

**CarBook**

*Un applicazione di Simone Venegoni, Luca Giovanni Nastasi, Daniele Consonni, Marco Picuno*

**Introduzione**

Benvenuto in Carbook, la nuova app dove inserire il tuo annuncio per vendere o comprare un’auto!

Visualizza le offerte del rivenditore di auto vicino a casa, troverai senz’altro quello che potrebbe diventare il tuo prossimo acquisto. Che si tratti di auto usate o nuove, auto eco-sostenibili o potenti vetture sportive, grazie a Carbook, troverai velocemente l’auto dei tuoi desideri a prezzi convenienti.

Nel nostro mercato on-line di autoveicoli puoi trovare auto usate o nuove in pochi clic, proprio con le caratteristiche che cercavi e sempre con un contatto diretto con venditori o concessionari.

Nell’annuncio vi sarà presente anche una sezione “Descrizione” dove il venditore si occuperà di fornire una dettagliata presentazione, raccontando la propria vettura nei particolari; più approfondita sarà la descrizione, più possibilità avrai di fare conoscere il tuo annuncio e di vendere il veicolo.

Ti auguriamo una buona permanenza sulla piattaforma e buona compravendita!

**Scopo e motivazioni**

Carbook ha come scopo quello di aiutare gli utenti a compiere dei buoni affari, le possibilità sono 2:

-Comprare automobili

-Vendere automobili

Ogni utente avrà la possibilità di pubblicare un annuncio contenente la macchina da vendere e successivamente potrà essere contattato da altri utenti per poter vedere l’auto di persona e trattare il prezzo per poi concludere un buon affare.

L’app è pensata per creare una sorta di mercato dove poter trovare qualsiasi tipologia di automobile, dalla più costosa alla più economica, dal nuovo all’usato, dalla supercar all’utilitaria.

La motivazione che sta dietro alla creazione di questo progetto è data dalla nostra passione per le automobili e la voglia di creare una connessione tra persone interessate al settore.



**Architettura del sistema**



Mobile Application:

Linguaggi Java, xml

Framework Retrofit

IDE Android studio

Web Application:

Linguaggi Html/css/typescript

Framework Angular, Bootstrap

Editor Visual studio code



Front-End(user interface)

Back-End(servizi web)

Business Logic:

Linguaggi Java

Framework Jpa, SprinBoot

IDE Eclipse

RDBMS

SQL Server

Per la realizzazione di Carbook, abbiamo utilizzato la  Service-Oriented Architecture (SOA) , indica generalmente un'[architettura software](https://it.wikipedia.org/wiki/Architettura_software) adatta a supportare l'uso di [servizi Web](https://it.wikipedia.org/wiki/Web_service) per garantire l'[interoperabilità](https://it.wikipedia.org/wiki/Interoperabilit%C3%A0) tra diversi sistemi così da consentire l'utilizzo delle singole [applicazioni](https://it.wikipedia.org/wiki/Applicazione_(informatica)) come componenti del processo di business e soddisfare le richieste degli utenti in modo integrato e trasparente.

**Passaggi operativi**

A livello operativo abbiamo implementato l’applicativo partendo dai wireframe per stabilire quali fossero le idee iniziali sulla struttura grafica dell’applicazione.

WIREFRAME ANDROID

* Login
* [Registrazione](https://wireframe.cc/NjuWP9)
* [Profilo](https://wireframe.cc/5QKU0w)
* [RegistraA](https://wireframe.cc/kciHSi)nnuncio
* [Homepage](https://wireframe.cc/w54CQa)
* [DettaglioAnnuncio](https://wireframe.cc/ROt9cU)
* [Cerca](https://wireframe.cc/1GFHzV)
* [Recensioni](https://wireframe.cc/rulkCn)
* [ReimpostaPassword](https://wireframe.cc/2lg8U3)

WIREFRAME WEB

* [Login](https://wireframe.cc/mQ6iSE)
* [Registrazione](https://wireframe.cc/9ERon8)
* [Profilo](https://wireframe.cc/5QKU0w)
* [RegistraAnnuncio](https://wireframe.cc/XWSdeu)
* [DettaglioAnnuncio](https://wireframe.cc/rt8oHJ)
* [RecuperaPassword](https://wireframe.cc/0a3EqP)
* [Cerca](https://wireframe.cc/wCUvr4)

Dopo di che abbiamo implementato un modello ER che è un modello teorico per la rappresentazione concettuale e grafica dei [dati](https://it.wikipedia.org/wiki/Dati) a un alto livello di [astrazione](https://it.wikipedia.org/wiki/Astrazione_(informatica))

DIAGRAMMA ER

* [Diagramma ER](https://github.com/LucaNastasi/LibroAuto/blob/main/diag-er.png)

Successivamente ci siamo dedicati alla stesura dello use case, un diagramma dedicato alla descrizione delle funzioni o servizi offerti da un [sistema](https://it.wikipedia.org/wiki/Sistema), così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso.

USE CASE

* [Use Case](https://github.com/LucaNastasi/LibroAuto/blob/main/UseCase.png)

Per quanto riguarda la scrittura del codice, abbiamo iniziato implementando una semplice schermata di login creata usando Il linguaggio HTML che è un linguaggio di markup utilizzato per la costruzione di pagine internet.  
È utilizzato principalmente per il disaccoppiamento della struttura logica di una [pagina web](https://it.wikipedia.org/wiki/Pagina_web) (definita appunto dal markup) e la sua rappresentazione, gestita tramite gli stili [CSS](https://it.wikipedia.org/wiki/CSS).

Parallelamente abbiamo creato un progetto facendo uso di Spring Boot che è uno strumento del [framework](https://it.wikipedia.org/wiki/Framework) [open source](https://it.wikipedia.org/wiki/Open_source) Spring per lo sviluppo di [applicazioni](https://it.wikipedia.org/wiki/Applicazione_(informatica)) su [piattaforma Java](https://it.wikipedia.org/wiki/Piattaforma_Java).

Dopo aver settato il progetto con le varie dipendenze(Spring Web, Spring Data JPA e MS SQL server driver) e averlo collegato al nostro DB abbiamo iniziato ad implementare le classi modello utilizzando la convenzione JavaBean, le convenzioni richieste per una classe JavaBean sono:

* La classe deve avere un [costruttore](https://it.wikipedia.org/wiki/Costruttore_(informatica)) senza argomenti;
* Le sue proprietà devono essere accessibili usando get, set.
* La classe dovrebbe essere [serializzabile](https://it.wikipedia.org/wiki/Serializzazione) (capace di salvare e ripristinare il suo stato in modo persistente);

Una volta creato il database facendo partire l’applicazione ci siamo occupati della creazione del servizio web RESTFUL implementando un rest controller per ogni entità modello creata.

**REST (***Representational State Transfer***)** è un modello architetturale, che permette la comunicazione/interoperabilità tra sistemi remoti, tipicamente connessi ad Internet.

La sua caratteristica principale risiede in risorse che devono essere soddisfatte secondo i seguenti requisiti:

* Indirizzabilità: ogni risorsa, per esempio una prenotazione, un prodotto o un articolo, deve poter essere identificata tramite un URL.
* Interfaccia unica: deve essere possibile accedere a ogni risorsa in modo semplice e unitario con l’aiuto di metodi standard. Per esempio metodi HTTP come GET, POST o PUT.
* Struttura client-server: in generale vale il principio client-server, secondo il quale un server mette a disposizione un servizio, che se necessario può essere richiesto da un client.
* Assenza di stato: la comunicazione tra server e client non presenta alcuno stato. Questo significa che tutti i messaggi scambiati contengono tutte le informazioni necessarie per poterli leggere. Il server non memorizza informazioni aggiuntive tra i due messaggi come per esempio sotto forma di sessioni. L’assenza di uno stato rende i servizi REST altamente scalabili, in quanto le richieste in entrata possono essere distribuite facilmente ai diversi server tramite bilanciamento del carico.
* Diverse rappresentazioni delle risorse: ogni risorsa può avere diverse forme di visualizzazione. In base a cosa il client richieda, deve poter essere consegnato in diversi linguaggi o formati come HTML, JSON o XML.
* Hypermedia: la messa a disposizione delle risorse avviene tramite hypermedia, per esempio sotto forma di attributi “href”- e “src” nei documenti HTML o tramite elementi JSON e XML, definiti per le rispettive interfacce. Di conseguenza il client usa un’API REST esclusivamente tramite URL.

In seguito cominciava a prendere forma l’applicazione distribuita su OS Android. Android è un sistema operativo(scritto in C) per dispositivi mobili, sviluppato da Google ed organizzato in un'architettura software che include un sistema operativo di base, i middleware per le comunicazioni e le applicazioni di base, è un sistema embedded progettato principalmente per smartphone.

Le applicazioni si basano su elementi denominati Activity che cooperano tra loro o che possono essere indipendenti l’una dalle altre, le activity sono le schermate delle applicazioni.

Le prime schermate da noi implementate sono state quelle di login e registrazione inizialmente grazie all’utilizzo dell’interfaccia grafica per poi passare a scrivere direttamente codice in XML(altro linguaggio di markup). Per la registrazione e il login dell’utente , inizialmente abbiamo creato un DB SQLite per poter salvare i dati localmente. Una volta introdotti i servizi web, grazie alla libreria retrofit, che consente di accedere ai servizi REST semplicemente decorando con opportune annotazioni i metodi delle interfacce che rappresentano i servizi da utilizzare, abbiamo implementato le varie richieste al back-end che poi interrogherà il DB remoto.

Successivamente siamo andati a creare lo scheletro della nostra applicazione che implementa una bottom navigation che permette di navigare all’interno dell’app, passando dalla schermata di pubblicazione di un annuncio dove tramite una POST request salviamo i dati dell’automobile per poi visualizzarli nella home dove troviamo la lista degli annunci e infine si può passare nella sezione profilo dove si possono visualizzare in dettaglio i dati dell’utente stesso.

Per la parte web dell’ applicazione abbiamo fatto uso di Angular, un framework open-source utilizzato per creare applicazioni web dinamiche grazie alle sue molteplici funzionalità, il compilatore AOT, e il Code Splitting e altre, semplifica lo sviluppo e il tempo e garantisce ottimi risultati a livello di prestazioni. Tra i linguaggi che sono presenti su Angular siamo andati ad utilizzare l’HTML, CSS e il Typescript(un’estensione di javascript).

Come prima cosa, siamo andati a generare il nostro primo componente, il Login con il Sign up integrato all’interno, cos’è un componente? È una piccola unità utilizzabile e componibile con altre, e ciò mi permette di suddividere la mia applicazione in tanti componenti. Ogni componente è composto da quattro file “nome.component.html”, “nome.component.css”, “nome.component.spec.ts”, “nome.component.ts”. Nel file con estensione HTML, andiamo a scrivere il linguaggio HTML, con l’ausilio di Bootstrap, una raccolta composta da modelli di progettazione basati su HTML e CSS già realizzati, e ciò mi velocizza la composizione grafica della mia pagina. Nel file CSS definisco delle classi che specificano lo stile dei tag HTML. Nel file Typescript utilizzo il linguaggio Typescript, che viene compilato in Javascript per essere interpretato dal web browser, e posso utilizzarlo per creare classi, metodi e definire variabili. Ritornando al nostro Login-Sign up component, nel suo file HTML ho introdotto un form per il login e un modal all’interno di un paragrafo per il Sign up. I 2 campi input text di login(email e password) li ho definiti con un NgModel all’interno dei loro Tag, questo permette di modificare le variabili dichiarate nel Typescript in base a quello che scrive l’utente passandogli i due valori, di conseguenza, quando l’utente premerà il pulsante “Accedi”, verrà chiamato un metodo, creato nel Typescript, che manderà questi valori al Database alla tabella “Persona”, tramite una richiesta Http Post, posso verificare se esiste un match di tali valori tra le persone registrate, e ovviamente ciò determina l’accesso alla pagina successiva.

Per il Sign up abbiamo sempre utilizzato una richiesta Http Post, quando l’utente premerà sul pulsante “Registrati”, verrà richiamato un metodo che manderà al Database i valori immessi dall’utente creando un nuovo record nella tabella “Persona”.

Come spiegato poc'anzi, siamo andati a creare altri componenti per rendere funzionale il programma finale. Abbiamo quindi fatto varie interfacce, come: -il recupero della password; - primapagina; questa esegue una richiesta get che permette di cercare l'auto interessata (richiesta http get); - p-annuncio, qui si crea un'inserzione dove si possono caricare foto e dati (inserimento da tastiera da parte dell'utente). Una volta salvata, questa sarà visualizzabile a tutti gli utenti che cercheranno l'auto in questione; - visioneannuncio, è il component che permette all'inserzione di essere visibile a tutti gli utenti (questa pagina non è modificabile da parte di utenti diversi da quello che ha creato l'annuncio).

Oltre ai componenti abbiamo creato la cartella Model con le entità che utilizziamo per i salvare/richiedere dati del Database, “automobile”, “persona”, “pubblicazione”.

Infine per creare i vari url della pagine abbiamo usufruito del “App-routing-module.ts” definendo i path dei component.

**Conclusioni**

Inizialmente Carbook era nata con l’idea di essere un social per appassionati di automobili, tuttavia grazie ad alcuni confronti ci siamo accorti che l’idea della compravendita di vetture poteva risultare la scelta vincente e su misura per le nostre capacità.

L’applicazione risulta facilmente navigabile e incentiva l’utente all’iscrizione.

**Sviluppi futuri**

Nell’immediato futuro vorremmo implementare:

* + Una sezione filtri dove poter cercare gli annunci in base ai requisiti delle auto.
  + Visualizzare gli annunci di una persona all’interno del corrispettivo profilo.
  + La possibilità di effettuare recensioni sui venditori
  + La possibilità di modificare i dati dell’utente e dell’annuncio
  + L’implementazione di foto