuitivo e veloce. Pistache è una scelta popolare per gli sviluppatori in C++ che cercano un framework minimalista ma potente per costruire applicazioni web scalabili e performanti.

2.1.5 Postman

Si tratta di uno strumento collaborativo per lo sviluppo e il test delle API. Consente agli sviluppatori di inviare richieste HTTP, visualizzare le risposte e automatizzare i test semplificando il processo di interazione con le API durante lo sviluppo. Con Postaman è possibile creare e gestire raccolte di richieste, configurare ambienti, variabili e documentare le API in modo efficiente. Grazie alla sua interfaccia user-friendly e alle potenti funzionalità di automazione è diventato uno strumento essenziale per lo sviluppo, debug e la gestione della API migliorando così la produttività e la collaborazione nei team di sviluppo

2.1.6 Karma

Karma è un test runner per Javascript sviluppato come parte dell'ecosistema Angular. Progettato per facilitare il processo di testing, consente di eseguire test unitari e di integrazione direttamente nel browser, garantendo che il codice funzioni correttamente in diversi ambienti. Supporta vari framework di test come Jasmine e Mocha e si integra facilmente nei flussi di lavoro e i sviluppo continuo. Con Karma,gli sviluppatori, possono automatizzare l'esecuzione dei test su più browser e dispositivi ricevendo feedback immediato sulle modifiche al codice, il che rende il processo di sviluppo delle applicazioni Angular più affidabile e efficiente

2.1.7 Jasmine

Jasmine è un framework di testing per Javascript, ampiamente utilizzato nel contesto Angular per scrivere test unitari.E'basato su un'architettura di testing behavior-driven(BDD), che consente di secrivere il comportamento del codice in modo leggibile e intuitivo. Jasmine non richiede dipendenze esterne e offre un ampia gamma di funzionalità, come gli assertion, gli spy per il monitoraggio delle funzioni e la gestione asincrona del test. Quando integrato con angular, Jasmine permette di testare i singoli componenti, servizi e moduli assicurandosi che ogni parte dell'applicazione funzioni come previsto contribuendo con alla creazione di codice affidabile a manutenibile

2.2 Formule applicate

Nello sviluppo di questo software ci siamo soffermati su due punti in particolare che richiedevano precise formule per soddisfare la desiderata. La prima riguarda la fase di configurazione dell'auto dove il prezzo finale deve sempre essere controllabile in ogni momento pertanto il calcolo che

viene fatto è il seguente:

$$p_f = p_0 + \sum_{0}^{i} p_i$$

$$p_f = Prezzo\ finale \quad p_0 = Prezzo\ base$$

$$i = Numero\ componenti\ scelti \quad p_i = Prezzo\ componente$$

Viene omesso il fatto che al prezzo finale ottenuto può essere applicato uno sconto in quanto avverrebbe soltanto in fase di creazione del preventivo e dipende fortemente dal modello e dal periodo dell'ordine

La seconda invece riguarda la data di consegna entro la quale l'utente puà ritirare l'auto ordinata che deve soddisfare il seguente presupposto:

$$d_c > date(now) + (30 + i * 10)$$

$$d_c = data\ consegna \quad i = numero\ componenti$$

Capitolo 3

Casi d'uso

I casi d'uso permettono di modellare i requisiti funzionali ossia quei requisiti che specificano cosa deve essere fatto. Essi sono indipendenti dalla tecnologia, dall'architettura, dalla piattaforma e dal linguaggio di programmazione.
I casi d'uso specificano cosa ci si aspetta da un sistema e allo stesso tempo nascondono come il sistema lo implementa. Sono una sequenza di azioni che producono un risultato osservabile da un attore e vengono utilizzati per descrivere i requisiti iniziali (analisi) e a convalidare l'architettura.
Un ottimo elemento per rappresentarli è il diagramma dei casi d'uso che procediamo quindi a sviluppare.

3.1 Diagramma dei casi d'uso

Si tratta di un diagramma che esprime un comportamento desiderato o offerto e, solitamente, l'oggetto esaminato è un sistema o una sua parte. Permette di individuare chi o che cosa ha a che fare con il sistema (attore) e cosa esso può fare.
Tipicamente il diagramma dei casi d'uso è la prima cosa ad essere creata in un processo o ciclo di sviluppo nell'ambito dell'analisi dei requisiti

Per comporre correttamente il diagramma dobbiamo prima fare un passo indietro e individuare tutti i casi d'uso che compongono il sistema.

Per sviluppare i casi d'uso è necessario prima analizzare quella che è la desiderata, ossia, in questo caso, la consegna fornita per lo sviluppo del progetto che è stata riportata nel capitolo 1. Formalizzare i desiderata individuando i requisiti è un elemento fondamentale in quanto una specifica errata o incompleta è una delle cause principali del fallimento dei progetti software.

Procedendo quindi per step andiamo ad analizzare tutto quello che ci viene richiesto, che possiamo riassumere nel seguente modo:

- Software per la vendita di auto (configurabili) online
- Un catalogo che prevede diversi modelli raggruppati per marca e con i relativi optional e immagini illustrative
- Sconti sui sigoli modelli applicabili e opportunamente indicati in fase di crazione del preventivo
- La possibilità per gli utenti che utilizzano la piattaforma di configurare in ogni momento l'auto per cui pensano di richiedere un preventivo anche senza autenticarsi e, previa autenticazione, la possibilità di richiedere il preventivo e la valutazione del suo usato

Lo step successivo è quindi quello di individuare gli attori ossia ruoli che vengono assunti dagli utenti o altre entità che interagiscono col sistema. Un attore deve essere per forza esterno al sistema inoltre non necessariamente deve essere un umano. Nello sviluppo del nostro progetto gli attori individuati sono i seguenti:

- Utente/Cliente
- Impiegati
- Segreteria

Per ognuno di questi attori è ora possibile individuare i casi d'uso ossia l'obiettivo prefissato all'inzio del capitolo e sono gli elementi che permettono la creazione del diagramma

- Il cliente consulta il catalogo, si registra e può effettuare acquisti
- Il cliente personalizza l'auto
- Il cliente richiede un preventivo e sceglie la sede per il ritiro
- l cliente può richiedere la valutazione dell'usato
- Gli impiegati si autenticano e gestiscono i preventivi che richiedono la valutazione dell'usato
- Gli impiegati gestiscono gli ordini e avvisano quando l'auto è pronta per la consegna
- La segreteria si occupa dell'inserimento delle informazioni dei modelli e optional per ogni marca
- La segreteria può visualizzare ogni preventivo per cliente, marca e per negozio di consegna
- Il cliente paga l'acconto
- Il cliente ritira l'auto

Arrivati a questo punto ci è possibile procedere allo sviluppo del diagramma del caso d'uso collegando innanzitutto i diversi attori ai relativi casi d'uso attraverso le associazioni ossia il primo elemento di collegamento.
Successivamente ci siamo occupati delle diverse dipendenze tra i casi d'uso, nel nostro caso abbiamo deciso di utilizzare:

- **Include:** dipendenza dove il caso incluso fa parte del comportamento di quello che lo include, si tratta di un comportamento obbligatorio
- **extend:** dipendenza dove il caso d'uso che estende specifica un incremento di comportamento a quello esteso, si tratta di un comportamento opzionale che gestisce casi particolari

Una volta esserci soffermati su tutti i punti siamo arrivati a creare il diagramma nella sua interezza che si presenta nel seguente modo:

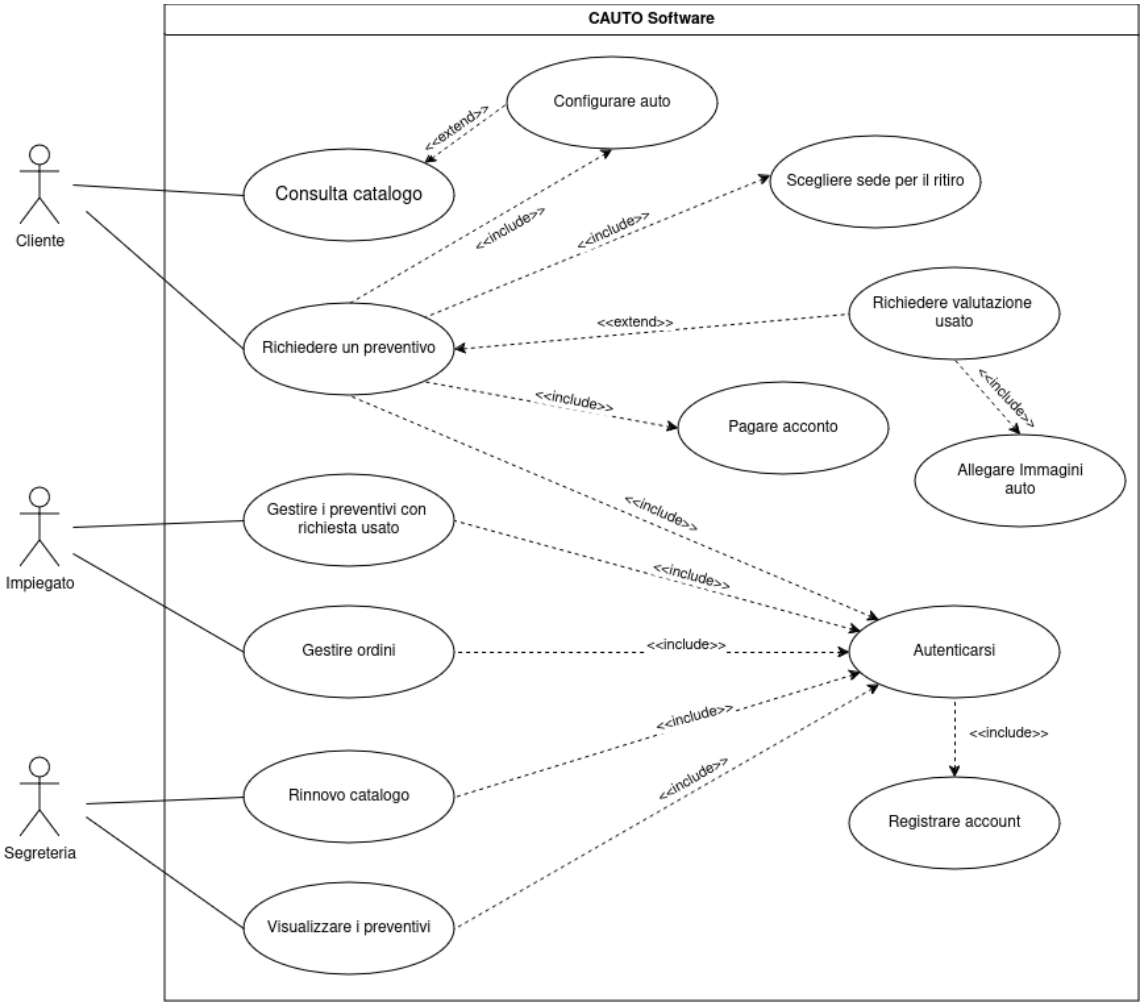


Figura 3.1: Diagramma Casi d'uso

Quanto riportato è il diagramma d'uso inteso come punto di partenza per lo sviluppo del progetto. Si ha però l'esigenza di abbinare a questo diagramma delle specifiche testuali, più formali, in quanto non sono adatti a mostrare la sequenza temporale delle operazioni oppure lo stato del sistema e degli attori prima e dopo l'esecuzione. É necessario quindi procedere con la descrizione (specifica) dei casi d'uso

3.2 Schede di specifica

Ogni caso d'uso ha un nome e una specifica, quest'ultima è composta da:

- **precondizioni:** condizioni che devono essere vere prima che il caso d'uso di possa eseguire
- **sequenza degli eventi:** i passi che compongono il caso d'uso.
- **postcondizioni:** condizioni che devono essere vere quando il caso d'uso termina l'esecuzione

Partendo quindi dagli utenti/clienti si procede ad analizzare quelle che sono le schede di specifica che lo riguardano

Consulta catalogo
ID: *CU1*
Attori*: Utente*
Precondizioni: *Nessuna*
Sequenza*:* 1. *Entrare nella pagina dedicata al catalogo delle auto* 2. *Osservare e configurare auto*
Postcondizioni*: Nessuna*

É importante evidenziare il fatto che dal consultare il catalogo non è stata specificata alcuna post-condizione in quanto per come il software è stato predisposto un utente generico ha la possibilità di consultarlo senza alcun vincolo. Il caso d'uso appena analizzato lo si può quindi definire un'azione "base" prevista dal software.
Si passa quindi ad analizzare il successivo caso d'uso che lo si può considerare quello principale all'interno di quello che è il funzionamento dell'applicazione.

Richiedere un preventivo
ID: *CU2*
Attori*: Utente*
Precondizioni: *Essersi Registrato sulla piattaforma*
Sequenza*:* 1. *Effettuare l'accesso alla piattaforma* 2. *Configurare l'auto che ha intenzione di acquistare* 3. *(a) Richiedere direttamente preventivo* *(b) Richiedere la valutazione del proprio usato* 4. *Scegliere la sede per effettuare il ritiro dell'auto* 5. *Attendere finalizzazione del preventivo* 6. *Entro 20 giorni confermare il preventivo*
Postcondizioni*: Pagare acconto*

Elaborate le schede di specifica legate agli utenti si procede ad analizzare quelle relative agli impiegati

Gestire i preventivi con richiesta usato
ID: *CU3*
Attori*: Impiegato*
Precondizioni: 1. *Un cliente ha richiesto la valutazione dell'auto usata durante la richiesta di un preventivo* 2. *Essere registrato alla piattaforma*
Sequenza*:* 1. *Effettuare l'accesso alla piattaforma* 2. *Accedere alla pagina dedicata con la lista dei preventivi da finalizzare* 3. *Procedere a valutare l'usato*
Postcondizioni*: Finalizzare il preventivo*

Gestire ordini
ID: *CU4*
Attori*: Impiegato*
Precondizioni: 1. *Essere registrato alla piattaforma* 2. *Acconto del preventivo pagato dall'utente* 3. *Terminato tempo attesa per la data di consegna*
Sequenza*:* 1. *Effettuare l'accesso alla piattaforma* 2. *Accedere alla pagina dedicata con la lista dei preventivi* 3. *Procedere ad avvisare l'utente che l'auto è pronta*
Postcondizioni*: Consegna l'auto nella sede indicata*

Terminate le schede di specifiche che riguardano gli impiegati si procede con quelle legate alla segreteria

Rinnovo catalogo
ID: *CU5*
Attori*: Segreteria*
Precondizioni: *Essere registrato alla piattaforma*
Sequenza*:* 1. *Effettuare l'accesso alla piattaforma* 2. *Accedere alla pagina dedicata con il catalogo delle auto* 3. *Inserimento di informazioni su modelli e optional di ogni marca*
Postcondizioni*: Nessuna*

Il caso d'uso che riguarda il rinnovo del catalogo è un'azione fine a se stessa, nel senso che si tratta di qualcosa che viene svolto dalla segreteria come parte del proprio lavoro, da questa azione non ne consegue nessun'altra ma è importante per avere il catalogo sempre aggiornato e influisce sui casi d'uso degli altri attori. Lo stesso vale per la prossima scheda, si potrebbe definire la segreteria come un attore passivo.

Visualizzare i preventivi
ID: *CU6*
Attori*: Segreteria*
Precondizioni: *Essere registrato alla piattaforma*
Sequenza*:* 1. *Effettuare l'accesso alla piattaforma* 2. *Accedere alla pagina dedicata con la lista dei preventivi* 3. *Visualizzare i preventivi per cliente, marca e negozio di consegna richiesto*
Postcondizioni*: Nessuna*

Capitolo 4

Diagramma della classi

Un diagramma delle classi è un tipo di diagramma utilizzato nella modellazione ad oggetti. Serve per rappresentare la struttura statica di un sistema mostrando le classi, gli attributi, i metodi e le relazioni tra di esse come l'ereditarietà, l'associazione e la composizione. Si tratta di uno strumento fondamentale per la progettazione del software poichè consente di visualizzare e definire chiaramente l'architettura del sistema, facilitando la comprensione, la manutenzione e l'implementazione del codice.

Si tratta del secondo step che abbiamo deciso di sviluppare in quanto serviva a dare un'idea generale della struttura del processo e permettere una stesura iniziale del codice in base alla metodologia che abbiamo deciso di adottare descritta nel capitolo dedicato. All'interno del diagramma ogni classe viene rappresentata in forma di un rettangolo, che può avere fino a 3 slot:

- nome e l'eventuale stereotipo (in UpperCamelCase, obbligatorio)
- attributi (in lowerCamelCase, opzionali)
- operazioni (in lowerCamelCase, opzionali)

Ogni classe a sua volta si divide in:

- **classi di analisi:** solitamente contengono solo quelli più importanti e spesso specificano solo il nome
- **classi di progettazione:** forniscono una specifica completa (implementabile) della classe e dei suoi attributi

Per rappresentare invece le istanze delle classi si utilizza una notazione molto simile, quinid avranno anch'esse una forma rettangolare. Le differenze sostanziali stanno:

- Il titolo degli oggetti è sottolineato e nella forma "nome:classe" con nome opzionale
- Non hanno uno slot per le operazioni, possono invece definire valori per gli attributi

Fatta questa premessa si procede a mostrare mostrare il diagramma delle classi nella sua interezza e successivamente verrà fornita una breve descrizione di come è stato composto.

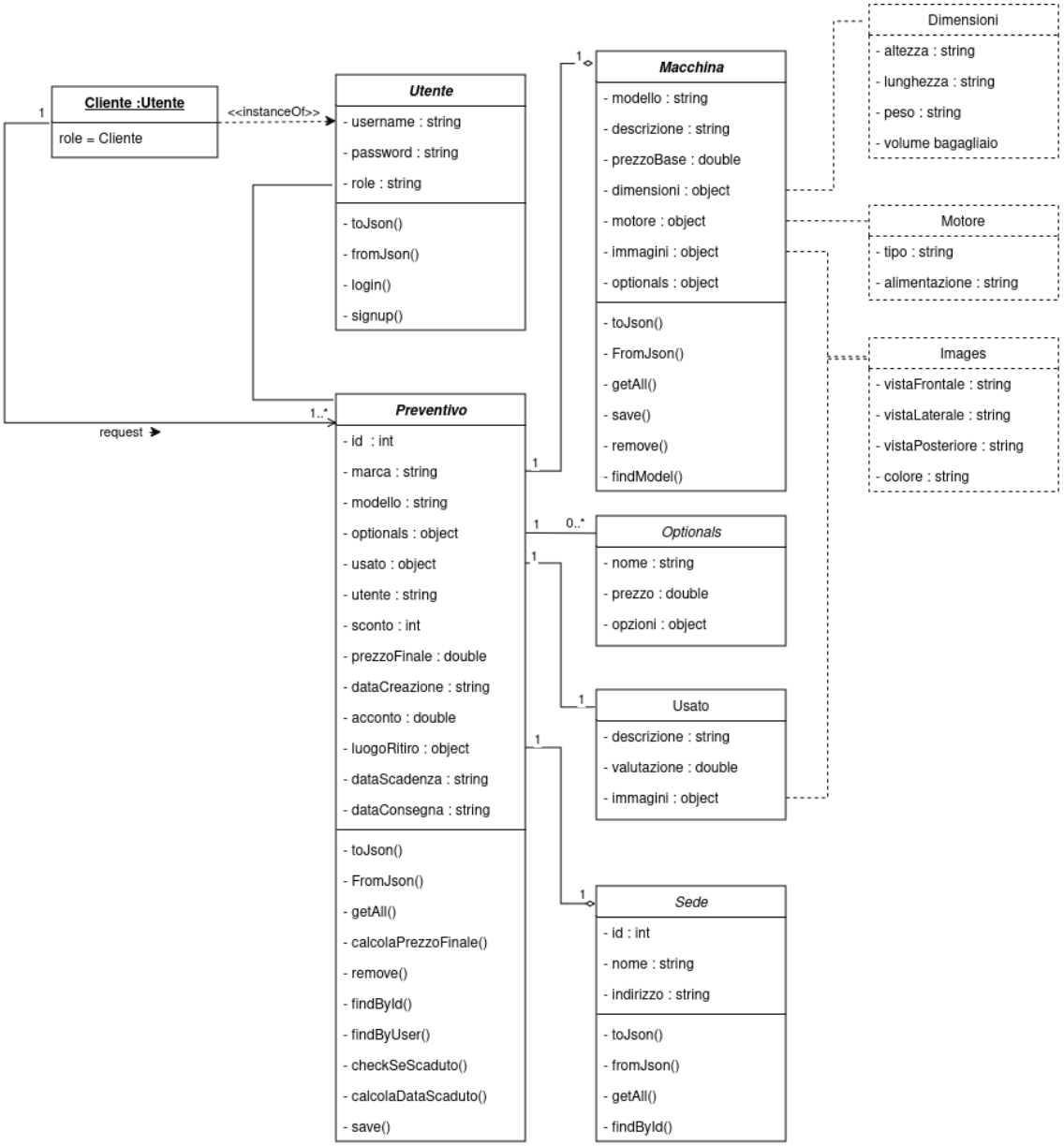


Figura 4.1: Diagramma delle Classi

All'interno del diagramma sono state riportate tutte le classi inserite all'interno del codice. Per completezza abbiamo deciso di mostrare anche tutte le operazioni e gli attributi che abbiamo creato, inoltre per scelte progettuali le abbiamo rese tutte pubblico, come si può vedere dal simbolo riportato accanto al nome
Per quanto riguarda invece i tipi di collegamento abbiamo scelto di utilizzare:

- **«instanceOf»:** Ábbiamo deciso di rappresentare una istanza per la classe utente, dove si va a passare l'attributo "role". questo è stato pensato per indicare la registrazione sulla piattaforma da parte dell'utente. A differenza degli altri due non ha possibilità di scegliere, il ruolo deve essere sempre definito
- **Associazione:** si tratta di un tipo di relazione generico che va ad indicare l'esistenza di un collegamento tra le istanze delle classi

- **Dipendenza:** si tratta di una forma più debole di relazione tra classi e operazioni
- **Aggregazione:** si tratta di una relazione poco forte dove le due parti possono esistere anche da sole ed inoltre ognuna di esse può appartenere a più aggregazioni

Dall'immagine si possono vedere dei riquadri leggermente diversi che non sono propriamente elementi del diagramma delle classi per questo motivo abbiamo deciso di rappresentarli solamente con linee tratteggiate. Questa scelta è stata fatta in quanto alcune classi sono state composte con degli object e per poter dare chiarezza su come è veramente composta la classe. Quindi i blocchi in questione sono da intendere come un espansione dell'elemento object di riferimento che mostra gli attributi di cui è composto

Capitolo 5

Diagrammi d'interazione

Si tratta di diagrammi di comportamento che modellano le interazioni tra varie entità di un sistema inoltre permettono di visualizzare lo scambio di messaggi trà entità nel tempo. Il loro scopo principale è quello di mostrare come un certo comportamento viene realizzato dalla collaborazione delle entità in gioco.
Sostanzialmente permette di illustrare il comportamento del sistema da una prospettiva interna inoltre possono avere diversi livelli di astrazione.
Per comporre un diagramma di interazione sono necessarie due cose:

- Un comportamento da realizzare tratto da un classificatore di contesto come per esempio:
  - Un caso d'uso
  - im'operazione di classe
- Una serie di elementi che realizzano il comportamento come:
  - attori
  - istanze di classe

Generalmente nei diagrammi di interazione non compaiono direttamente classificatori come le classi, al loro posto ci sono:

- **Istanze** di classificatori (oggetti, istanze di attori)
  - rappresenta uno specifico oggetto, e solo quello.
- **Linee di vita** (lifeline) di classificatori
  - è più generale.
  - rappresenta un'istanza arbitraria di un classificatore.
  - fornisce modi per specificare come quest'istanza viene scelta
  - esprime il ruolo giocato da un'istanza senza preoccuparsi della sua identità

La differenza tra questi due elementi è sottile ma, in generale, è preferibile usare le linee di vita

In tutto ci sono 4 tipi di diagrammi di interazione ma andremo ad elaborare solamente il **diagramma di sequenza** che, come dice il nome, è adatto a mostrare la sequenza temporale degli avvenimenti per ogni entità nel diagramma e cominciamo sviluppando quelli relativi ai casi d'uso presi singolarmente

5.1 Casi d'uso

Per quanto riguarda i casi d'uso del Cliente viene omessa la consulta del catalogo in quanto l'iterazione con il sistema è veramente minima se non quasi nulla pertanto anche rappresentandola tramite diagramma non mostrerebbe molte più informazioni rispetto a quelli precedentemente trattati. Si procede quindi a mostrare quello riguardante la creazione del preventivo

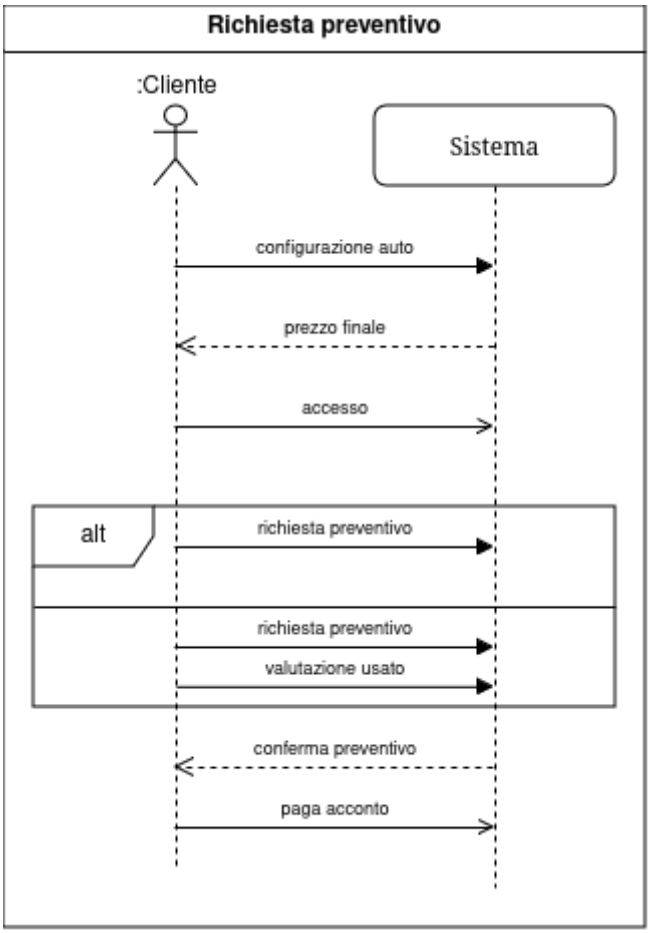


Figura 5.1: Diagramma di sequenza Cliente

A livello temporale si è cercato di esprimere in sequenza come dovrebbe avvenire il processo per la richiesta di un preventivo da parte di un cliente. Innanzitutto l'utente può consultare liberamente ogni auto a catalogo e ha la facoltà di configurarla, durante questo processo il sistema dovrà fornire in ogni momento il prezzo finale dell'auto. Una volta scelta e configurata l'auto il cliente può procedere con l'acquisto ma prima di poter fare ciò deve essere per forza autenticato. Da qui in avanti avrà dunque la possibilità di procedere in due strade, la prima è quella di richiedere direttamente il preventivo, nella seconda invece può scegliere anche di richiedere la valutazione del proprio usato. Per entrambi i casi l'ultimo passaggio che rimane da fare è quello di attendere di ricevere conferma da parte del sistema dopo la quale l'utente può procedere a pagare l'acconto dovuto ed attendere che l'auto sia pronta per poterla andare a ritirare nella sede da lui specificata.

Si procede dunque a mostrare il diagramma creato per gli Impiegati.

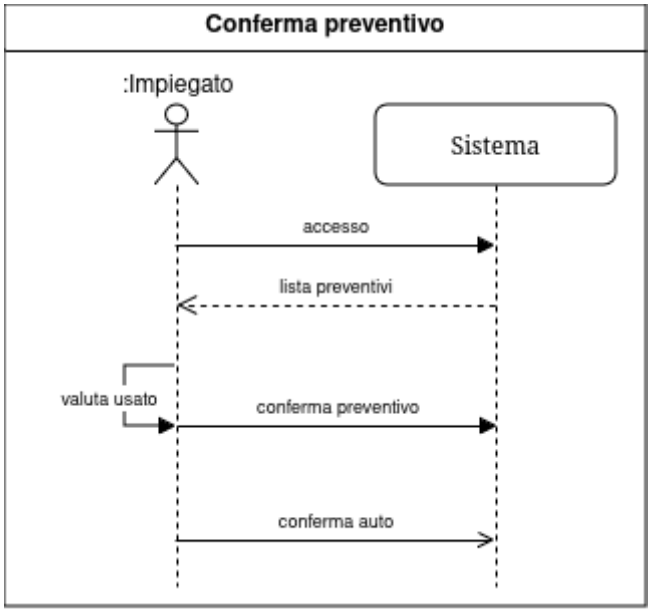


Figura 5.2: Diagramma di sequenza Impiegato

In questo caso il passaggio che avviene è più semplice, una volta che l'impiegato effettua l'accesso alla piattaforma, quest'ultima fornirà la lista di tutti i preventivi che sono in attesa di conferma ossia quei preventivi nel quale è stata richiesta la valutazione dell'usato. Quello che deve fare l'impiegato è quindi controllare le immagini che sono state fornite e stimare un prezzo per l'auto usata, dopodichè può procedere con la conferma del preventivo. Una volta fatto questo non gli resta altro che avvisare il cliente quando l'auto è pronta per il ritiro

Procediamo infine a mostrare il diagramma creato per la segreteria.

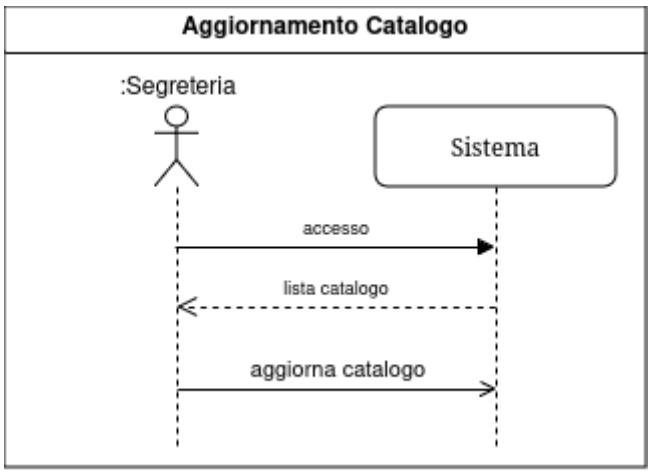


Figura 5.3: Diagramma di sequenza Segreteria

Anche in questo caso la procedura è molto semplice, la segreteria effettua l'accesso alla piattaforma che provvederà a fornire la lista di tutte le auto a catalogo. Avrà quindi la possibilità di aggiornare le auto già presenti oppure aggiungere nuove auto.

5.2 Sistema

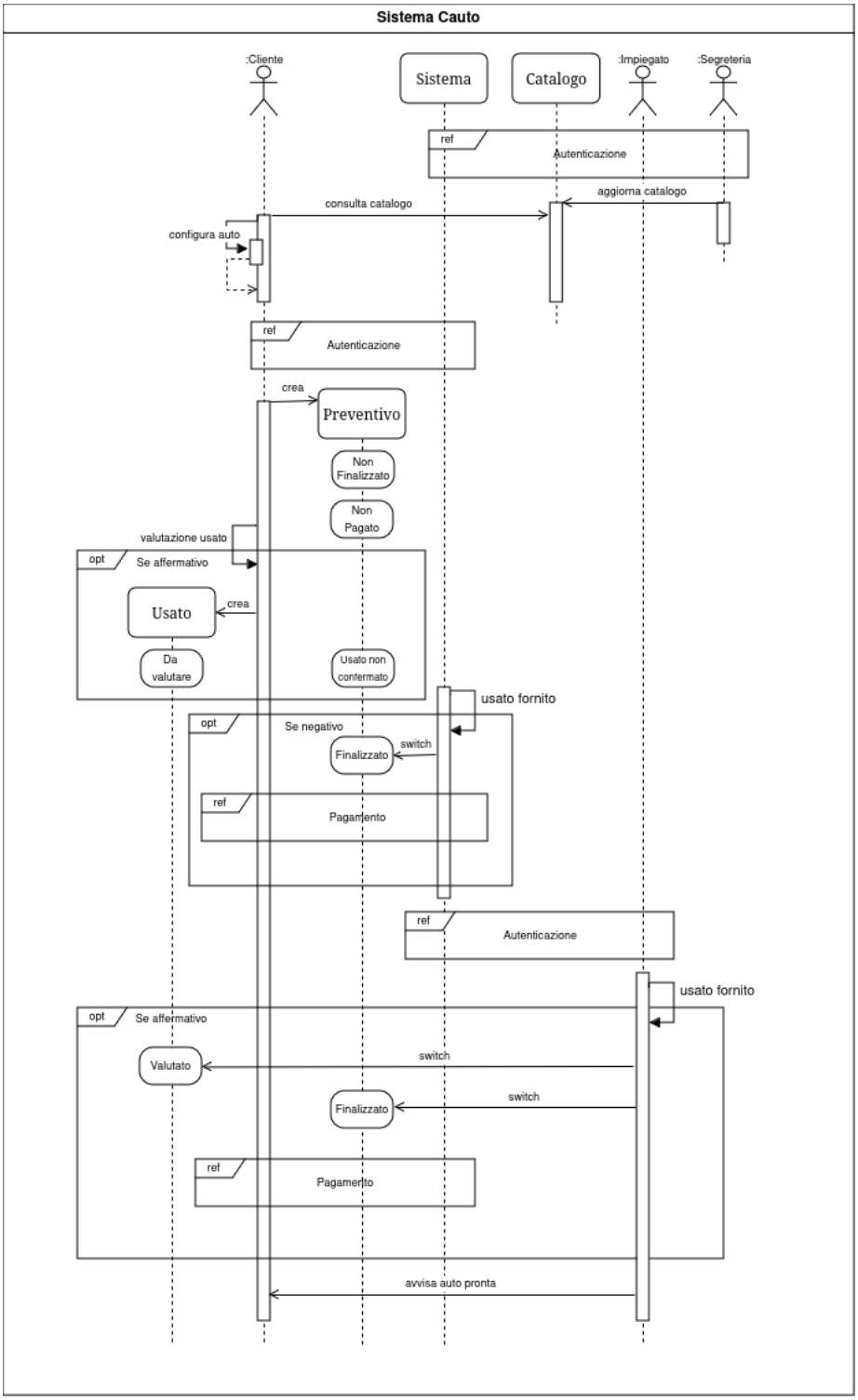


Figura 5.4: Diagramma di sequenza sistema

Quanto riportato è il diagramma di sequenza che rappresenta il funzionamento dell'intero sistema a livello temporale di quello che accadrebbe dall'inizio alla fine nello svolgimento della sua funzione

principale, mentre di seguito vengono riportati i diagrammi che fanno riferimento all'operatore ref utilizzato nell'immagine soprastante.

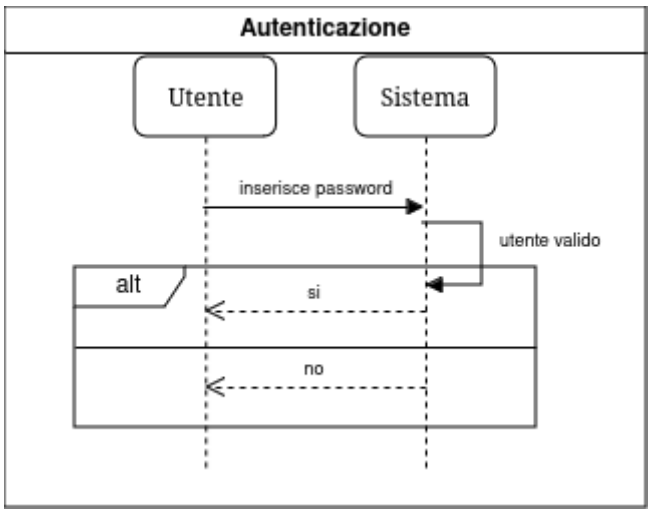


Figura 5.5: Diagramma di sequenza autenticazione

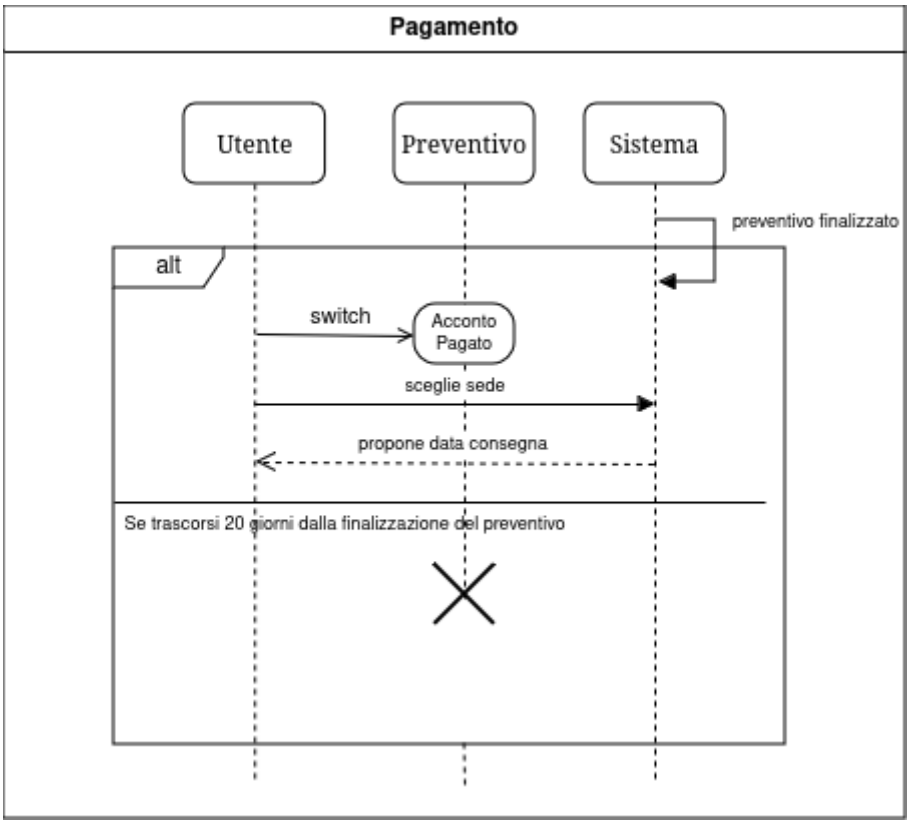


Figura 5.6: Diagramma di sequenza pagamento

Con il diagramma riportato nella figura 5.4 abbiamo messo in evidenza come le varie entità collaborino nel fornire l'output richiesto dalla desiderata

Nella costruzione di tutti i diagrammi di attività riportati abbiamo utilizzato:

- **Messaggi:**
  - chiamata sincrona: ossia quei messaggi che bloccano chi li manda in attesa di un messaggio di ritorno del destinatario
  - chiamata asincrona: dove il mittente non si aspetta una risposta e prosegue immediatamente
  - chiamata di ritorno: anche se opzionali abbiamo scelto di utilizzarli ogni qual volta è stata definita una richiesta sincrona per indicare cosa ci si aspetta per poter proseguire
  - creazione: come dice il nome duranta la sequenza servono ad indicarela creazione di un nuovo oggetto
  - distruzione: similmente a quello di creazione indica la distruzione dell'oggetto, una volta arrivati a quel punto non sarà più accessibile
- **Attivazioni annidate:** le linee vita mandano messaggi a se stesse si tratta di operazioni frequenti per indicare per esempio invocazioni di operazioni private
- **stato:** è definito come una condizione o situazione durante la vita dell'oggetto in cui esso soddisfa una condizione, uno stato o aspetta un evento
- **operatori:** determina la semantica dell'esecuzione di un frammento combinato ossia una sottoarea che racchiude una parte dell'interazione e le modalità della sua esecuzine, quelli utilizzati sono:
  - opt: corrisponde a if/then e accetta solo un operando che viene eseguito se e solo se la guardia è valutata true
  - alt: corrisponde a case/select e accetta un qualunque numero di operandi e ne esegue al più uno
  - ref: rende i diagrammi di sequenza modulari, l'operatore contiene il nome di un'altra interazione che viene eseguita qui

Capitolo 6

Diagrammi di Stato e attività

Si tratta di diagrammi che permettono di descrivere il comportamento, procediamo ad analizzarli singolarmente

6.1 Diagramma di stato

Serve a modellare il comportamento (generalmente di una sola entità) come variazione del suo stato interno. Come dice il nome esso permette di descrivere la sequenza di stati in cui si trova un oggetto durante il suo ciclo di vita e in risposta agli eventi.
In UML qualunque oggetto può essere associato a una macchina a stati che descrive il funzionamento delle sue istanze ed è caratterizzato da 3 elementi:

- Uno **stato**, ossia una condizione o situazione nella vita di un oggetto in cui esso:
  - soddisfa una condizione
  - esegue un'attività
  - aspetta un evento
- Un **evento**, ossia la specifica di un'occorrenza che ha una collocazione nel tempo e nello spazio
- Una **transizione**, ossia il passaggio da uno stato a un altro in risposta ad un evento. Oltre allo stato d'origine e destinazione una transizione può specificare
  - **event:** un trigger che attiva il passaggio di stato
  - **guard:** una condizione che, se vera, permette il passaggio di stato
  - **action:** un'azione che risulta dal cambio di stato

Procediamo quindi ad elaborare i due diagrammi di stato che abbiamo deciso di sviluppare

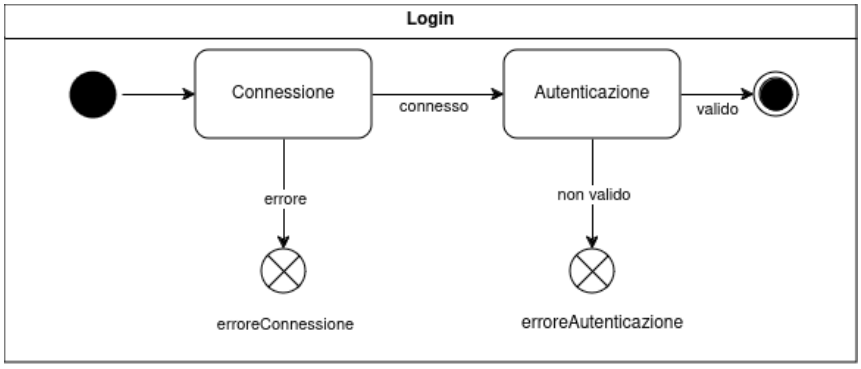


Figura 6.1: Diagramma di stato autenticazione

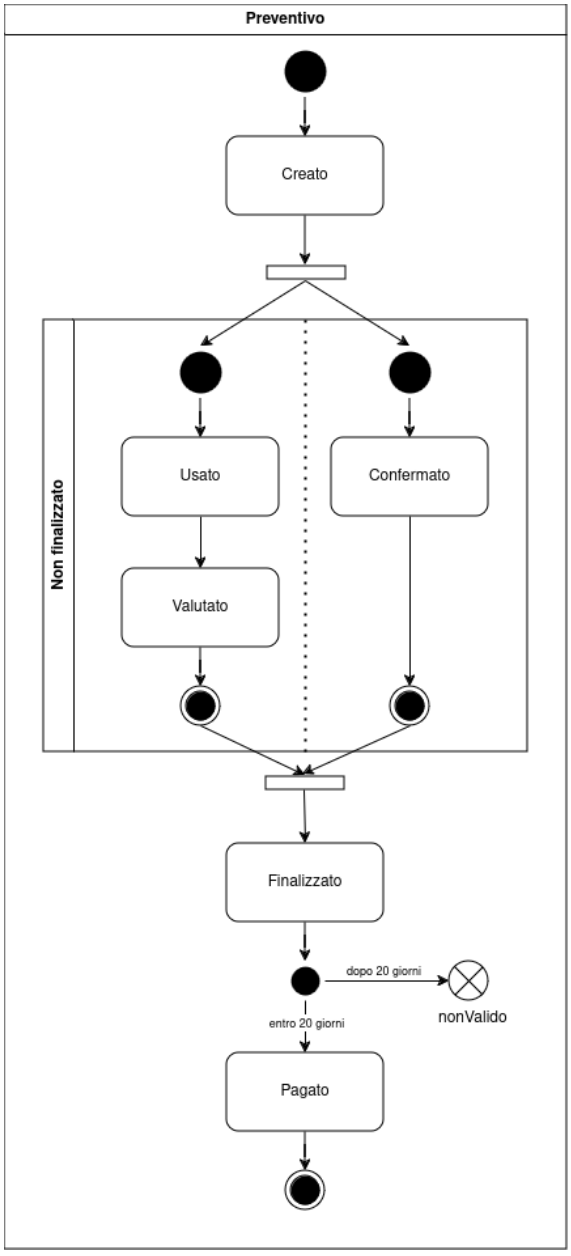


Figura 6.2: Diagramma di stato preventivo

Nella costruzione del diagramma sono stati utilizzati solamente gli elementi base, l'unico elemento particolare che è stato aggiunto per il preventivo è lo **stato composito e sincronizzazione** che permette di suddividere la complessità del modello. Dall'esterno si vede un macro-stato al cui interno però ci sono altri stati. Con la sincronizzazione invece si divide lo stato composito in sotto-diagrammi e si utilizzano gli operatori fork e join.

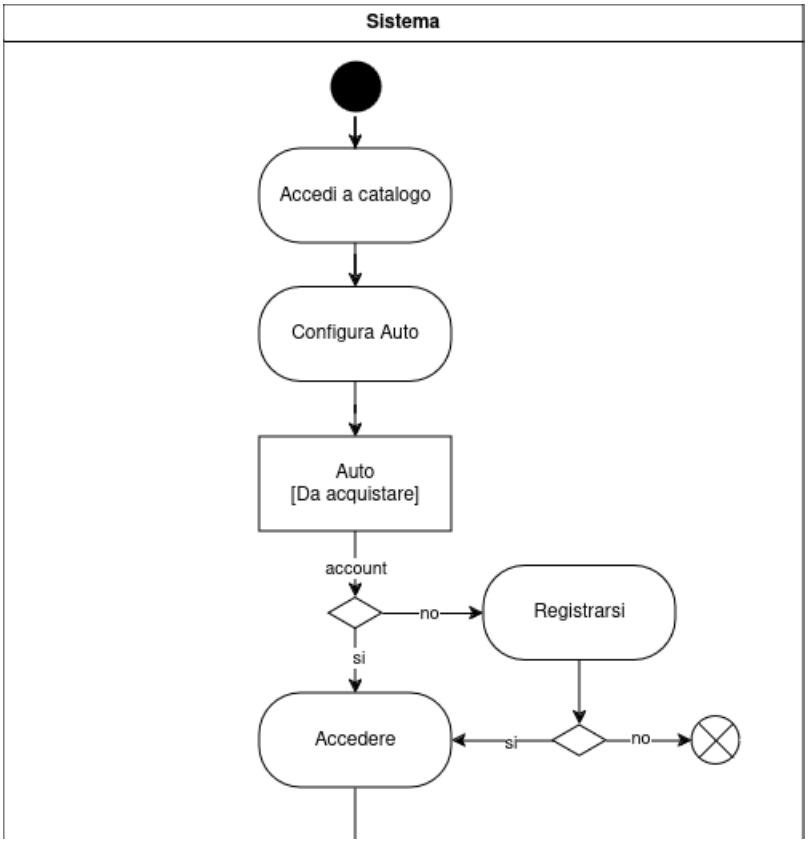
6.2 Diagramma di attività

I diagrammi di attività servono per modellare un attività relativa ad un qualsiasi oggetto,tra i suoi usi più comuni troviamo il modellare il flusso di un caso d'uso, modellare il funzionamento di un operatore di classe e modellare un algoritmo e sono composti da tre elementi principali:

- Nodi **azione:** specificano unità di comportamento e possono invocare un'attività, un comportamento o un'operazione
- Nodi **oggetto:** specificano oggetti usati come input e output di azioni
- Nodi **controllo:** specificano il flusso delle attività.

Per capire la semantica dei diagrammi di attività bisogna immaginare delle entità chiamate token, che viaggiano lungo il diagramma. Il flusso di questi token definisce quello dell'attività inoltre essi possono rimanere fermi in un nodo azione/oggetto in attesa che si avveri una condizione su una freccia, oppure una precondizione o postcondizione su un nodo. Un nodo azione viene eseguito quando sono presenti token su tutti gli archi in entrata e tutte le precondizioni sono soddisfatte. Quando invece termina un azione vengono generati control token su tutti gli archi in uscita.

Procediamo a mostrare il suddetto diagramma:



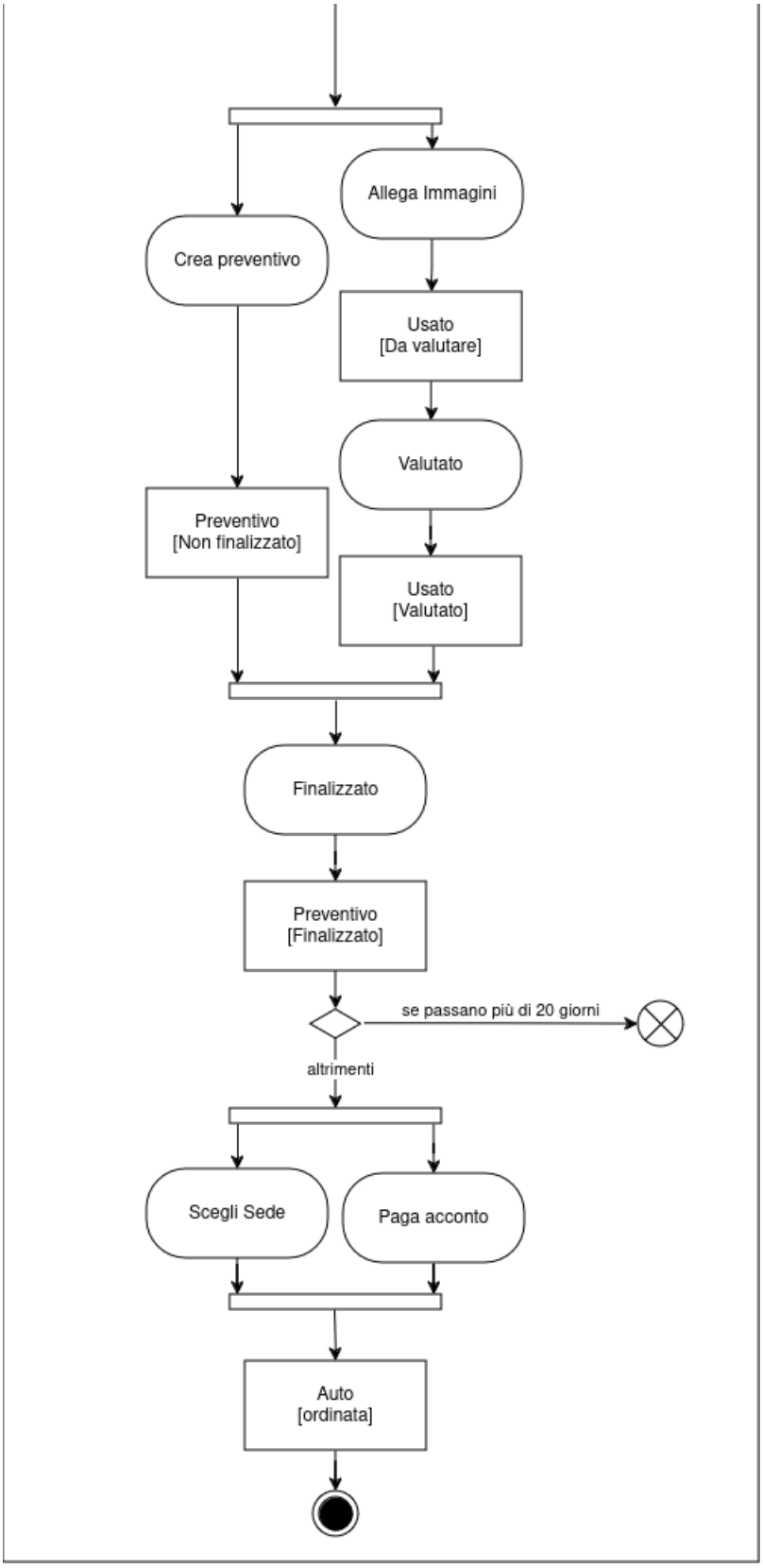


Figura 6.3: Diagramma di attività

Oltre a quanto già riportato, nella creazione del diagramma sopra riportato abbiamo utilizzato dei particolari nodi di controlli ossia:

- **nodi iniziali/finali:** classici nodi del diagramma, il disco nero marca l'inizio dell'attività e quindi la generazione del token mentre quello nero bordato indica che l'attività ha termine
- **nodi di decisione:** hanno un input e vari output mutuamente esclusivi
- **nodi fork:** hanno un ingresso e varie uscite, ciò corrisponde al token in ingresso che viene duplicato verso tutte le uscite
- **nodi join:** hanno vari ingressi e una sola uscita, quando sono presenti token su tutti gli ingressi, viene prodotto almeno un token in uscita
- **nodi finali di flusso** quando raggiunti da un token, causano la terminazione solo del flusso che li ha toccati

Arrivati alla conclusione del capito si può notare come i diagrammi di stato e di attività siano in pratica la stessa cosa semplicemente usati in modo diversi. Sostanzialmente i diagrammi di attività sono diagrammi di stato in cui gli stati vengono convertiti in azioni. Infatti come si può notare, confrontando quanto riportato si può notare una certa similitudine nella struttura e negli elementi tra i vari diagrammi

Capitolo 7

Test del Software

Per verificate la solidità, l'efficenza, la chiarezza e la precisione nel soddisfare la richiesta del software prodotto sono state eseguite le seguenti attività:

1. Revisione della desiderata e del documento creato
2. Verifica della consistenza tra diagrammi e codice prodotto
3. Ispezione del codice per verificare la corretta struttura e ricerca errori a livello di leggibilità
4. Unit test eseguito con l'ausilio di Karma e Jasmine per il Front End e Postman per il Back End
5. Test da parte degli sviluppatori del software
6. Test da parte di un utente esterno con raccolta feedback

L'esecuzione di questi test ha portato a modifiche, migliorie e nuove implementazioni del software che a conti fatti corrisponderebbe a diverse versione di una beta prima del rilascio/vendita effettiva del software che possono essere riassunti nel seguente modo

Versione	Descrizione
v0.1	Si tratta di una versione base durante la creazione del documento per portarsi avanti con lo sviluppo dove erano presenti semplici elementi comuni a tutte le interfaccie
v0.2	Si tratta della versione prodotta una volta concluso il documento e quindi tutta la parte di analisi e progettazione
v0.3	Si tratta della versione pordotta dovo aver terminato i diversi Unit Test
v0.4	Si tratta della versione prodotta dopo aver concluso i test da parte degli sviluppatori
v1.0	Si può considerare la prima release del software dovo aver eseguito i test da parte degli utenti esterni. Si tratta delle versione del software presentata oggi

7.1 Revisione e Ispezione

In questa fase siamo semplicemente partiti analizzando nuovamente la desiderata per accertarci che non ci fosse sfuggito niente durante le prime letture e analisi del contenuto. Una volta accertati che avessimo inquadrato tutti i punti siamo passati a controllare il documento confrontando i diversi diagrammi UML prodotti con il codice sviluppato per verificarne la consistenza. Si è trattato del punto più decisivo nello sviluppo del software.

Una volta conclusa l'attività siamo passati a ricontrollare il codice per controllare che non fossero presenti errori a livello di leggibilità e struttura del software, dopo aver eseguito questo passaggio si può considerare il software quasi completo, i restanti test che abbiamo fatto hanno solamento portato a correzzioni o piccole rivisatazioni

7.2 Unit Test

Per la parte di unit test, tramite gli strumenti descritti nel Capitolo 2, abbiamo testato il corretto funzionamento delle implementazioni del Software sviluppato. Partendo dal file main.cpp riportato qui sotto abbiamo provato ad effettuare delle prove per la creazione, rimozione e modifica di tutti gli elementi necessari sulla piattaforma.

```
1  #include "utils.h"
2  #include "server.h"
3  #include "cauto/macchine_management/optional.h"
4  #include "cauto/user_management/user_management.h"
5  #include "cauto/macchine_management/macchine_management.h"
6  #include "cauto/sede.h"
7
8  #include <functional>
9
10 int main(int argc, char *argv[])
11 {
12     rest_server::server server(Pistache::Address(Pistache::Ipv4::any(),
       Pistache::Port(1984)));
13     server.init(2);
14     server.start();
15     //* test signup and login
       ------------------------------------------------------------------
16     // cauto::user_management userManager;
17     // bool res = userManager.signup("alice_visit", "password123");
18     // std::cout << res << std::endl;
19     // res = userManager.login("alice_visit", "password123");
20     // std::cout << res << std::endl;
21     // res = userManager.login("alice_", "password123");
22     // std::cout << res << std::endl;
23
24     //* test creazione rimozione auto
       ------------------------------------------------------------------
25     // cauto::macchine_management database;
26     // database.get_all();
27
28     //* aggiunta
29     // cauto::macchina nuovoModello;
30     // nuovoModello.nome_univoco = "Toyota Yaris 2024";
31     // nuovoModello.descrizione = "City car compatta e moderna.";
32     // nuovoModello.prezzo_base = "20000";
33     // nuovoModello.dimensioni = {"1500 mm", "3940 mm", "1040 kg", "286 L
       "};
34     // nuovoModello.motore = {"1.5L 3 cilindri", "Ibrida"};
35     // nuovoModello.immagini = {"yaris_frontale.jpg", "yaris_laterale.jpg",
        "yaris_posteriore.jpg", {"yaris_bianco.jpg"}};
36     // nuovoModello.add_optional(cauto::optional("colore", 200.0, {"Bianco
       ", "Nero"}));
37     // nuovoModello.add_optional(cauto::optional("ruota di scorta", 300.0,
       {}));
38
39     // database.marche_auto["Toyota"].push_back(nuovoModello);
```

```
40      // database.save();
41
42      //* rimozione
43      // database.remove("Toyota", "Toyota Yaris 2024");
44      // database.save();
45
46      //* get sedi ----------------------------------------------------
47      // cauto::sedi_management db;
48      // db.load_all();
49
50      // for (const auto& c : db.get_all()) {
51      //     std::cout << "ID: " << c.id << "\nNome: " << c.nome << "\
        nIndirizzo: " << c.indirizzo << "\n\n";
52      // }
53
54  }
```

I test effettuati hanno utilizzato delle chiamate API che a loro volta erano state precedentemente testate tramite PostMan, i test sono stati eseguiti per ogni singola chiamata API creata. Di seguito riportiamo un esempio per quanto riguarda la creazione del preventivo, è comunque possibile vedere tutte le chiamate dall'apposita applicazione.

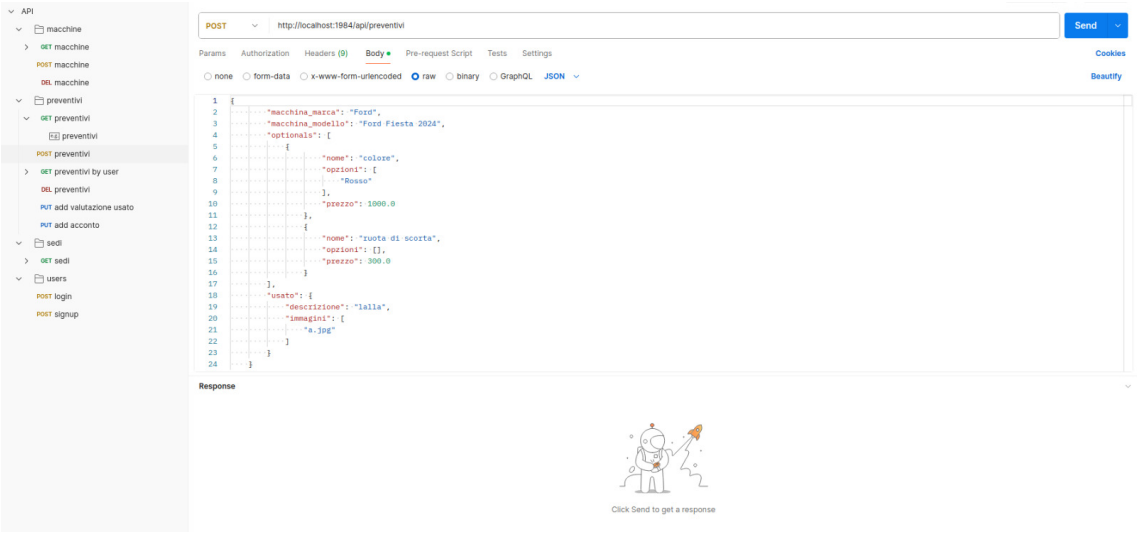


Figura 7.1: Postman POST

Oltre ai test sopra riportati abbiamo applicato dei controlli anche sulla Vista utilizzando la suite di Angular (Karma e Jasmine già analizzati nell'apposita sezione) che si occupa di lanciare i test direttamente sul browser, verificando l'integrità dei singoli componenti e il collegamento tra i diversi modelli. Di seguito viene riportato un esempio di quello che viene restituito dall'interfaccia di test, nello specifico riguardante il servizio dedito ad importare i vari moduli per le chiamate http ed effettuare le chiamate alle api.

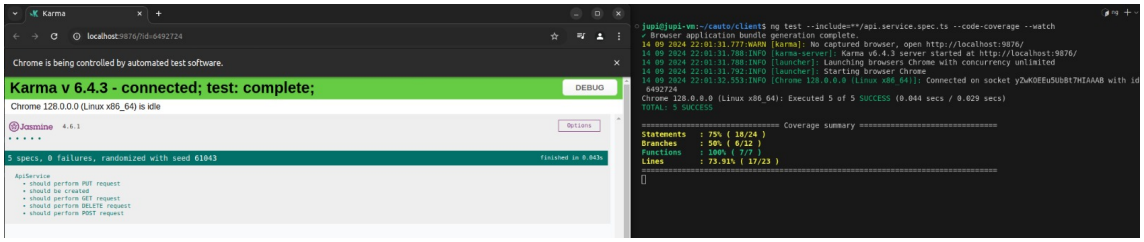


Figura 7.2: Test con Karma

7.3 Test degli Sviluppatori

In questa fase abbiamo testato personalmente il sistema per controllare che le funzioni previste dal software funzionassero correttamente. Nello specifico i test che si sono svolti sono i seguenti:

- **Autenticazione:** Per questo test abbiamo provato a simulare la creazione di nuove utenze relative al personale ossia impiegato e segreteria. Con queste utenze si è provato ad effettuare l'accesso per controllare se funzionasse correttamente e restituisse la visualizzazione dedicata alle mansioni che devono svolgere. Per l'utenza dei clienti si è provato a creare l'utenza attraverso la pagina dedicata e abbiamo verificato che non ci fossero problemi

- **Catalogo:** Per questo test abbiamo provato ad accedere alla pagina dedicata al catalogo e abbiamo verificato che restituisse correttamente tutte le auto a catalogo, abbiamo inoltre provato a configurare alcune delle auto presenti per verificare che il prezzo finale venisse mostrato correttamente

- **Preventivo:** La prima cosa che abbiamo controllato in questa fase e che la pagina dedicata per la creazione del preventivo fosse accessibile solo da autenticato quindi abbiamo provato a controllare che non ci fosse alcun modo per accederci senza aver effettuato il login. Una volta fatto questo abbiamo provato a creare un preventivo e verificare che avvenisse correttamente e che fosse correttamente mostrato nella pagina dedicata della lista dei preventivi per tutte le utenze interessate

- **Auto usata:** Da un preventivo precedentemente creato abbiamo provato ad aggiungere la richiesta per una valutazione dell'usato allegando le immagini necessarie e abbiamo verificato che quest'ultime venissero salvate correttamente e che la richiesta fosse visibile

- **Impiegato:** Provando ad accedere con un utenza da impiegato abbiamo verificato che si vedessero correttamente tutti i preventivi sia quelli che richiedevano la valutazione dell'usato sia quelli già avvisati che erano in attesa della consegna. Abbiamo controllato che riuscisse a vedere correttamente le immagini per la valutazione e che riuscisse a completare la valutazione. Abbiamo inoltre porvato ad avvisare il cliente quando l'auto fosse pronta per la consegna

- **Segreteria:** Provando ad accedere con un utenza da segreteria abbiamo verificato che vedesse correttamente tutte le pagina ossia la lista dei preventivi e del catalogo. Nella pagina dei preventivi abbiamo controllato che ci fossero tutti e che avesse la possibilità di filtrarli mentre nella pagina del catalogo abbiamo controllato che potesse modificare, aggiungere e rimuovere sia auto e preventivi

- **Acquisto:** Dopo tutte le premesse, con un preventivo finalizzato, abbiamo testato che il pagamento dell'acconto e la proposta della data di consegna funzionasse correttamente. Come ultimo test abbiamo inoltre verificato che passati i 20 giorni senza che avvenisse il pagamento il preventivo venga rimosso

7.4 Test degli Utenti

Come test finale abbiamo fatto provare l'applicazione a diversi utenti con più o meno dimestichezza con la tecnologia. Per lo svolgimento del test è stata solamente fornita la finalità del software. Nella prima fase abbiamo dato agli utenti piena libertà di navigazione mentre in un secondo momento abbiamo richiesto alcune particolari simulazioni indicando cosa volevamo ottenere per testare in particolare le utenze degli impiegati e delle segreteria. Durante ogni fase del test abbiamo risposto ad eventuali domande e raccolti tutti i feedback ricevuti, in questo modo abbiamo avuto la possibilità di correggere alcuni piccoli problemi e migliorato il software del punto di vista dell'usabilità.

Capitolo 8

Pattern Architetturali

Un pattern architetturale è una soluzione collaudata e riutilizzabile per affrontare problemi comuni nella progettazione dell'architettura del software. Esso definisce una struttura organizzativa generale per un sistema fornendo linee guida su come suddividere il software in componenti, moduli e livelli e su come questi interagiscono tra loro. Questa tipologia di pattern aiutano a garantire che il software sia scalabile, manutenibile e flessibile e la sua adozione consente ai progettisti di beneficiare delle esperienze accumulate nella risoluzione di problemi simili, migliorando l'efficienza e la qualità del progetto.

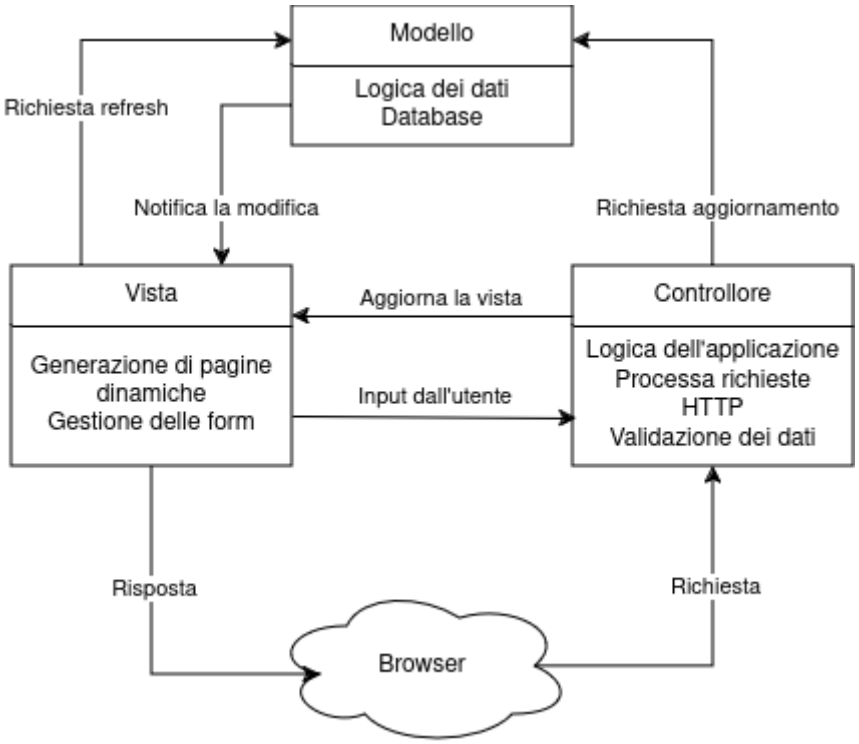


Figura 8.1: Architettura MVC

Per il nostro progetto abbiamo scelto di utilizzare il **pattern MVC**. Come dice il nome, questo modello è caratterizzato da tre elementi:

- **Modello:** Rappresenta la logica di gestione dei dati. Esso gestisce i dati dell'applicazione, risponde alle richieste della View per l'accesso ai dati e risponde alle istruzioni del Controller per modificare i dati
- **Vista:** Si occupa della presentazione dei dati all'utente. Essa visualizza i dati che riceve dal Modello e riflette eventuali cambiamenti nei dati. E' responsabile dell'interfaccia utente e di come i dati vengono presentati all'utente finale
- **Controllore:** Funziona come intermediario tra il Modello e la Vista. Esso riceve l'input dell'utente attraverso la Vista, elabora tale input (eventualmente interagendo con il Modello) e aggiorna la Vista di conseguenza.

Le azioni dell'utente saranno catturate dai listener del controllore, quest'ultimi sono pensati per reagire di conseguenza andando a modificare le informazioni contenute nel Modello e di conseguenza aggiornetà la vista che rappresenta il Modello
É stato scelto questo modello in quanto si adatta bene a quello che abbiamo progettato, i componenti dei modelli e del controllore possono essere riutilizzati con diverse viste il che aumenta l'efficenza nello sviluppo di nuove funzionalità o versioni dell'applicazione. Il modello MVC è una scelta popolare per lo sviluppo di applicazioni web e software, soprattutto in contesti in cui la manutenibilità, la modularità e la collaborazione sono essenziali.

Capitolo 9

Pattern adottati

I design pattern sono per l'appunto pattern che utilizzano metodi e classi di un linguaggio OO. Durante lo progettazione, la maggior parte dello sviluppo è avanzato attraverso la conoscenza dei programmatori dovuta all'esperienza lavorativa, non sono stati quindi stati seguiti consciamente dei particolari pattern. Alla fine dei conti l'unico con il quale ci siamo allineati è il **Data Access Object Pattern** (DAO).
Questo pattern nello specifico viene utilizzato per separare la logica di accesso ai dati dalla logica delle operazioni, in sostanza agisce come interfaccia tra il programma e la fonte di dati, come per esempio un database, consentendo di eseguire operazioni di CRUD (Create, Read, Update, Delete) senza esporre direttamente i dettagli di implementazione dei meccanismi di persistenza. Il DAO Pattern è strutturato nel seguente modo:

- **Interface**(astrazione del DAO): definisce i metodi per l'accesso ai dati come getUsr() e createUser()
- **Concrete DAO**: Implementa l'interfaccia e cotiene la logica specifica per interagire con il database o altre fonti di dati
- **Entity Model**: la classe che rappresenta l'entità che mappa i dati(ad esempio User) contenente gli attributi corrispondenti alle colonne del database
- **Service Layer**: Il livello che utilizza il DAO per accedere e gestire la logica del business

Questo particolare pattern è stato leggermente riaddatato secondo le nostre esigenze, infatti all'interno della nostra struttura non è presente l'interface in quanto abbiamo deciso di non creare alcuna classe astratta. Abbiamo deciso di utilizzarlo in linea generale nella costruzione delle API e comprende ottenimento, creazione, aggiornamento e rimozione di tutte le entità in gioco in questo progetto

Capitolo 10

Note sul processo di sviluppo

Il progetto è stato sviluppato principalmente seguendo il modello di tipo a Cascata implementando alcuni aspetti del tipo Agile. Il modello a cascata è caratterizzato da fasi separate e distinte di specifiche e sviluppo, in linea di principio una fase deve essere completata prima di passare alla fase successiva. Il modello a cascata rende difficile "accogliere" i cambiamenti a processo avviato quindi non è l'ideale per rispondere ad eventuali modifiche richieste dal cliente. Al contrario però si presta molto bene nel caso in cui i requisiti siano ben chiari fin dall'inizio come per il nostro caso dove è stato proposto lo sviluppo di una desiderata che rimasta fissa. Il fatto dell'applicazione del modello a cascata è ben visibile anche dall'elaborato appena sviluppato. Ogni aspetto dello sviluppo è avvenuto passo per passo, abbiamo definito con chiarezza:

- I requisiti richiesti
- L'ideazione del design del software
- L'implementazione e gli Unit Test
- Test del sistema

Procedendo in questo modo abbiamo creato una solida base su cui lavorare nello step successivo riducendo così il rischio di "perdere tempo" nel dover tornare indietro nell'accorgerci che qualcosa era stato definito in maniera errata.
Per permettere comunque un lavoro più fluido si è deciso comunque di uscire un pò dalla rigida struttura del modello cascata per agevolare un pò le tempistiche visti gli impegni del gruppo. Periodicamente ci fornivamo a vicenda gli aggiornamenti riguardo ciò che avevamo fatto, in questo modo se uno doveva lavorare allo step successivo poteva cominciare a portarsi avanti con il lavoro defininendo almeno gli aspetti base e una volta terminato lo step antecedente poteva completarlo aggiungendo ciò che restava in sospeso.
Per quanto riguardo l'intera implementazione ci siamo suddivisi i ruoli in base alla confidenza dei membri del gruppo sulle parti richieste da sviluppare, ciò ha permesso uno sviluppo più rapido in quanto abbiamo ridotto il rischio di problemi dovuti allo sviluppo simultaneo del codice. L'elemento chiave per la riuscita nello sviluppo del software è stata la pianificazione dettagliata creata all'inizio che abbiamo seguito dall'inizio alla fine. Nell'ottica del team work eravamo comunque pronti a fornire supporto in caso di necessita sulle attività degli altri membri del gruppo.

10.1 Scelte Progettuali

Come già possibile intuire da quanto presentato il software è stato sviluppato con linguaggi diversi da quelli presentati al corso inoltre la struttura del codice non è composta da classi astratte,

protette o private come quanto studiato dal linguaggio Java quest'anno. Tutto è stato creato come pubblico.
Inoltre nonostante i vari test che abbiamo effettuato e le diverse release rilasciate siamo consapevoli che il software in fin dei conti è ancora in fase di sviluppo. Abbiamo implementato tutte le richieste derivanti dalla desiderata ma ci sono alcuni aspetti non presenti ad oggi alla presentazione per poterla definire completa. Ciò che manca viene inteso come sviluppo futuro, e nello specifico si tratterebbe di:

- **Integration test:** Questo specifico test si riferisce al processo di verifica delle interfaccie tra due moduli software per verificare come i dati vengono trasferiti tra di loro, nel nostro caso andrebbe applicato per testare lo scambio tra la vista ed il controllore
- **Sicurezza e Crittografia:** In questo momento sul software non è stata implementata alcuna misura di sicurezza, essendo un qualcosa che deve essere pubblicata sul Web, al giorno d'oggi è di vitale importanza che vengano applicati degli accorgimenti per quanto riguarda la sicurezza. Prendiamo per esempio il login, in questo momento questi dati vengono passati da un modulo all'altro senza che vengano criptati, in questo stato chiunque sappia interciettare la chiamata può vedere tranquillamente i dati sensibili degli utenti, lo stesso vale per i possibili dati inseriti durante il pagamento
- **Sistemi di pagamenti:** Al momento della presentazione viene simulato l'inserimento del pagamento, però, per rendere il software veramente completo andrebbero implementati dei veri metodi di pagamento, almeno quelli classici come la carta di credito o PayPal.
- **Email per la conferma:** Si tratta di una possibile implementazione che ci è venuta in mente durante le fasi di revisione. Nella desiderata è stato fatto presente che l'impiegato si occupa anche di avvisare gli utenti nel momento in cui l'auto da lui ordinata è pronta per il ritiro, non è stato fatto però presente come avviene ciò. Ad oggi è stato deciso che si tratta di un qualcosa che avviene al di fuori del funzionamento del software per esempio tramite chiamata telefonica. Abbiamo ritenuto comunque interessante considerare il fatto di poter implementare una funzione che si occupa in automatico di avvisare un email all'utente utilizzando quella che ci ha fornito in fase di registrazione