System Desing Document

FIX-EXPERT

Studenti:

De Litteris Domenico 0124002204

Frascogna Antonio 0124002187

Cucinella Paolo 0124002167

Nasto Vincenzo 0124001748

Indice

Capitolo 1

1.1Scopo del sistema

Lo scopo di Fix-Expert sta nel fornire un app funzionale e semplice da usare per la gestione dell’officina per i dipendenti e non solo anche per i clienti vi sono funzionalità utili.

1.2Obiettivi di progettazione

- Usabilità
  FixExpert l’app dovrebbe essere intuitiva da usare e l’interfaccia utente dovrebbe essere semplice da capire
- Portabilità
  FixExpert dovrebbe essere un sistema multi-piattaforma
- Sicurezza
  FixExpert dovrebbe essere sicuro, cioè non deve consentire l’accesso ad utenti non registrati nell’app.

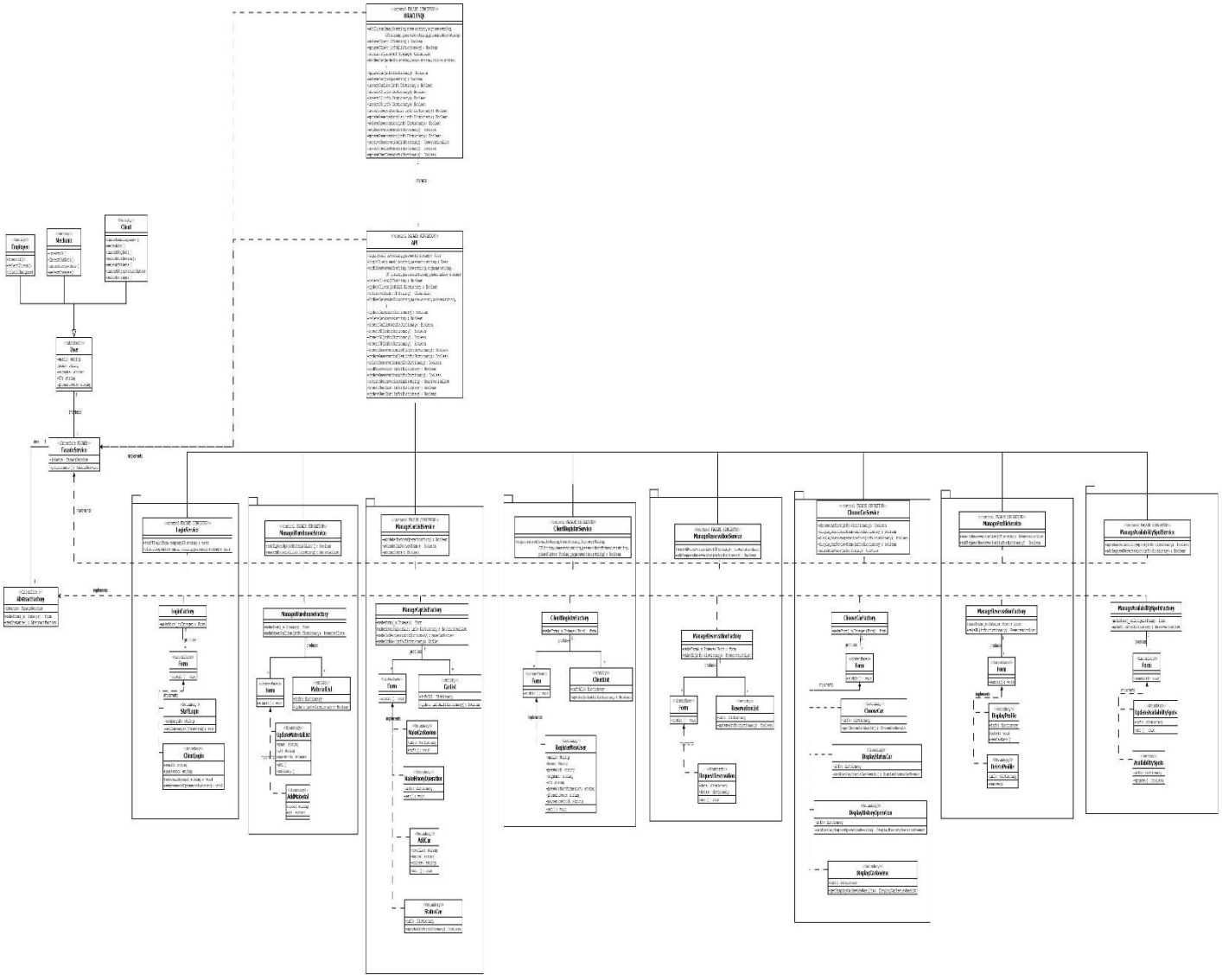
1.3 Definizioni

All’interno di questo file del System design Document, si possono incontrare la seguente terminologia

- MaterialList
  E’ la cartella che contiene la lista dei materiali a disposizione dell’officina in questione.
- CarList
  E’ una cartella che contiene la lista che con la lista delle auto presenti nell’officina in questione.
- ClientList
  E’ la cartella che contiene la lista dei clienti dell’officina.
- ReservationList
  E’ una cartella che contiene la lista delle richieste di prenotazione da parte dei clienti.

1.4 Panoramica

Diagramma delle classi con un ulteriore raffinamento di quanto ottenuto nella fase di analisi.



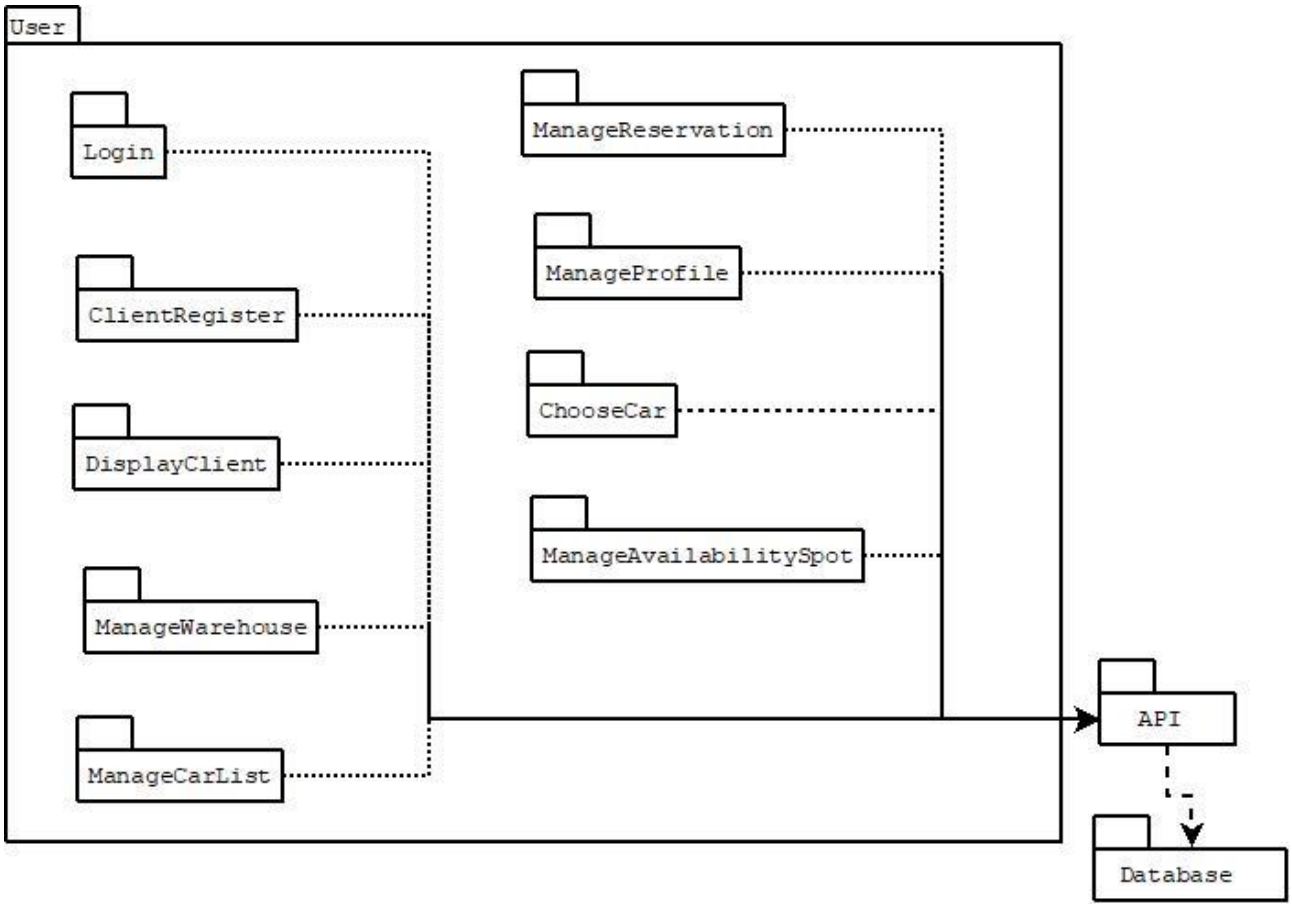
1.5Current software architecture

Fix Expert è un progetto greenfield, ovvero non si basa su un’architettura già esistente.

1.6Proposed software architecture

Lo stile architetturale scelto è il client/server, ovvero quei sistemi che utilizzano un'architettura nella quale generalmente un client si connette ad un server per usufruire di un certo servizio. Il quale permette la gestione delle macchina, dei clienti e dei materiali e la gestione del magazzino da parte dell'Employee, mentre anche per il Mechanic c'è la gestione del magazzino, con le funzionalità di aggiungere macchine in officina, fare revisioni,lo storico delle operazioni e cambiare lo status della macchina qualora il client volesse vedere tramite app se il veicolo è in lavorazione o è conclusa. Il server dispone di un API(Application programming interface) che permette di fare una serie di servizi ai client,mechanic ed employee e inoltre gestisce l'accesso al database. La scelta dello stile architetturale client/server è dovuta al fatto che così si garantisce una maggiore usabilità visto che ci sono vari utenti e inoltre rendere possibile l'accesso tramite qualunque dispositivo si voglia.

1.7 Subsystem decomposition



Login area del sistema che permette l'accesso di un User;

ClientRegister area dove l'user in questo caso il cliente può registrarsi all'interno dell'applicazione;

DisplayClient all'interno del quale vengono visualizzati la lista dei clienti registrati all'interno dell'applicazione e quindi dell'officina in questione.

ManageWarehouse area del sistema che permette di far gestire i materiali presenti in officina e la loro visualizzazione per Employee e Mechanic;

ManageCarList area del sistema che permette di far gestire la liste degli autoveicoli presenti in officina, l'aggiunta di nuovi autoveicoli, e le varie operazioni (MakeHistoryOperation,ChangeStatusCar,MakeCArReview) da parte del Mechanic;

ManageReservation area del sistema che permette di gestire all'Employee la liste delle richieste di prenotazioni effettuate dai Client;

ManageProfile area di accesso per il Client ove può visualizzare, cancellare o modificare il proprio profilo;

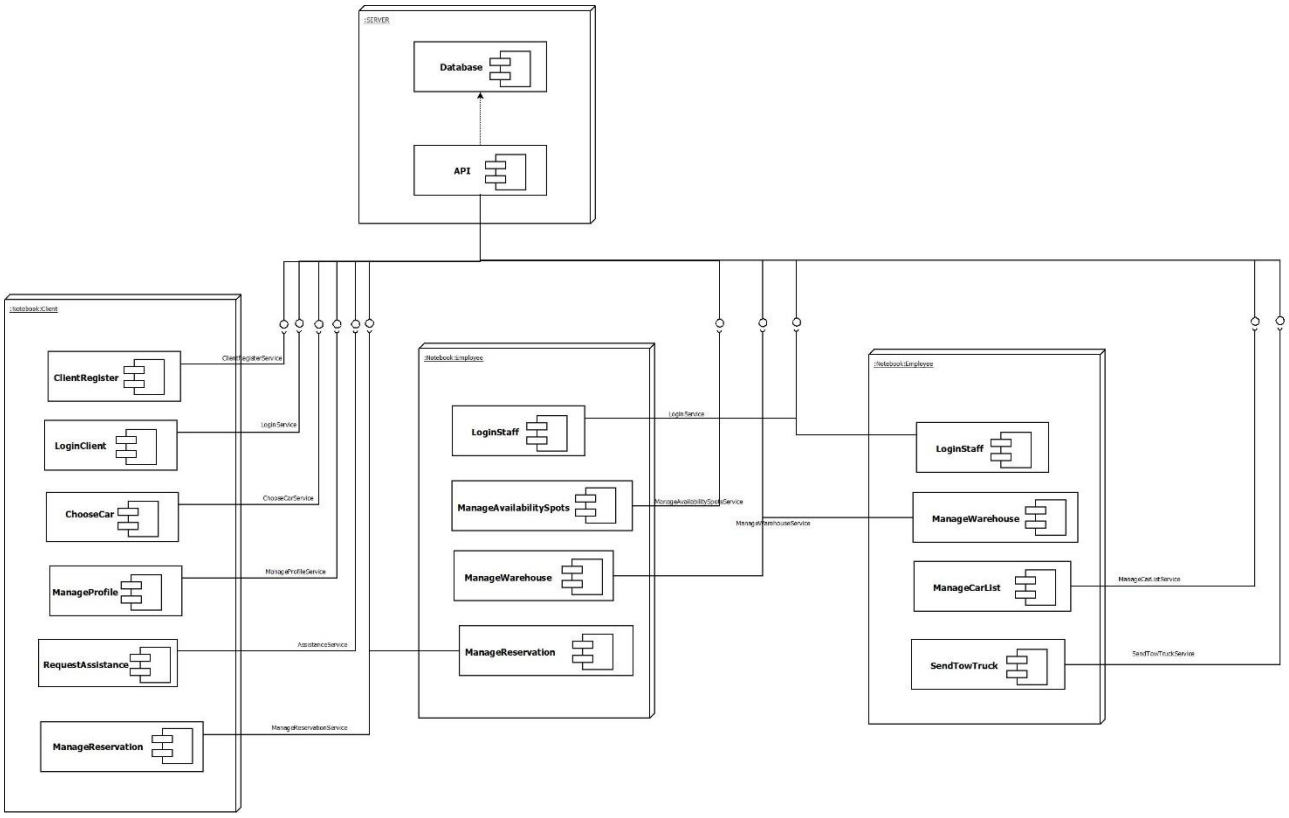
ChooseCar area di accesso per il Client ove può selezionare i suoi autoveicoli e visualizzare le varie informazioni relative alle operazioni effettuate sui propri autoveicoli.

ManageAvailabilitySpots all’interno del quale vengono gestiti la disponibilità dei posti in officina da parte dell’Employee.

API sottosistema facente da ponte tra i client,employee, mechanic ed il server. Serve per semplificare la possibilità di integrazione tra un’applicazione e un’altra evitando ridondanze e inutili replicazioni di codice. Ed offre un’interfaccia alle funzioni effettuabili.

Database ove vi è il salvataggio dei dati di sistema.

1.8 Hardware/Software mapping



1.9 Persistent data management

La Gestione dei dati persistenti data la mole di dati e di informazioni che FIXEXPERT deve supportare, ha portato alla scelta di un database, al fine di mantenere traccia dei dati riguardanti gli autoveicoli ossia la lista,le riparazioni le varie operazioni effettuate sugli autoveicoli, la lista delle prenotazioni poi i dati correlati ad un particolare utente che sia Client,Employee o Mechanic.

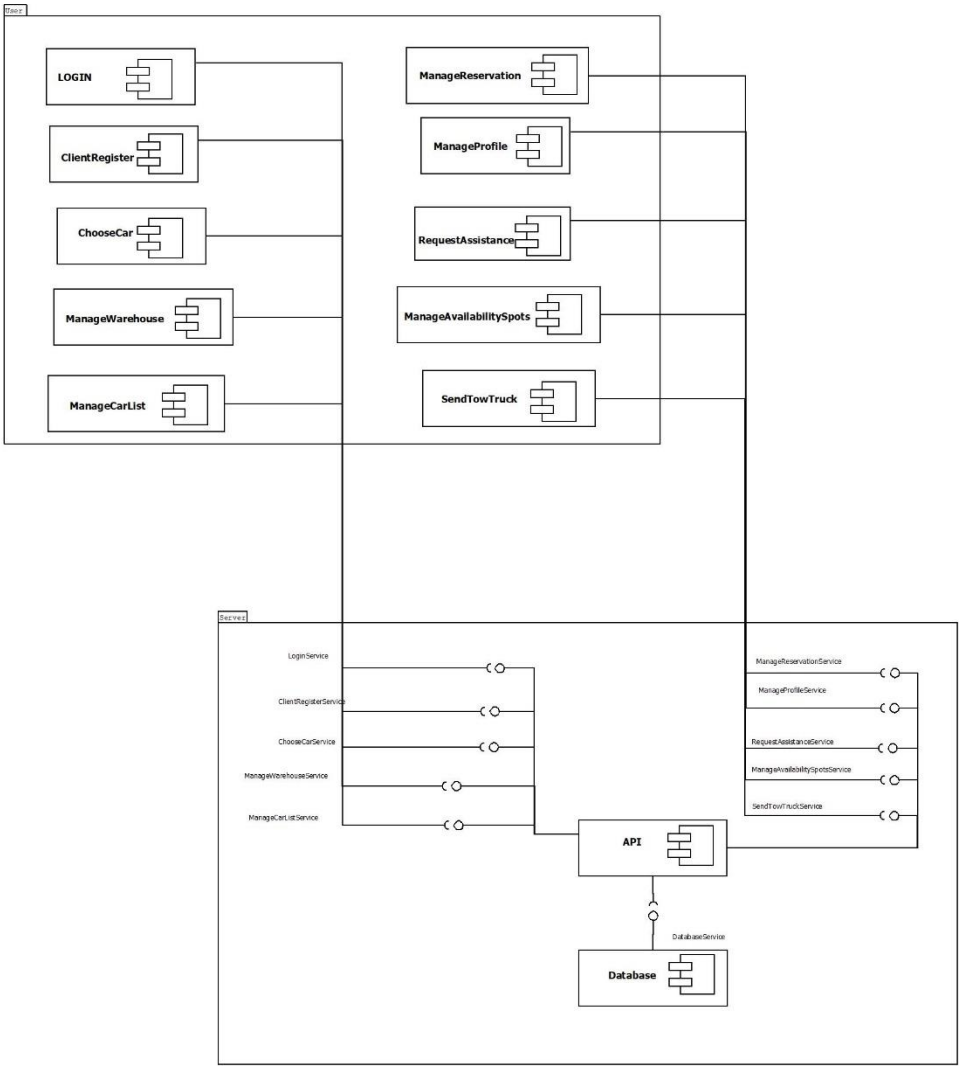
1.10 Access control and security

Oggetti/attori	Client	Employee	Mechanic
LoginService	Loginclient()	LoginStaff()	LoginStaff()
ManageWarehouseService	—	configueUpdateMaterialList() searchMaterialList()	configueUpdateMaterialList() searchMaterialList()
ManageCarListService	—	—	addMakeHistoryOperation() addMakeCarReview() addCar()
ClientRegisterService	Registration()	—	—
ManageReservationService	ViewMyreservation() RequestReservation() UpdateReservation() DeleteReservation()	MangeReservation()	—
ChooseCarService	ChooseCar()DisplayStatusCar() DisplayHistoryoperation() DisplayCarReview() searchCarView()	—	—
ManageProfileService	searchReservationList() addRequestReservation()	—	—
ManageAvailabilitySpotsService	—	updateavailabilitySpots()	—
AssistanceService	addRequestAssistance()	—	—

1.11 Decisioni sul flusso di controllo globale

Visto l’elevato numero di richieste sottomesse al server porta alla naturale scelta di un flusso di controllo thread based. Ciò comporta ad una necessaria gestione della concorrenza di tali richieste. Il pattern Facade, utilizzato tra gli altri con l’oggetto API, individua quindi un thread control che deve garantire l’accesso da parte di più utenti in modo concorrente ai servizi del server.

1.12 Subsystem Service



1.13 Glossary

- **User:** Entità generalizzata, è la persona che interagisce direttamente con il prodotto software. Non corrisponde necessariamente al cliente
- **Employee:** Entità che specializza User e rappresenta i metodi e gli attributi specifici dell'Employee.
- **Mechanic:** Entità che specializza User e rappresenta i metodi e gli attributi specifici del Mecanic.
- **FacadeService:**Interfaccia che conosce le classi nel sottosistema che sono responsabili di una richiesta e che delega la richiesta del client agli oggetti appropriati e fornisce un punto di accesso per ogni sottosistema.
- **LoginService:** Entità ove vi è il punto di accesso di login e ove sono implementati le funzionalità. Si è scelto il pattern Singleton poiché può impedire che da una classe possa essere creato più di un oggetto, e può mantenere un controllo stretto sulle

modalità con le quali i client vi possono accedere.

- **ManageWarehouseService:** Entità che rappresenta il punto di accesso al sotto-sistema ManageWarehouse e ne implementa le funzionalità. Si è scelto il pattern Singleton poiché può impedire che da una classe possa essere creato più di un oggetto, e può mantenere un controllo stretto sulle modalità con le quali i client vi possono accedere.
- **ManageCarListService:** Entità che rappresenta il punto di accesso al sotto-sitstema ManageCarList e ne implementa le funzionalità. Si è scelto il pattern Singleton poiché può impedire che da una classe possa essere creato più di un oggetto, e può mantenere un controllo stretto sulle modalità con le quali i client vi possono accedere.
- **ClientRegisterService:** Entità che rappresenta il punto di accesso al sotto-sistema

ClientRegister e ne implementa le funzionalità. Si è scelto il pattern Singleton poiché può impedire che da una classe possa essere creato più di un oggetto, e può mantenere un controllo stretto sulle modalità con le quali i client vi possono accedere.

- **ManageReservationService:** Entità che rappresenta il punto di accesso al sotto-sistema ManageReservation e ne implementa le funzionalità. Si è scelto il pattern Singleton poiché può impedire che da una classe possa essere creato più di un oggetto, e può mantenere un controllo stretto sulle modalità con le quali i client vi possono accedere.
- **ChooseCarService:** Entità che rappresenta il punto di accesso al sotto-sistema ChooseCar e ne implementa le funzionalità. Si è scelto il pattern Singleton poiché può impedire che da una classe possa essere creato più di un oggetto, e può mantenere un controllo

stretto sulle modalità con le quali i client vi possono accedere.

- **ManageProfileService:** Entità che rappresenta il punto di accesso al sotto-sistema ManageProfile e ne implementa le funzionalità. Si è scelto il pattern Singleton poiché può impedire che da una classe possa essere creato più di un oggetto, e può mantenere un controllo stretto sulle modalità con le quali i client vi possono accedere.
- **ManageAvailabilitySpotService:** Entità che rappresenta il punto di accesso al sotto-sistema ManageAvailability e ne implementa le funzionalità. Si è scelto il pattern Singleton poiché può impedire che da una classe possa essere creato più di un oggetto, e può mantenere un controllo stretto sulle modalità con le quali i client vi possono accedere.
- **AssistanceService:** Entità che rappresenta il punto di accesso al sotto-sistema Assitance e

ne implementa le funzionalità. Si è scelto il pattern Singleton poiché può impedire che da una classe possa essere creato più di un oggetto, e può mantenere un controllo stretto sulle modalità con le quali i client vi possono accedere.

- **AbstractFactory:** Si è scelto di utilizzare il pattern AbstractFactory poiché è in grado fornire un'interfaccia per creare famiglie di oggetti correlati o dipendenti senza specificare le loro classi concrete limitandone quindi l'uso diretto.
- **Form:** E' un interfaccia implementata da oggetti che rappresentano form.
- **LoginFactory:** Implementa AbstractFactory e crea i form necessari per il Login per lo Staff o Client.
- **LoginStaff:** Form necessario per il login dell'Employee o Mechanic.
- **LoginClient:** Form necessario per il login dei Client.

- **ManageWarehoseFactory:** Implementa AbstractFactory e crea i form necessari per l’aggiornamento dei materiali e l’aggiunta dei materiali. Inoltre, permette la creazione di oggetti di tipo MaterialList.
- **MaterialList:** Classe che rappresenta e contiente tutte le informazioni inerenti ai materiali presenti in officina.
- **UpdateMaterialList:** Form necessario per l’aggiornamento dei materiali.
- **AddMaterial:** Form necessario per l’aggiunta di materiali.
- **ManageCarListFactory:** Implementa AbstractFactory e crea i form necessari per fare le revisioni, creare lo storico di un auto,aggiungere un auto all’inventario e cambiare lo status dell’auto(in riparazione o conclusa). Inoltre permette la creazionei di oggetti tipo CarList.
- **CarList:** Classe che rappresenta e contiente tutte le informazioni inerenti alle auto presenti nell’officina.

- **MakeCarReview:** Form necessario per l'inserimento dei dati una volta effettuata la revisione.
- **MakeHistoryOperation:** Form necessario per la creazione dello storico degli autoveicoli.
- **AddCar:** Form necessario per l'aggiunta di auotoveicoli.
- **StatusCar:** Form necessario per il cambio di status dell'autoveicolo(in riparazione o conclusa).
- **ClientRegisterFactory:** Implementa AbstractFactory e crea i form necessari per la registrazione di un nuovo Cliente. Inoltre, permette la creazione di oggetti di tipo ClientList.
- **ClientList:** Classe che rappresenta e contiene le informazioni riguardanti la lista dei Clienti.
- **RegisterNewUser:** Form necessario per la registrazione di nuovi clienti.
- **ManageReservationFactory:** Implementa AbstractFactory e crea i form necessari per la richiesta di prenotazione in officina. Inoltre,

permette la creazione di oggetti di tipo ReservationList.

- **ReservationList:** Classe che rappresenta e contiene informazioni riguardati la lista di prenotazioni.
- **RequestReservation:** Form necessario per la richiesta di prenotazione da parte dei client.
- **ChooseCarFactory:** Implementa AbstractFactory e crea i form necessari per la scelta dell'auotveicolo, la visualizzazione dello stato dell'autoveicolo, dello storico e della revisione.
- **ChooseCar:** Form necessario per la scelta dell'autoveicolo da visualizzare.
- **DisplayStatusCar:** Form necessario che permette la visualizzazione dello status dei veicoli.
- **DisplayHistoryOperation:** Form necessario che permette la visualizzazione dello storico dei veicoli.
- **DisplayCarReview:** Form necessario che permette la visualizzazione delle revisioni.

- **ManageProfileFactory:** Implementa AbstractFactory e crea i form necessari la visualizzazione del profilo e la sua cancellazione.
- **DispalyProfile:** Form necessario che permette la visualizzazione del profilo richiesto.
- **DeleteProfile:** Form necessario per la cancellazione del profilo.
- **ManageAvailabilityFactory:** Implementa AbstractFactory e crea i form necessari per l'aggiornamento degli Spots disponibili in officina.Inoltre permette la creazione di oggetti di tipo AvailabilitySpots.
- **AvailabilitySpots:** Classe che rappresenta e contiente informazioni inerenti alla disponibilità in officina.
- **UpdateAvailabilitySpots:** Form necessario per l'aggiornamento dei posti disponibili in officina.

- **AssistanceFactory:** Implementa AbstractFactory e crea i form necessari per la richiesta del “TowTruck”.
- **RequestTowTruck:** Form necessario per la richiesta di un carroattrezzi.
- **ORACLESQL:** Entità che funge come punto di ingresso al database Oracle, e contiene le funzione che consentono di effettuare operazioni direttamente sul database.
- **API:** Entità che permette ai Service di interfacciarsi con il database fornendo tutte le funzioni atte a modificare l’interazione con il Database. Permettendo ai servizi di comunicare con altri prodotti o servizi senza che sia necessario sapere come vengono implementati, semplificando così lo sviluppo delle app e consentendo un netto risparmio di tempo e denaro.