**INTERFACCIA WEB**

*In questo capitolo vengono illustrate le scelte effettuate per l’implementazione dell’interfaccia web, a livello di decisioni progettuali e di tecnologie utilizzate.*

Per la realizzazione di tale porzione dell’applicazione è stato usato Spring MVC, framework utilizzato per realizzare web-app basate sul modello MVC che sfrutta i punti di forza offerti dal framework Spring come l’**inversion of control** (tramite dependency injection) e la **aspect oriented programming**. Spring MVC si occupa di mappare metodi e classi Java con determinati url, di gestire differenti tipologie di “viste” restituite al client, di realizzare applicazioni internazionalizzate e di gestire i cosiddetti temi per personalizzare al massimo l’esperienza utente.

Il pattern MVC

Per comprendere al meglio il framework è necessario introdurre il pattern architetturale che implementa ovvero il modello MVC. MVC rappresenta un acronimo per Model View Controller ovvero le tre componenti principali di un’applicazione web.

* Il **Model**  si occupa di accedere ai dati necessari alla logica di business implementata nell’applicazione ed è indipendente dall’interfaccia utente;
* le **View** si occupano di creare l’interfaccia utilizzabile dall’utente e che espone i dati da esso richiesti;
* i **Controller** si occupano di implementare la vera logica di business dell’applicazione integrando le due componenti precedenti, ricevendo gli input dell’utente, gestendo i modelli per la ricerca dei dati e la creazione di viste da restituire all’utente.

MVC in Spring

Spring MVC implementa perfettamente questo approccio mantenendo sia i concetti che la nomenclatura del pattern. All’interno di una applicazione Spring MVC avremo quindi:

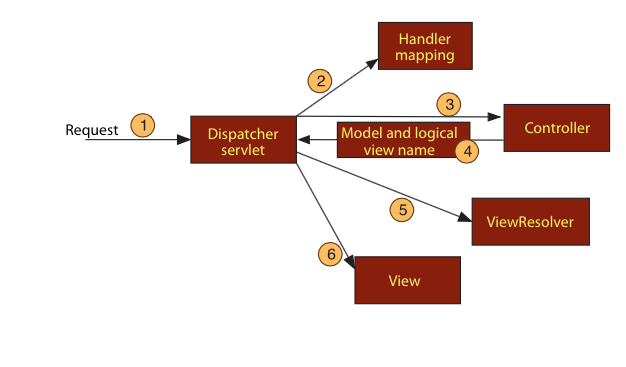
* il Model, rappresentato dalle classi che a loro volta rappresentano gli oggetti gestiti e le classi di accesso al database;
* le View, rappresentate dai vari file JSP (che vengono compilati in HTML) e da eventuali classi per l’esportazione in formati diversi da HTML (PDF, XLS, CSV…);
* i Controller, rappresentati da classi (chiamate appositamente Controller) che rimangono “in ascolto” su un determinato URL e, grazie al Model e alle View, si occupano di gestire la richiesta dell’utente.

Secondo la documentazione ufficiale Spring MVC presenta inoltre molti altri vantaggi oltre alla netta separazione tra le funzionalità:

* è adattabile, flessibile e non intrusivo grazie alla presenza di comode e chiare Java Annotations;
* permette di scrivere codice riusabile;
* possibilità di essere esteso tramite adattatori e validatori scritti ad hoc per le specifiche esigenze;
* url dinamici, SEO-friendly e personalizzabili;
* gestione integrata dell’internazionalizzazione e dei temi;
* libreria JSP sviluppata ad hoc per facilitare alcune operazioni ripetitive;
* nuovi scope per i bean (request e session) che permettono di adattare i container base di Spring anche al mondo web.

**Gestione di una richiesta in Spring MVC**

Nell’utilizzo di Spring MVC, una delle prime cose che vanno studiate attentamente è il processo di una gestione delle richieste. Una generica richiesta nasce quando un utente, attraverso il suo browser, clicca un link oppure completa ed invia un form. Prima che al browser dell’utente giunga una risposta, la richiesta attraversa diversi step e viene processata più volte. La seguente figura riassume tutte le trasformazioni che una richiesta deve subire.



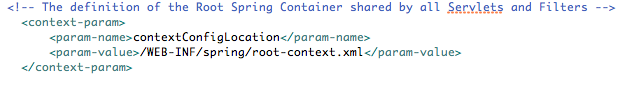
Come si evince dal punto 1, la richiesta inizialmente viene presa in carico dalla **Dispatcher servlet**. Come molti framework Java, anche Spring MVC gestisce le richieste attraverso un’unica servlet che ricopre il ruolo di **front controller**. Un front controller è un comune pattern usato nelle applicazioni web dove ad una singola servlet è affidato il compito di smistare la richiesta al componente che eseguirà l’elaborazione vera e propria. Il compito della Dispatcher servlet è quello di inviare la richiesta ad un controller. Un **controller** è un componente in grado di processare la richiesta. Una tipica applicazione è composta da diversi controller e ld Dispatcher servlet necessita di maggiori informazioni per decretare quale controller deve prendere in carico la richiesta. Per tale motivo la Dispatcher servlet consulta uno o più **Handler mapping** per capire quale sarà il prossimo componente che dovrà analizzare la richiesta. L’Handler mapping presta particolare attenzione all’URL della richiesta per aiutare la Dispatcher servlet nel suo processo decisionale. A questo punto, in base alle informazioni in suo possesso, la Dispatcher servlet può inviare la richiesta al **Controller** opportuno. Il Controller ricaverà dalla richiesta il payload (le informazioni realmente inviate dall’utente) che lo userà per effettuare il compito di elaborazione per il quale è stato implementato. Solitamente un Controller ben progettato compie piccole elaborazioni. Elaborazioni più complesse sono affidate alla logica di business. Il risultato dell’elaborazione del controller è, genericamente, un insieme di informazioni che devono essere trasmesse all’utente che ha inviato la richiesta. Inviare queste informazioni direttamente al client non è il modo più consono di operare. Tali dati devono essere formattati in un formato user-friendly, tipicamente in HTML. Per tale motivo è necessario trasferire tali informazioni ad una **view**. Un altro compito affidato al controller è quello di impacchettare il modello con il nome della vista che dovrà formare l’output. Pertanto, come mostrato in figura dalla freccia n.4, il Controller invia alla Dispatcher Servlet il modello ed il nome logico della vista. Il Controller, in questo modo, non è accoppiato nessuna vista. Il nome logico, infatti, non identifica direttamente nessuna JSP. Alla Dispatcher Servlet viene comunicato un nome che servirà, in seguito, ad identificare quale vistà sarà designata a rappresentare le informazioni. Pertanto la Dispatcher Servlet utilizzerà il **View Resolver** per identificare una specifica implementazione di una vista a partire dal nome logico. A questo punto la Dispatcher Servlet conosce quale vista rappresenterà i dati e la gestione della richiesta iniziale dell’utente può essere considerata completata.

**Settare Spring MVC**

Come evidenziato poc’anzi, la DispatcherServlet rappresenta un elemento importante all’interno di Spring MVC. Essendo una servlet va configurata all’interno del file web.xml che compare all’interno di ogni applicazione web realizzata in Java.



Nel frammento del file web.xml riportato in foto è stata specificata la classe DispatcherServlet che avrà il ruolo di front controller dell’applicazione. Particolare importanza ha anche il tag servlet-name al cui interno è stato specificato il valore “car2go”. Per default, quando la DispatcherServlet è caricata, questa carica anche l’application context di Spring a partire dal file XML il cui nome è basato sul contenuto del tag servlet-name. In questo caso, siccome la servlet è chiamata car2go, la DispatcherServlet proverà a caricare l’application-context da un file chiamato car2go-servlet.xml che deve essere presente all’interno della cartella WEB-INF.



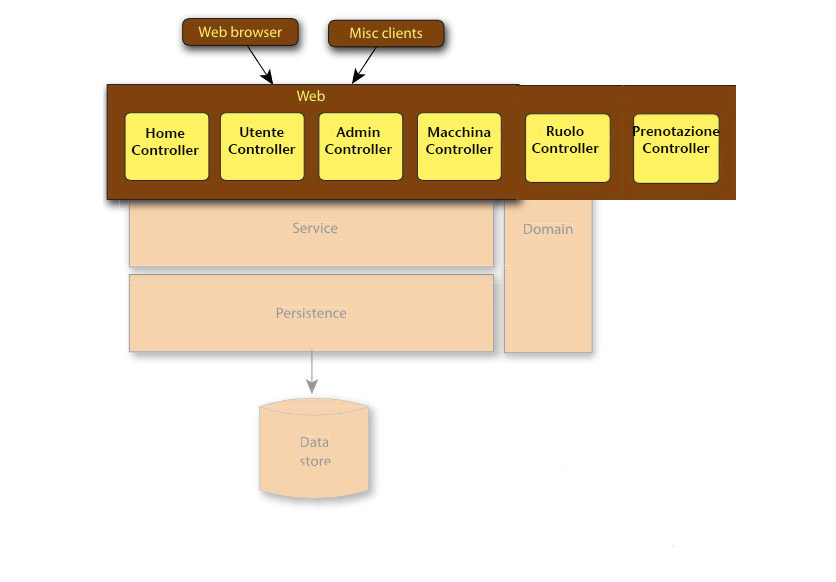
All’interno del tag context-param è possibile definire altri file di configurazione di Spring. In questo modo è possibile applicare i principi della modularità anche ai file di configurazione di Spring. Si potrebbe infatti usare un file di configurazione per ogni scopo: data source, persistenza e sicurezza.

**Controller**

Settato l’ambiente per utilizzare Spring MVC, è necessario stabilire quanti e quali controller realizzare. Il principio di base seguito è stato quello di implementare un singolo controller per uno tipo elementare di risorsa che l’applicazione deve gestire. In Car2Go sono stati quindi realizzati i seguenti controller:

* **UtenteController**
* **MacchinaController**
* **RuoloController**
* **PrenotazioneController**

Ai seguenti controller, viste le numerose operazioni che deve compiere un utente amministratore, è stato creato un apposito controller chiamato **AdminController** proprio per gestire in modo più efficiente le funzionalità riservate all’Admin.



Ai controller appena citati va aggiunto l’**HomeController** il cui compito è quello di svolgere le operazioni necessarie a mostrare le informazioni sulla prima pagina dell’applicazione. E’ stato realizzato un HomeController perché i compiti che deve realizzare non sono associati direttamente a nessun concetto di base dell’applicazione.

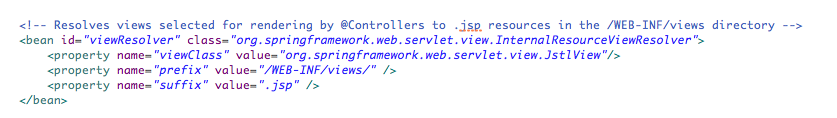
L’HomeController è il primo controller che deve essere scritto.

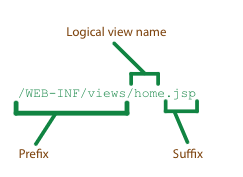


Come si evince dal codice mostrato, è possibile e consigliato usare le annotazioni per facilitare il lavoro di implementazione. Mediante l’annotazione @Controller si indica che la classe HomeController è un controller. L’annotazione @Controller è una specializzazione dell’annotazione @Component ed in virtù di questa relazione Spring scopre e registra HomeController direttamente come un bean senza nessuna configurazione esplicita da parte del programmatore nel file dell’application-context. Per raggiungere gli scopi per i quali l’homeController è stato creato, quest’ultimo necessita di recuperare l’elenco delle macchine non prenotate. Tale lista sarà poi passata alla vista che si occuperà di comporre il codice HTML da mandare in output. Per recuperare la lista di auto non prenotate viene fatto uso di MacchinaService. Tale interfaccia specifica delle operazioni di business la cui reale implementazione dipende dalla particolare tecnologia usata. Nel caso di car2go-simple-web esisterà un’apposita classe che realizzerà l’implementazione di MacchinaService. Nella realizzazione di Car2Go mediante RMI, MacchinaService è l’interfaccia remota del servizio. La reale implementazione sarà affidata a degli elementi Servant lato Server. Non implementando la logica di business direttamente nel Controller ed usando il meccanismo delle interfacce per la realizzazione di Porte ed Adattatori è stato raggiunto lo scopo di ottenere l’indipendenza dalla tecnologia con la quale sono implementate le operazioni di business. Il compito di un controller è pertanto quello di catturare le richieste ed i relativi dati provenienti dall’utente ed, in base a questi, attivare l’opportunità funzionalità di business. I dati ricavati dai metodi di business saranno poi passati alla vista che si occuperà di mostrarli all’utente finale. Attraverso il costruttore di HomeController e mediante le annotazioni @Autowired ed @Inject viene iniettato in macchinaService un riferimento ad un oggetto che implementa l’interfaccia MacchinaService. showHomePage è il metodo che si occupa di recuperare l’elenco delle macchine e passarlo ad una vista. Tale metodo è annotato con @RequestMapping({“/”,”/home”}) per indicare che ogni richiesta il cui path è formato da / oppure /home sarà gestita dalla funzione showHomePage. Tale metodo, come parametro di input prevede una Map di Stringa-Oggetto. Tale Map rappresenta il modello. Dopo aver recuperato la lista delle macchina non prenotate tale elenco viene aggiungo al modello in modo da essere mostrato dalla vista. Per concludere l’analisi di tale controller è necessario discutere del valore di ritorno. Come si nota dal codice precedente il metodo ritornaun oggetto di tipo String. Tale stringa è il nome logico della vista che dovrà mostrare i dati. Un controller, come precedentemente spiegato, non deve prendere parte direttamente nel processo di individuazione delle reale vista che dovrà formattare in modo opportuno i dati ma dovrà solo identificare un nome logico di una vista. Una volta che il controller ha terminato il suo lavoro la DispatcherServlet, mediante l’aiuto del ViewResolver, userà tale nome logico per trovare la vera vista che si occuperà di presentare i dati all’utente.

Il ViewResolver va configurato all’interno dell’application-context principale ossia, nel caso di Car2Go, car2go-servlet.xml.

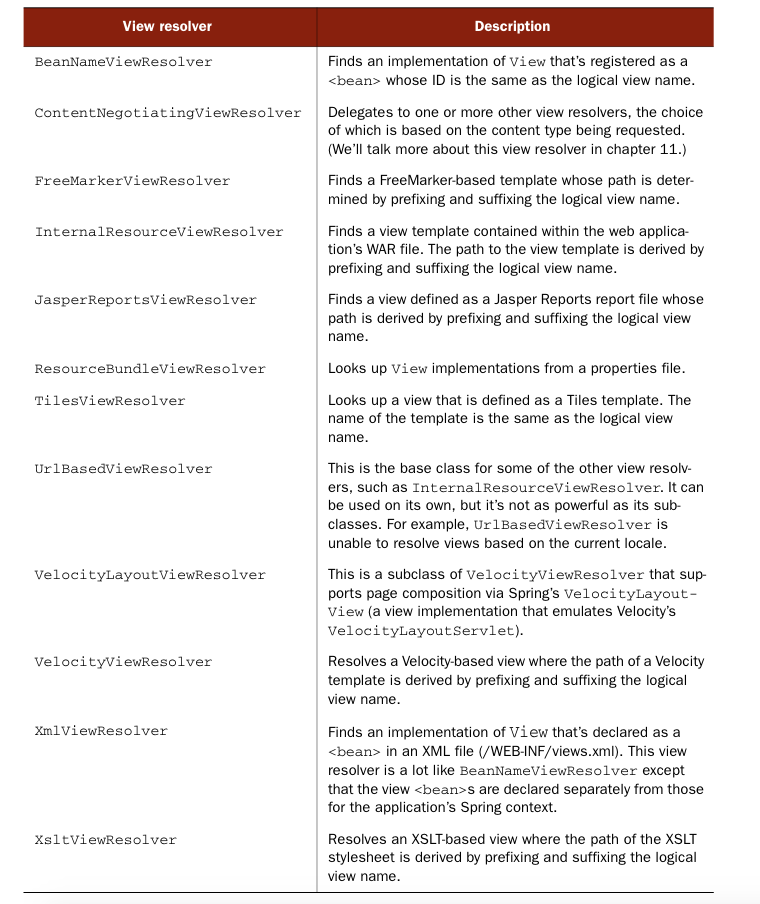
Di seguito si riporta la configurazione usata.



Il ViewResolver usato è InternalResourceViewResolver. Il comportamento di tale resolver è molto semplice e può essere esemplificato dalla figura seguente. 

InternalResourceViewResolver aggiunge al nome logico il suffisso .jsp mentre il prefisso è /WEB-INF/views/.

Spring mette a disposizione diversi resolver. E’ possibile scegliere il resolver più appropiato in base alle esigenze dell’applicazione che si intende realizzare.

Di seguito una tabella che mostra nome del View Resolver e una breve descrizione.

**Car2Go-Web**

Attraverso l’interfaccia web, un **utente non registrato** può accedere a diverse funzionalità, tra cui:

* Visualizzare l’elenco delle auto disponibili;
* Visualizzare per ogni auto il dettaglio su posizione, pulizia interna, stato esterno e benzina;
* Registrazione al sistema;
* Autenticazione;

Un **utente che si è registrato** al sistema, dopo essersi autenticato può:

* Visualizzare il riepilogo dei proprio noleggi;
* Visualizzare il riepilogo dei costi;
* Prenotare un’auto, previa autenticazione;
* Annullare la richiesta di prenotazione di un’auto;
* Effettuare la simulazione di guid;
* Effettuare il logout.

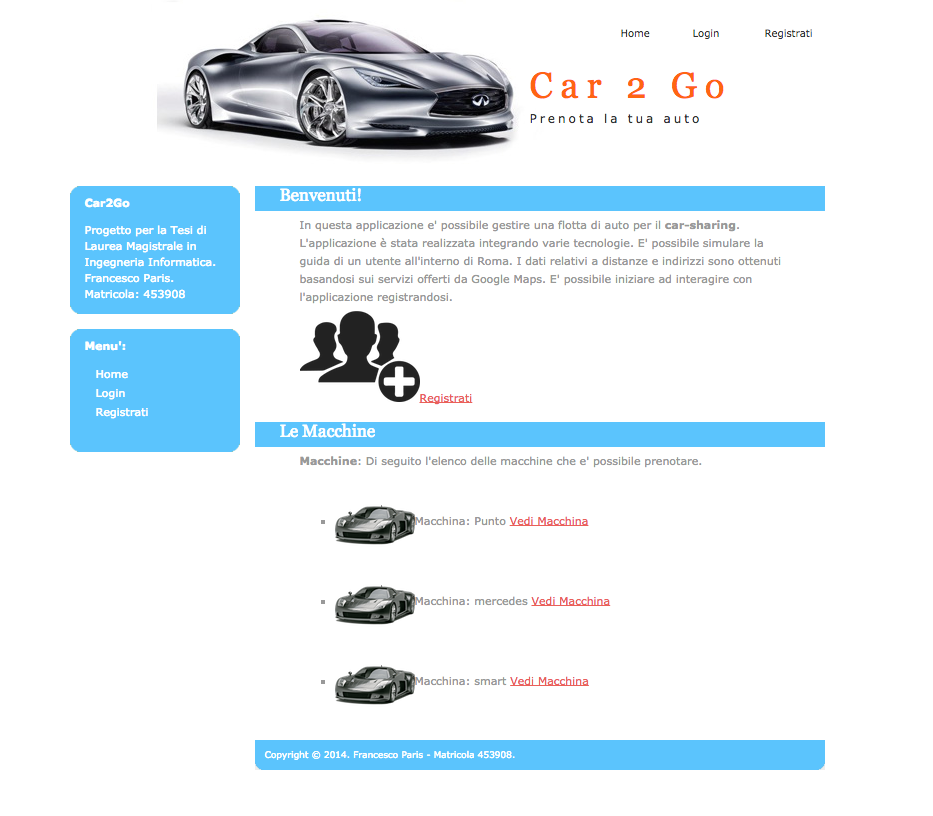
Un utente denotato come **Amministratore**, una volta autenticato, può:

* Abilitare al ruolo di Amministratore utenti semplici;
* Creare nuove macchine;
* Effettuare il rifornimento alle auto;
* Pulire internamente le auto;
* Riparare le auto;
* Effettuare il logout;

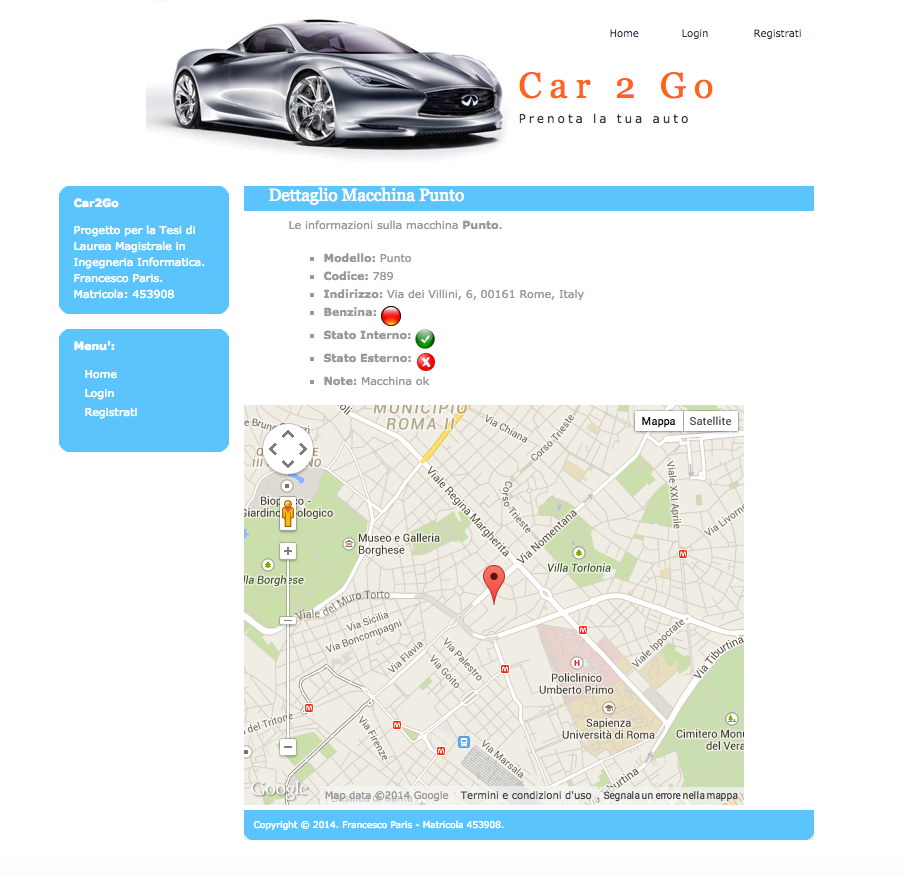
Queste operazioni sono offerte all’interfaccia web da tutti gli elementi che implementano le interfacce presenti in car2go-service-common. Le interfacce presenti nel package it.car2go.service, ossia MacchinaService, PrenotazioneService, RuoloService, UtenteService, rappresentano una API per la versione web dell’applicazione verso la logica di business. Gli oggetti che implementano tali interfacce, sebbene realizzati con tecnologie differenti, si devono occupare di accedere ai dati per realizzare le operazioni di business.

Di seguito alcune immagini che raffigurano l’applicazione in esecuzione.

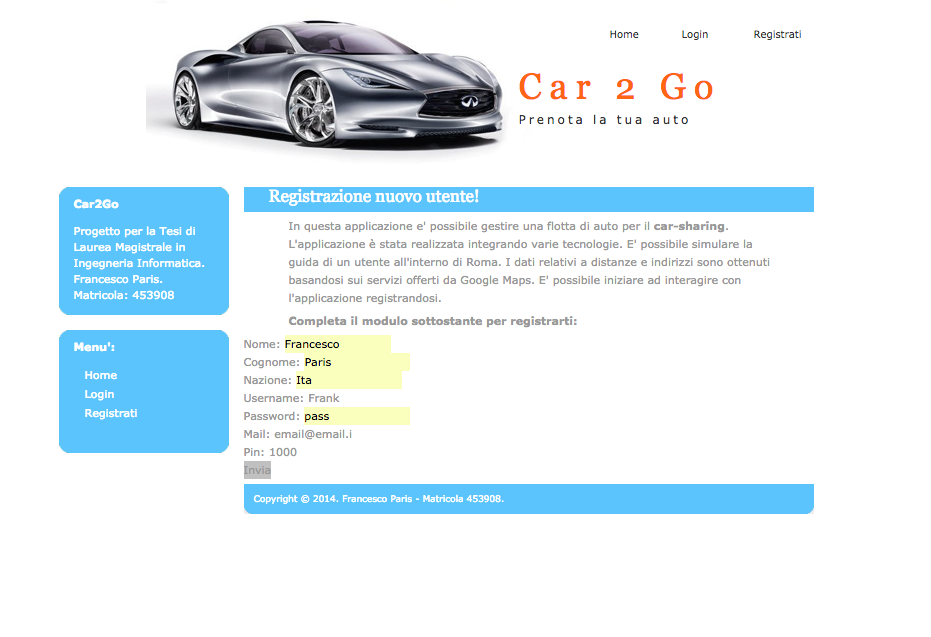
La seguente immagine mostra la home page dove è possibile, a partire dal menu, scegliere di effettuare il login oppure registrarsi. E’ possibile, attraverso l’elenco presente nella sezione Le Macchine, selezionare una vettura per vederne tutti i dettagli.



Richiedendo di vedere il dettaglio di una macchina si ottiene una schermata come la seguente:

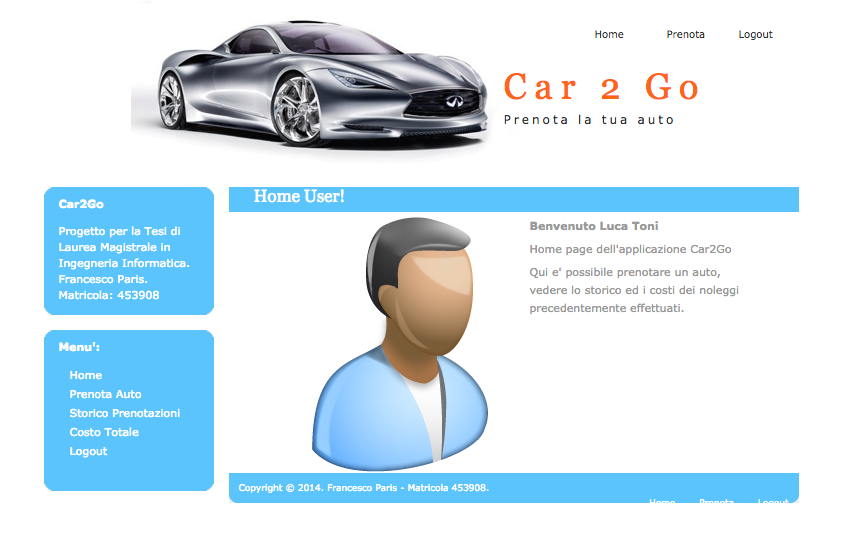


Le icone di benzina e stato interno/esterno cambiano colore a seconda dello stato della macchina. I valori di latitudine e longitudine sono usati per rappresentare (attraverso il marker rosso) la posizione della macchina sulla Mappa di Google. La schermata per permettere la registrazione o l’autenticazione di un utente è un semplice form. Di seguito si riporta la schermata per registrare un nuovo utente.

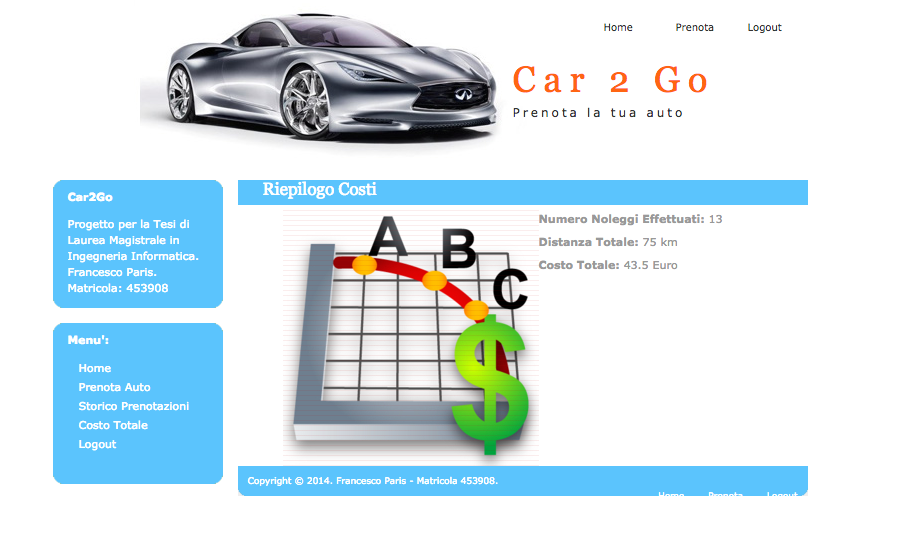


Una volta che un nuovo utente si sarà registrato, a quest’ultimo, per default, gli sarà assegnato il ruolo User. Solo l’Amministratore ha la possibilità di eleggere Admin un utente semplice.

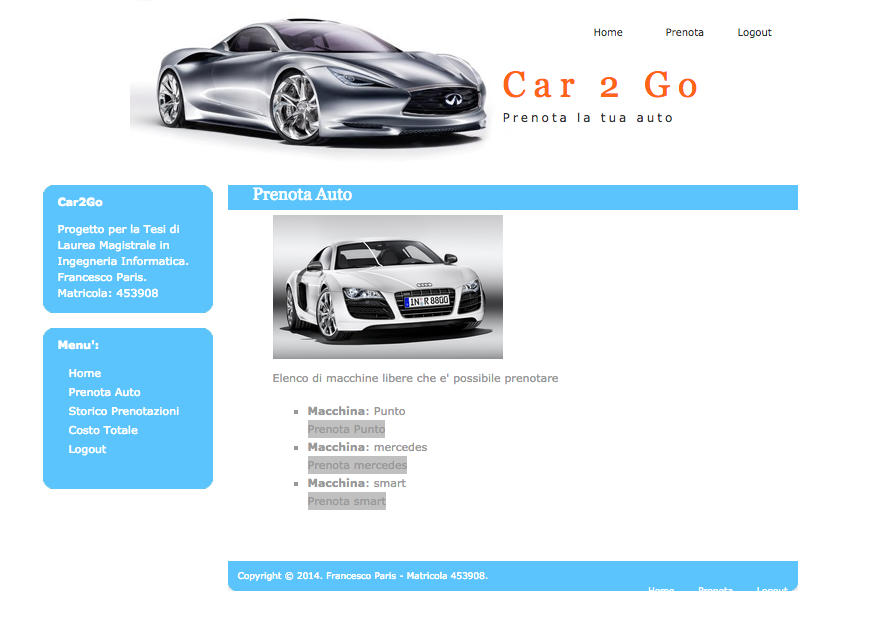
L’immagine seguente mostra la pagina che sarà mostrata quando un utente con ruolo User si autenticherà al sistema. Come si evince dal menù è possibile consultare lo storico dei noleggi effettuati e vedere i costi sinora sostenuti.



Ecco la schermata raggiungibile selezionando la voce “Costo Totale” dal menù presente a sinistra della precedente immagine.



La funzionalità più complessa è quella relativa a Prenota Auto. Per noleggiare un auto, la prima cosa che l’utente dovrà fare, sarà la scelta di una vettura tra quelle al momento libere. Tale azione si può compiere scegliendo, mediante clic sul pulsante grigio, una fra le vetture disponibili.

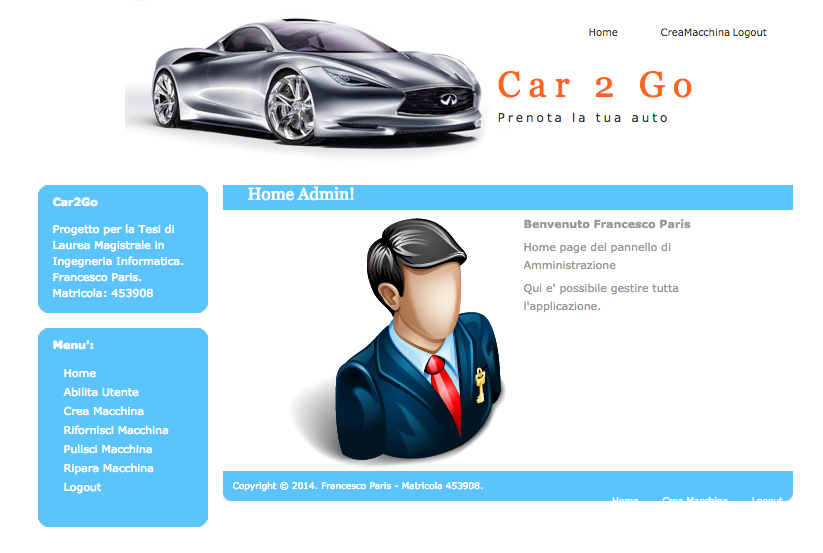


Scelta la vettura l’utente si troverà proiettato in un’altra schermata come la seguente:

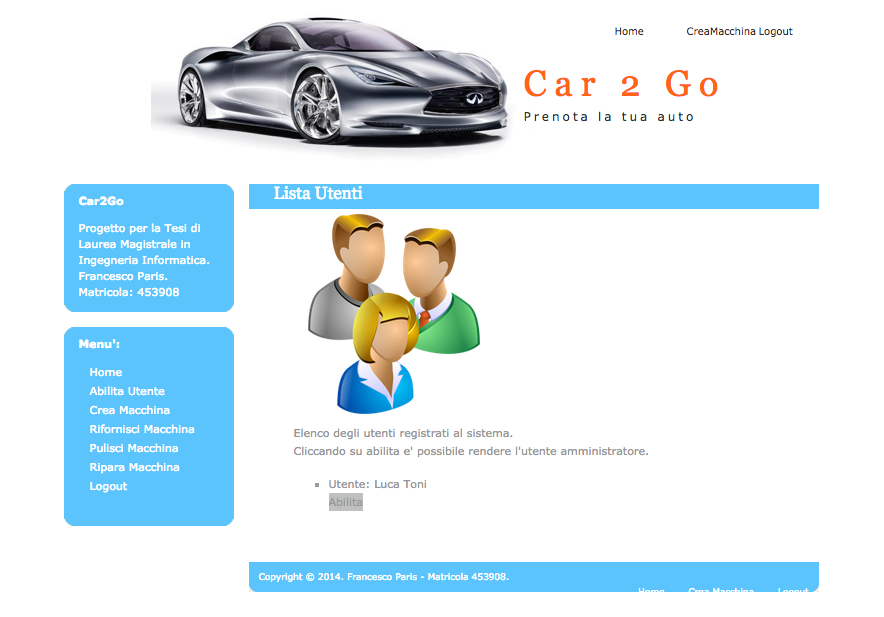


Come si nota dall’immagine, tale pagina è composta da due parti. La parte inferiore permette ad un utente di annullare la prenotazione. Cliccando sul pulsante grigio, la vettura appena prenotata ritornerà ad essere libera pronta per accettare altre prenotazioni. La parte superiore della pagina presenta alcune informazioni sull’utente che ha effettuato la prenotazione e sull’auto oggetto del noleggio. Prima di poter confermare l’intenzione di guidare il mezzo, proprio come accade all’interno di una vettura appartenente a Car2Go, l’utente è chiamato a dare un giudizio sullo stato di pulizia interno della vettura, sulle condizioni esterne ed, eventualmente, a specificare alcune note aggiuntive. Cliccando su Guida, verrà simulato un percorso all’interno del Grande Raccordo Anulare di Roma. Il sistema genererà delle nuove coordinate di latitudine e longitudine e calcolerà la distanza dalla destinazione randomica appena creata ed il punto di partenza. Come già spiegato per i dati geografici il sistema usa il servizio messo a disposizione da Google Maps. L’applicazione, in base alla distanza, stima tempi e costi del viaggio ed infine abbassa il livello di carburante. Al termine di questa elaborazione l’utente verrà riportato nella sua home page dove potrà vedere i dettagli del viaggio appena effettuato nello storico relativo ai noleggi ed ai costi.

L’altra tipologia di utente che può interagire con l’applicazione è l’Admin.

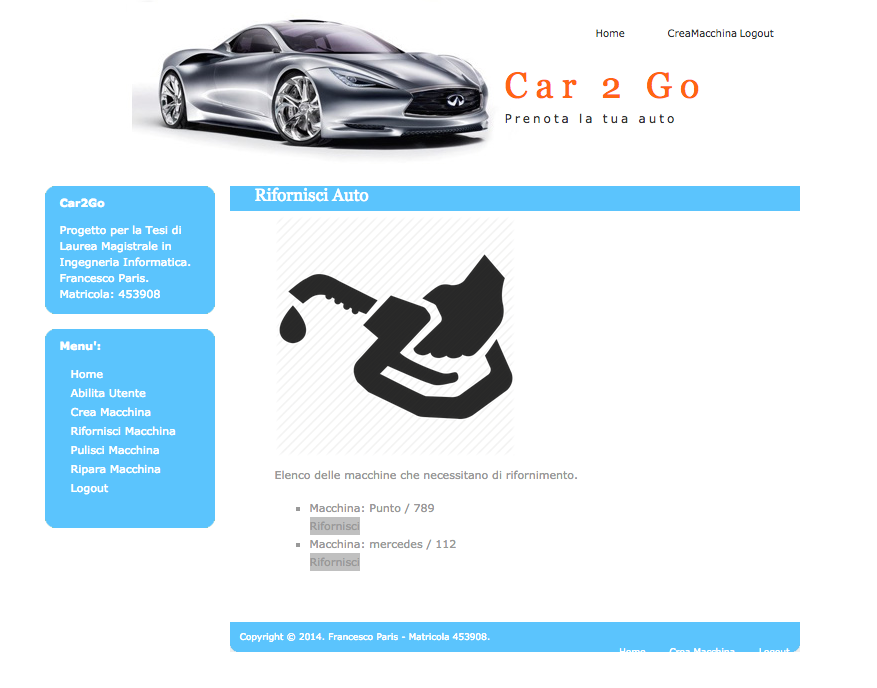


Un Admin, come mostra la schermata precedente ottenuta dopo il login, può abilitare un utente oppure creare e gestire le automobili. Nell’immagine seguente si mostra la schermata mediante la quale un amministratore può abilitare un utente.



La creazione di una nuova macchina verrà effettuata mediante un semplice form dove sarà cura dell’Amministratore inserire tipo e codice della vettura. I valori relativi alla posizione geografica saranno completati in modo randomico direttamente dall’applicazione.

Le schermate per rifornire, pulire e riparare una vettura sono fondamentalmente tutte uguali.



Tali pagine sono costituite da un elenco di macchine che necessitano di essere rifornite/pulite/riparate. L’amministratore può completare l’azione cliccando sul pulsante grigio. Terminato tale compito l’amministratore verrà riportato sulla sua home page.

Un utente che si è autenticato al sistema, di qualche tipo esso sia, può effettuare il logout. Una apposita pagina confermerà che l’azione di logout è stata completata con successo.

Eventuali errori dell’applicazione causati da un inserimento errato dei dati da parte dell’utente saranno notificati a quest’ultimo mediante una apposita “pagina d’errore”. Nel caso della registrazione di un nuovo utente, se i campi del form vengono compilati in maniera errata, verranno mostrati dei suggerimenti nella stessa pagina dove è presente il modulo.

VEDERE SE AGGIUNGERE ALTRE FOTO

POI SE METTERE LA MAPPA DEL SITO

ED EVENTUALI CONCLUSIONI