

## Programmazione III e Laboratorio di Programmazione 3

Proff.: Angelo Ciaramella e Raffaele Montella

Anno Accademico 2020/2021



Studente: Gaetano Ippolito (0124001867)

Relazione del progetto d'esame di programmazione III:

My Delivery.



## Sommario

1.0 – Traccia d’esame.....	3
1.1 – Diagrammi.....	4
2.0 – MVC: Model .....	6
2.1 – Builder Pattern .....	7
2.2 – Observer Pattern .....	8
2.3 – Singleton Pattern .....	9
3.0 – MVC: Controllers .....	10
3.1 – Algoritmo NextFit.....	11
4.0 – MVC: Views .....	12
5.0 – Come utilizzare l’applicazione MyDelivery .....	14

## 1.0 – Traccia d'esame

### Traccia - Logistica

Si vuole sviluppare un'applicazione relativa alla consegna di merci nel campo della logistica. La logistica è l'insieme delle attività organizzative, gestionali e strategiche che governano i flussi di materiali e delle relative informazioni dalle origini presso i fornitori fino alla consegna dei prodotti finiti ai clienti e al servizio post-vendita.

Si suppone di avere diverse aziende di trasporto per consegnare la merce (corrieri). Ogni azienda ha a disposizione un numero di veicoli identificati da un codice, tipo veicolo e capienza container (numeri di colli che può contenere). Il collo è identificato da un codice, mittente, destinatario e peso. L'applicazione deve gestire il carico di N colli nei container. Per il riempimento si utilizza un algoritmo approssimato (Next Fit) che risolve il problema del Bin Packing (vedi documento allegato).

Il corriere, inoltre, aggiorna lo stato del collo ad ogni centro di smistamento, il quale deve essere rintracciato dal destinatario mediante il suo codice.

### Note di sviluppo.

La prova d'esame richiede la progettazione e lo sviluppo della traccia proposta. Lo studente può scegliere di sviluppare il progetto nelle due modalità: Applicazione Web o programma standalone con supporto grafico.

Il progetto deve essere sviluppato secondo le seguenti linee:

- usare almeno due pattern (almeno uno per chi sceglie la modalità Web Application) tra i design pattern noti;
- attenersi ai principi della programmazione SOLID;
- usare il linguaggio Java;
- inserire sufficienti commenti (anche per Javadoc) e annotazioni;
- gestione delle eccezioni;
- usare i file o database. Lo studente deve presentare una relazione sintetica (per chi usa latex è possibile scaricare un template dalla piattaforma e-learning).

La relazione deve contenere:

- una breve descrizione dei requisiti del progetto;
- il diagramma UML delle classi;
- altri diagrammi se opportuni;
- parti rilevanti del codice sviluppato.

## Consegna progetto.

La relazione e il codice del progetto devono essere messi a disposizione secondo le modalità ritenute più opportune (Dropbox, Google Drive, Piattaforma Sebeto, Pendrive, CD, . . .) entro la data di scadenza della prenotazione on-line dell'esame.

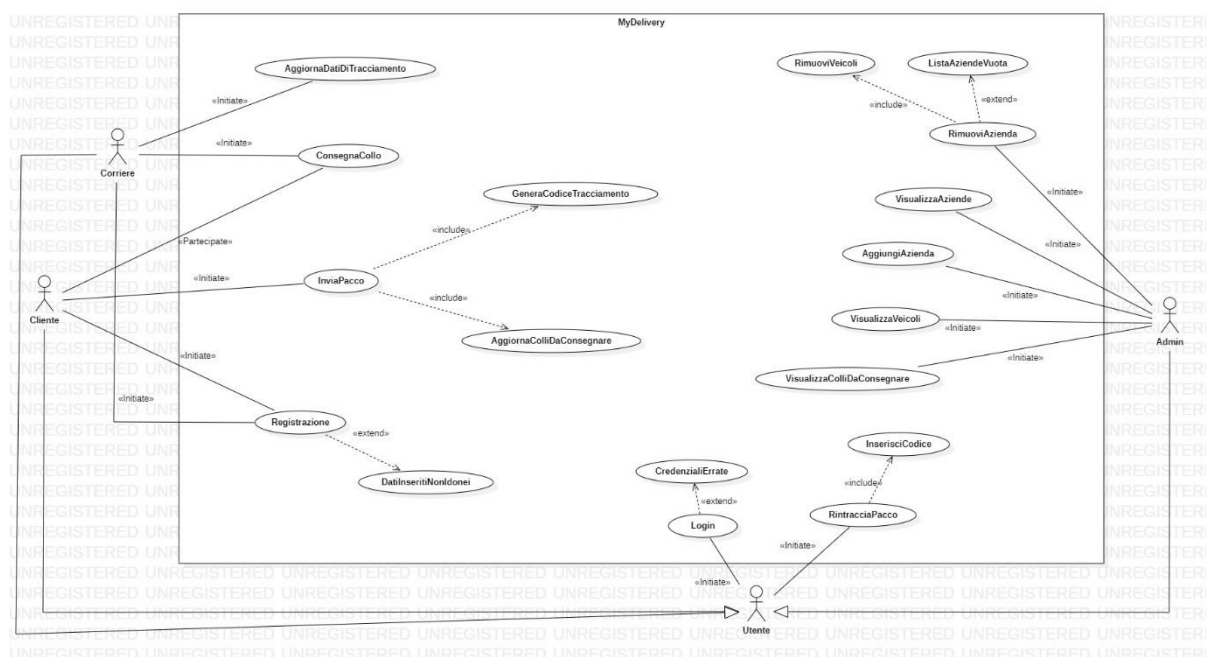
## Modalità di esame.

La prima parte della prova di esame verterà sulla discussione del progetto. Lo studente deve preparare una presentazione sintetica (slide) per descrivere il progetto svolto. La seconda parte della prova verterà sulla discussione degli argomenti affrontati a lezione.

### 1.1 – Diagrammi

Per permettere una scoperta dei requisiti precisa e che non permettesse in alcun modo di uscire fuori dai domini dell'applicazione, la decisione di creare un **Diagramma dei casi d'uso** come prima azione nei confronti del progetto si è rivelata piuttosto efficace.

Il Diagramma dei casi d'uso di cui si sta parlando è il seguente:



La lettura della traccia ha permesso la creazione di questo Diagramma e la scoperta dei seguenti attori:

- **Cliente**: che rappresenta colui che, oltre ad avere la possibilità di registrarsi ed effettuare un login sulla piattaforma MyDelivery, effettua l'ordinazione presso un **Azienda** (caso d'uso: Invia **Pacco**). L'ordinazione genererà un codice che servirà al **destinatario** per poter reperire tutte le informazioni relative all'ordine che deve ricevere.

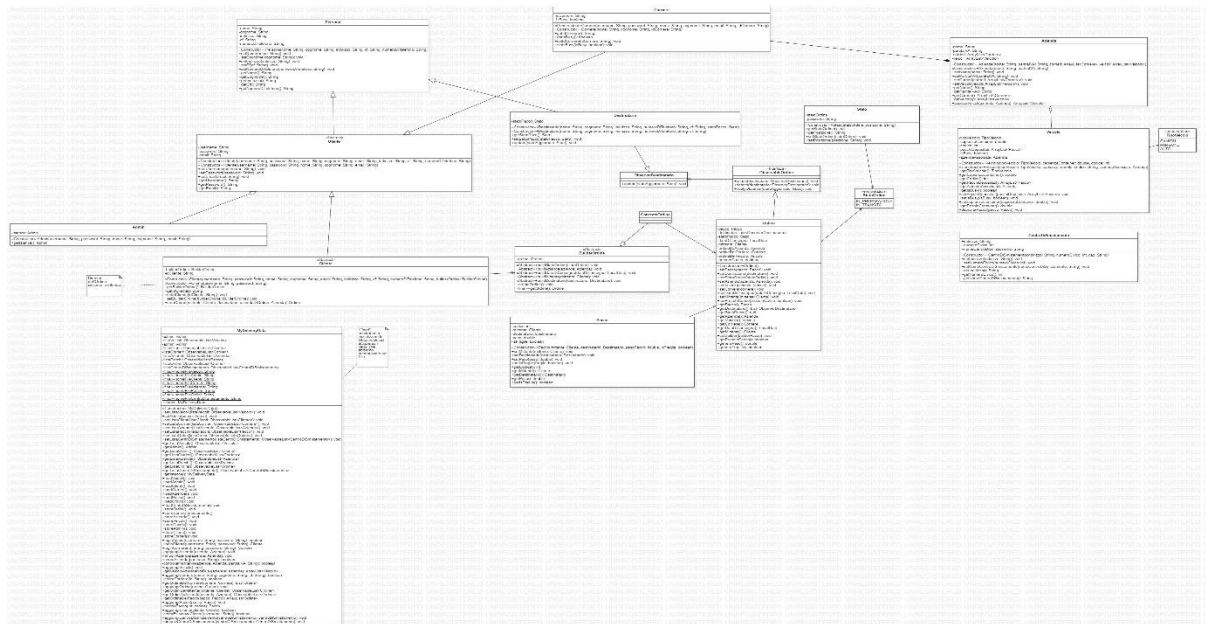
- **Corriere:** che rappresenta colui che prende in carico in un ordine presso l'azienda per cui lavora. È una figura importante dato che grazie a lui si ha il cambiamento di **stato** dell'ordine.
- **Admin:** che rappresenta una figura che gestisce e visualizza tutte le informazioni che fanno parte di MyDelivery. L'Admin è colui che gestisce le collaborazioni con le aziende, che visualizzerà i **veicoli** dell'azienda con cui collabora e visualizzerà varie informazioni riguardo i Colli che vengono generati quando viene creato un **Ordine**. Essendo colui che gestisce le collaborazioni con le Aziende, potrà anche rimuovere un'Azienda con cui non si hanno più rapporti. La sua esistenza è molto importante anche per gestire l'algoritmo **NextFit** imposto dalla traccia del progetto, siccome è una responsabilità dell'applicazione quella di eseguire il seguente algoritmo.

Dopo un'attenta analisi al diagramma dei casi d'uso, sono stati rivelati altre importanti figure all'interno del dominio di MyDelivery:

- Destinatario.
- Veicolo.
- Azienda.
- Centro di smistamento.

la quale hanno avuto maggiore importanza nella rappresentazione del Diagramma delle classi.

Avendo trovato tutte le informazioni necessarie dal Diagramma dei casi d'uso, il **Diagramma delle classi** si presenta in questo modo:

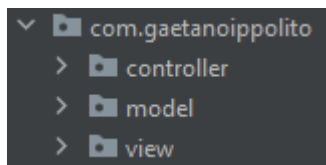


[Per una visualizzazione migliore del seguente Diagramma, seguire il seguente percorso: Documentazione -> UML -> DiagrammaDelleClassi -> JPEG -> ClassDiagram1.jpg].

## 2.0 – MVC: Model

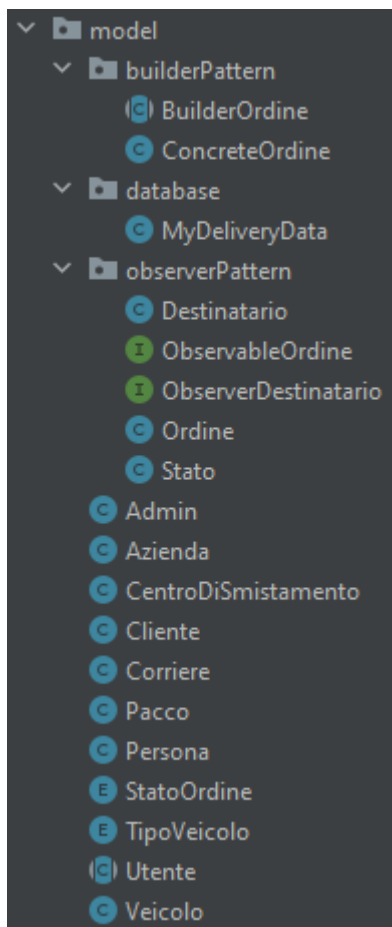
Il Model View Controller è un pattern architetturale che, rispetto ai design pattern, imposta l'organizzazione strutturale di un sistema software. Grazie a questo pattern architetturale, vengono descritti i ruoli che assumono i vari componenti software e le relazioni che hanno tra di loro.

Questo pattern architetturale è stato utilizzato per impostare la struttura del progetto. Infatti:



Tutte le entità che sono state scoperte durante le fasi di creazione del Diagramma delle classi e Diagramma dei casi d'uso, appartengono alla cartella "model". Infatti, all'interno della cartella "model" vanno tutte le classi il cui stato viene cambiato (operazione permessa dalle classi del Controller) oppure vi è una richiesta della visualizzazione dello stato cambiato (operazione permessa dalle classi del View).

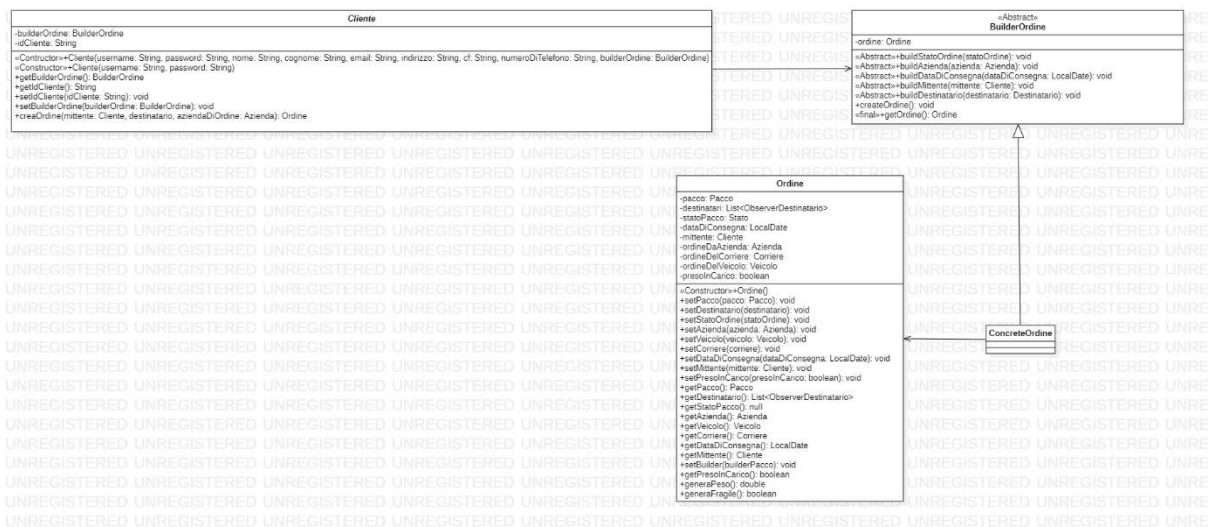
Al suo interno vi troviamo:



- Package "builderPattern", in cui vi sono le classi che collaborano nel funzionamento del pattern **Builder**.
- Package "database" in cui vi è la classe che astrae il concetto di persistenza dei dati.
- Package "observerPattern" in cui vi sono le classi che collaborano per il funzionamento del pattern **Observer**.
- Le classi che fungono da modello per il progetto, ovvero tutte le classi il cui stato va mostrato oppure modificato in base alle operazioni del Controller.

Anche se le altre classi come "Ordine", "Stato" e "Destinatario" si trovano in un Package loro, non vuol dire che non facciano parte del model. Infatti, la scelta è stata fatta per i fini organizzativi del progetto e per visualizzare meglio lo scopo di quelle classi in particolare.

## 2.1 – Builder Pattern



Per il progetto MyDelivery è stato utilizzato il pattern Builder.

Questo pattern è stato sfruttato per la creazione di un ordine da parte di un Cliente, essendo l'ordinazione un oggetto piuttosto complesso e bisognava separare la sua costruzione dalla sua rappresentazione per ottenere rappresentazioni sempre differenti.

La struttura del pattern possiede i seguenti elementi:

- Cliente: rappresenta il Director del Builder Pattern. Il Director ha lo scopo di costruire un oggetto di tipo Ordine sfruttando la classe astratta BuilderOrdine.
- BuilderOrdine: rappresenta una classe astratta utilizzata per la creazione del Product.
- ConcreteOrdine: il suo scopo è quello di costruire il Product in base ai metodi definiti nel BuilderOrdine.
- Ordine: rappresenta il Product del pattern da costruire.

```
/**
 * Questo è il metodo che avvia il Builder dell'ordine.
 * @param mittente Rappresenta il mittente dell'ordine da buildare
 * @param destinatario Rappresenta il destinatario dell'ordine da buildare
 * @param aziendaDiOrdine Rappresenta l'azienda dell'ordine da buildare
 * @return Ritorna un ordine buildato
 */
public Ordine creaOrdine(Cliente mittente, Destinatario destinatario, Azienda aziendaDiOrdine){
    Random random = new Random();
    int maxDays = 25;
    long randomDays = random.nextInt(maxDays);

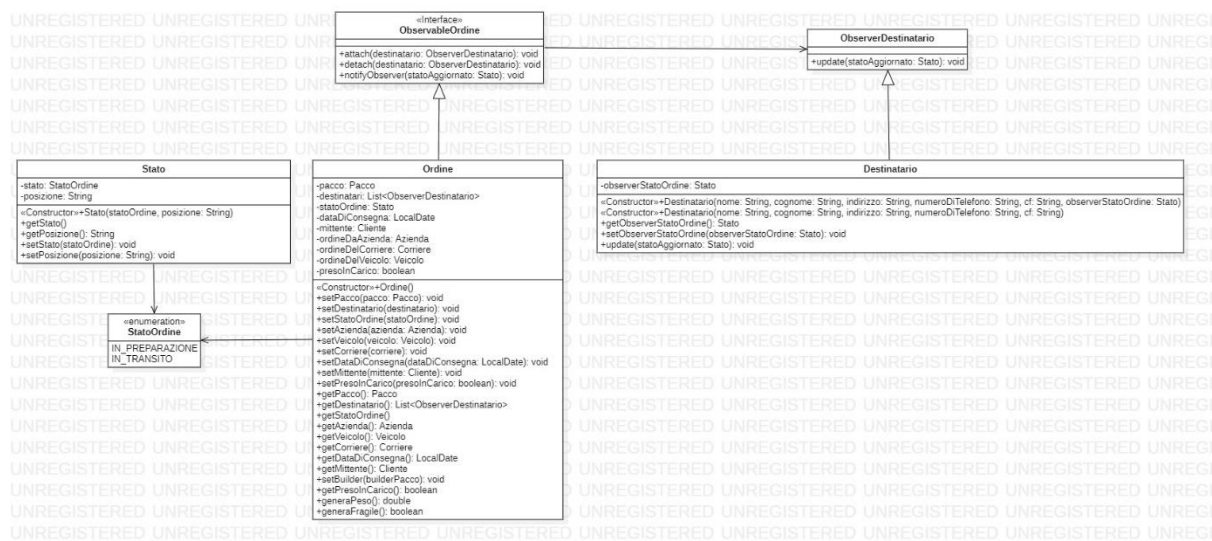
    this.builderOrdine.creaOrdine();
    this.builderOrdine.buildMittente(mittente);
    this.builderOrdine.buildDestinatario(destinatario);
    this.builderOrdine.buildStatoOrdine(new Stato(StatoOrdine.IN_PREPARAZIONE, posizione: "Deposito"));
    this.builderOrdine.buildDataDiConsegna(LocalDate.now().plusDays(randomDays));
    this.builderOrdine.buildAzienda(aziendaDiOrdine);

    return this.builderOrdine.getOrdine();
}
```

Nella classe “Cliente” è presente il metodo “creaOrdine”. Questo metodo è importante per la creazione di un Ordine da parte del Cliente.



## 2.2 – Observer Pattern



Per il progetto MyDelivery è stato sfruttato il pattern Observer.

Visto che il corriere ha la possibilità di aggiornare lo stato dell'ordine, l'Observer pattern si è rivelato un candidato ottimo, tra i vari pattern, per la costruzione del progetto MyDelivery.

Infatti, visto che l'ordine cambia lo stato in base al raggiungimento di un centro di smistamento, è solo grazie a questo pattern che ogni entità, che ha una dipendenza con l'ordine, viene notificata dei cambiamenti di stato dell'ordine.

La struttura del pattern possiede i seguenti elementi:

- **ObservableOrdine**: rappresenta l'interfaccia che consente agli osservatori di iscriversi (metodo "attach()") e cancellarsi (metodo "detach()"). Inoltre, mantiene una reference a tutti gli osservatori iscritti, in modo tale che se lo stato dell'ordine cambiasse tutti gli osservatori verrebbero notificati (metodo "notify()").
- **Ordine**: rappresenta l'oggetto concreto che deve essere osservato. Al suo interno vi è lo Stato che deve essere osservato che, in caso di cambiamenti, manda una notifica a tutti gli osservatori del suo cambiamento.
- **ObserverDestinatario**: rappresenta l'interfaccia che consente di aggiornare gli osservatori in caso di cambiamenti di stato dell'ordine (metodo "update()").
- **Destinatario**: rappresentata l'osservatore concreto che osserva un Observable. Al suo interno salva lo stato di ciò che sta osservando, la quale cambia in base ai cambiamenti dell'Observable.
- **Stato**: rappresenta lo Stato dell'Ordine da cambiare in base al raggiungimento di un centro di smistamento.

```
/**
 * Questo metodo notifica tutti gli observer dei cambiamenti di stato dell'ordine
 * @param statoAggiornato Rappresenta il cambiamento di stato da notificare a tutti gli observer
 * @see Destinataro
 * @see Stato
 */
@Override
public void notifyObserver(Stato statoAggiornato) {
    for(ObserverDestinatario destinatario : this.destinatari){
        destinatario.update(statoAggiornato);
    }
}
```

Questo metodo è presente all'interno della classe "Ordine". Tramite questo tutti i destinatari dell'ordine verranno notificati del cambiamento dello stato dell'ordine.



```

/**
 * Questo metodo aggiorna lo stato del pacco che il destinatario sta osservando
 * @param statoAggiornato Rappresenta lo stato aggiornato.
 */
@Override
public void update(Stato statoAggiornato){
    this.observerStatoOrdine = statoAggiornato;
}

```

Il metodo “notify()” precedentemente mostrato sfrutta il metodo “update()” presente nella classe “Destinatario”.

## 2.3 – Singleton Pattern

Per questo progetto è stato utilizzato il Pattern Singleton per la classe “MyDeliveryData”. Questa scelta è stata presa siccome bisognava fornire alla classe “MyDeliveryData”, che tramite i file simula il DB, una singola istanza da cui reperire tutte le informazioni che bisogna salvare all’interno del progetto.

MyDeliveryData è stato implementato con il pattern Singleton di tipo “Eager”:

```

// Eager Initialization
private static final MyDeliveryData instance = new MyDeliveryData();

```

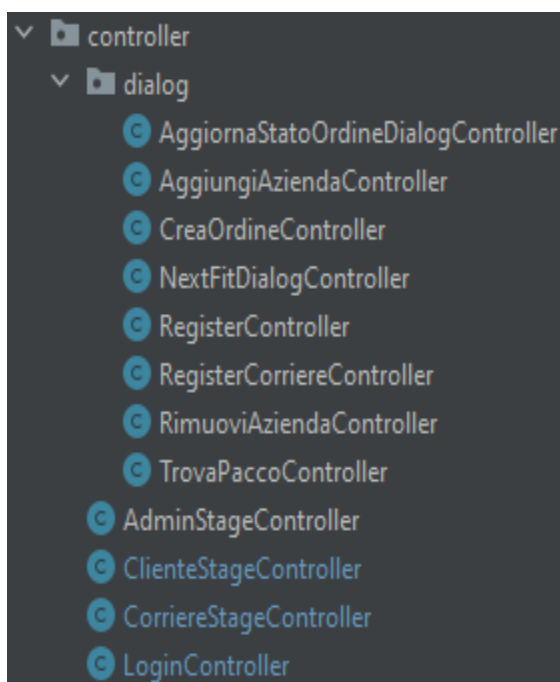
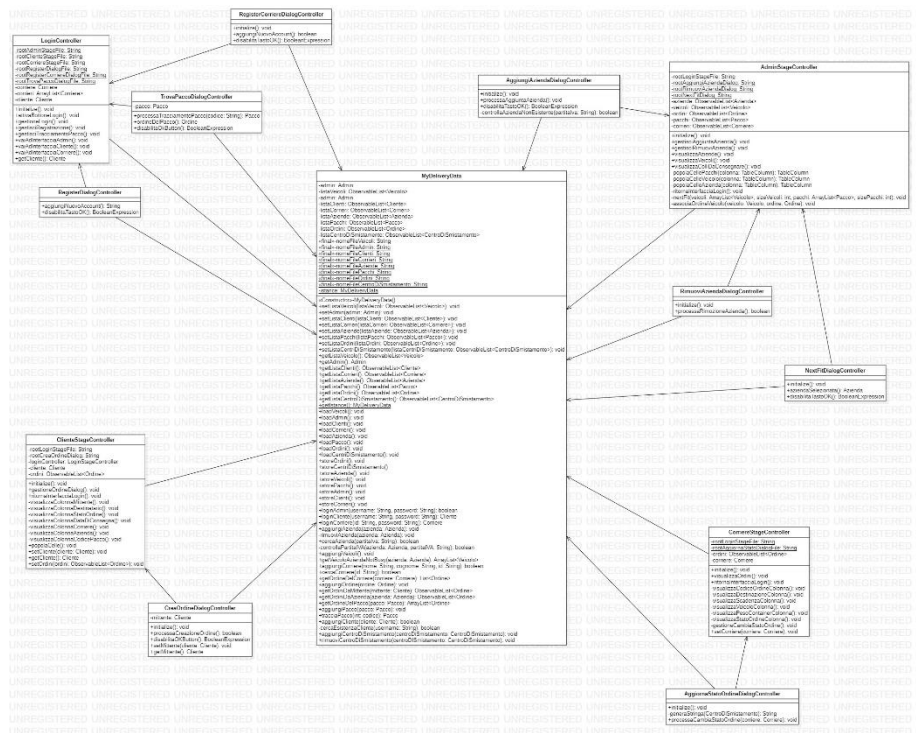
```

/**
 * Questo rappresenta il costruttore della classe "MyDeliveryData". Esso è privato siccome è stato applicato un
 * Singleton, infatti viene richiamato un'unica volta al momento dell'inizializzazione all'interno della classe
 * stessa. Ciò che fa al momento della creazione è quello di inizializzare le ObservableList.
 */
private MyDeliveryData(){
    this.aziende = FXCollections.observableArrayList();
    this.veicoli = FXCollections.observableArrayList();
    this.clienti = FXCollections.observableArrayList();
    this.corrieri = FXCollections.observableArrayList();
    this.ordini = FXCollections.observableArrayList();
    this.pacchi = FXCollections.observableArrayList();
    this.centriDiSmistamento = FXCollections.observableArrayList();
}

////////// GETTER //////////
/**
 * Metodo che ritorna l'unica istanza della classe (siccome è stato applicato un Singleton)
 * @return Ritorna l'istanza di MyDelivery
 */
public static MyDeliveryData getInstance(){
    return instance;
}

```

## 3.0 – MVC: Controllers



Sopra è rappresentato il diagramma di tutti i controller che sono stati creati per il progetto MyDelivery, mentre a sinistra vi è uno screen di dove sono contenuti i controllers. Avendo utilizzato il pattern architetturale MVC, i controller sono stati utilizzati per:

- Cambiare lo stato delle classi del Model.
- Selezionare le view da mostrare.
- Eseguire operazioni in base all'input dell'utente.
- Incapsulare il comportamento dell'applicazione.

Visto che i controller cambiano lo stato del model, la comunicazione con "MyDeliveryData" è importantissima per fare in modo che ciò avvenga.

Ogni Controller ha lo scopo di selezionare una view ed eseguire operazioni in base alla "view" correlata, per cui: "LoginController" rappresenta il controller della view "Login.fxml", quindi se l'utente sbaglia ad inserire una password, l'operazione verrà gestita dal controller che comunicando con la view gli dirà di mostrarci che la password inserita non è valida.

### 3.1 – Algoritmo NextFit

```
* Algoritmo NextFit
* Questo algoritmo accetta in input un ArrayList di veicoli (BlockSize), la size dell'ArrayList di veicoli,
* un ArrayList di Ordini (ProcessSize) e la size dell'ArrayList di Ordini.
* L'algoritmo prende in considerazione solamente quanto può depositare il veicolo al passo j-esimo e quanto
* pesa il pacco al passo i-esimo. In particolare "i" è minore della size dell'ArrayList dei pacchi, mentre "j"
* è minore della size dell'ArrayList dei veicoli.
* Dopo aver effettuato un controllo in cui: presa la capienza del veicolo al passo j-esimo sottratta dal peso di ciò
* che è stato depositato all'interno del container del veicolo al passo
* j-esimo, si controlla se sia maggiore del peso del pacco al passo i-esimo.
* Nel caso in cui if statement risulti essere true, allora il veicolo al passo j-esimo viene rimosso da MyDeliveryData
* per effettuare un Update; subito dopo viene depositato il pacco all'interno del veicolo in questione, per poi settare
* il veicolo come "Busy" (ovvero impegnato, che simboleggia un veicolo riempito e che non potrà più essere richiamato
* da un prossimo nextFit).
* Di seguito viene richiamato il metodo "associaOrdineVeicolo" in cui all'ordine al passo i-esimo viene associato il
* veicolo che sta venendo riempito di pacchi.
* Dopo aver eseguito l'associazione, viene richiamato un "break" siccome il veicolo può continuare a depositare pacchi,
* per cui l'indice "i" viene incrementato mentre l'indice j non viene incrementato, indicando che se riferisce ad un pacco
* di un ordine diverso che però viene depositato (nel caso in cui le condizioni fossero rispettate) sempre all'interno
* dello stesso veicolo.
* Nel caso in cui un veicolo NON può contenere il pacco oppure, avendo depositato vari pacchi, quello al passo i-esimo
* fa risultare la capienza del veicolo oltre la sua capacità, allora si entra nel blocco Else, in cui viene visualizzato
* nella console il peso del pacco j-esimo e si incrementa l'indice j. Questo non incrementerà l'indice i, in modo tale
* che il controllo venga effettuato al prossimo veicolo e non al prossimo ordine.
* @param veicoliNextFit Rappresenta un arrayList in cui sono contenuti solo veicoli che posso depositare pacchi
* @param sizeVeicoli Rappresenta la size dell'ArrayList dei veicoli
* @param ordiniNextFit Rappresenta un arrayList di Ordini in cui sono contenuti i pacchi da depositare
* @param sizeOrdini Rappresenta la size dell'ArrayList degli ordini
*/
private void nextFit(ArrayList<Veicolo> veicoliNextFit, int sizeVeicoli, ArrayList<Ordine> ordiniNextFit, int sizeOrdini){
    int j = 0;

    for(int i = 0; i < sizeOrdini; i++){
        while(j < sizeVeicoli){
            if((veicoliNextFit.get(j).getCapienzaContainer() - veicoliNextFit.get(j).getPesoInContainer()) >= ordiniNextFit.get(i).getPacco().getPesoPacco()){
                this.veicoli.remove(veicoliNextFit.get(j));

                veicoliNextFit.get(j).depositaPacco(ordiniNextFit.get(i).getPacco());

                veicoliNextFit.get(j).setIsBusy(true);

                associaOrdineVeicolo(veicoliNextFit.get(j), ordiniNextFit.get(i));

                break;
            }
            else{
                System.out.println("Veicolo: " + veicoliNextFit.get(j).getPesoInContainer() + " pieno");
            }

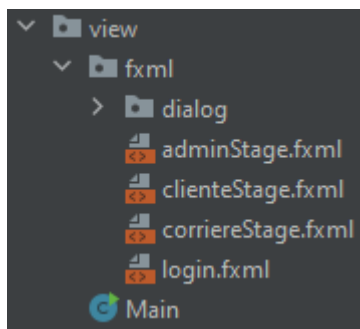
            j = (j + 1) % sizeVeicoli;
        }
    }
}
```

Questo algoritmo è stato utilizzato per risolvere il problema del Bin Packing, in cui:

si hanno a disposizione un numero N di oggetti di data misura (lunghezza, peso, o qualunque altra cosa) ed un numero infinito (o un qualsiasi numero maggiore di N) di contenitori (bins) di data capacità. Scopo del problema è inserire gli oggetti nel minore numero di contenitori possibile. Naturalmente la capacità dei contenitori deve essere tale da contenere anche il più grande degli oggetti dati.

L'algoritmo è responsabilità dell'applicazione, secondo la traccia, per cui la scelta più giusta è stata quella di applicare l'algoritmo all'interno di "AdminStageController" siccome è stato creato proprio per gestire l'applicazione. Il controller "AdminStageController" grazie a questo algoritmo può cambiare lo stato dei modelli che sono presi in considerazione (Veicolo, Ordine, Corriere) in modo tale da associare un numero N di pacchi ad un Veicolo, finché sarà possibile riempirlo.

## 4.0 – MVC: Views



La visualizzazione dello stato dei model e l'interfaccia grafica dell'applicazione MyDelivery è contenuta all'interno del Package "view".

Al suo interno troviamo "FXML", "dialog" e la classe Main. La classe Main rappresenta l'entry point dell'applicazione e dove viene mostrata la prima schermata "login.fxml".

Nel package "FXML" vi sono tutti i file con estensione "FXML" che servono per la realizzazione dell'interfaccia grafica

dell'applicazione. Infine, nel package "dialog" troviamo tutti quei file con estensione "FXML" che permettono la creazione dell'interfaccia delle finestre di dialogo che si aprono quando l'utente svolge determinate operazioni.

Per la realizzazione dell'interfaccia grafica è stato utilizzato "JavaFX", ovvero un cross-platform GUI toolkit per Java, creato per essere il successore delle librerie Java Swing.

Sfruttando il linguaggio di programmazione Java, con JavaFX è possibile creare interfacce grafiche per applicazioni Desktop. Per il progetto, infatti, è stato utilizzato nel seguente modo:

```
@FXML
public void initialize(){
    // Creazione dei MenuItem per il Menu "show"
    MenuItem listaAziendaItem = new MenuItem( s: "Lista Azienda");
    MenuItem listaVeicoliItem = new MenuItem( s: "Lista Veicoli");
    MenuItem listaColliDaConsegnareItem = new MenuItem( s: "Lista Colli Da Consegnare");

    // Creazione dei MenuItem per il Menu "Azienda" che si trova all'interno del Menu "Edit"
    MenuItem aggiungiAziendaItem = new MenuItem( s: "Aggiungi Azienda");
    MenuItem rimuoviAziendaItem = new MenuItem( s: "Rimuovi Azienda");

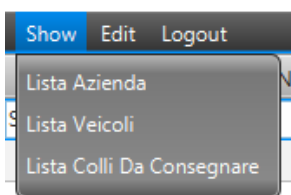
    // Creazione del MenuItem per il Menu "Exit"
    MenuItem exitItem = new MenuItem( s: "Exit..");

    // Creazione del MenuItem per il nextFit
    MenuItem nextFitItem = new MenuItem( s: "NextFit");

    // Dichiaro i Menu
    Menu show = new Menu( s: "Show");

    Menu edit = new Menu( s: "Edit");
    Menu azienda = new Menu( s: "Azienda");

    Menu logout = new Menu( s: "Logout");
```



Nel primo screen è mostrato il codice per creare un MenuBar all'interno dell'interfaccia "Admin". Il secondo screen mostra ciò che è stato mostrato grazie al codice all'interno del metodo "initialize()". I Menu "Show", "Edit" e "Logout" contengono al suo interno degli Item con cui fare varie operazioni, mostrato nel terzo screen.

I file “fxml” invece si presentano in questo modo

```
<center>
  <TableView fx:id="ordineView">
    <columns>
      <TableColumn fx:id="mittenteColonna" editable="false" text="Mittente" />

      <TableColumn fx:id="destinatarioColonna" editable="false" text="Destinatario" />

      <TableColumn fx:id="statoOrdineColonna" editable="false" text="Stato Ordine" />

      <TableColumn fx:id="dataDiConsegnaColonna" editable="false" text="In arrivo il.."/>

      <TableColumn fx:id="corriereColonna" editable="false" text="Corriere" />

      <TableColumn fx:id="nomeAziendaColonna" editable="false" text="Azienda" />

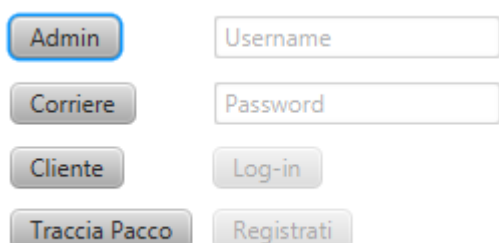
      <TableColumn fx:id="codicePaccoColonna" editable="false" text="Codice Pacco" />
    </columns>
  </TableView>
</center>
```

Questo screen in particolare è servito per creare le colonne della TableView, che si presenterà in questo modo:

Mittente	Destinatario	Stato Ordine	In arrivo il..	Corriere	Azienda	Codice Pacco
Gaetano Ippolito	Renato T	IN_TRANSITO	2021-01-26	Marco P	Samsung	31825

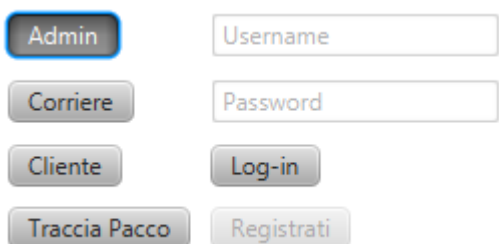
## 5.0 – Come utilizzare l'applicazione MyDelivery

L'applicazione MyDelivery è molto semplice da utilizzare. La prima interfaccia che l'utente vedrà una volta aperta l'applicazione sarà la seguente:



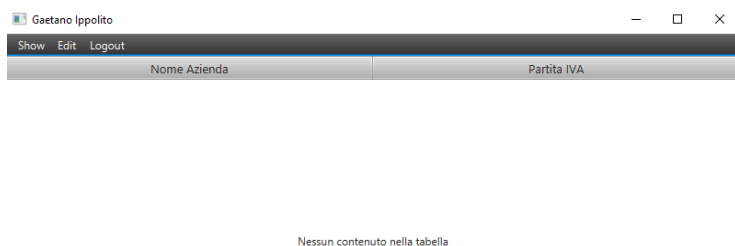
The login form consists of two columns. The left column contains four buttons: 'Admin' (highlighted with a blue border), 'Corriere', 'Cliente', and 'Traccia Pacco'. The right column contains two input fields: 'Username' (above the 'Admin' button) and 'Password' (above the 'Corriere' button). Below the 'Password' field are two buttons: 'Log-in' and 'Registrati'.

qui l'utente dovrà scegliere con chi effettuare il login.



The login form is identical to the previous one, but the 'Admin' button is now highlighted with a blue border. The 'Log-in' and 'Registrati' buttons are now disabled (grayed out).

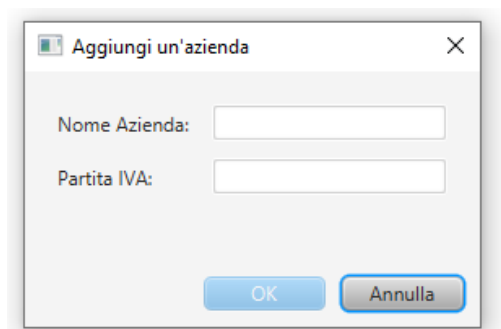
Una volta scelto il tipo di utente, i tasti "Log-in" e "Registrati" si attiveranno e potranno essere cliccati. Essendo l'Admin unico, sarà l'unico che non potrà Registrarsi, per cui effettuare un login con il profilo "Username" come Username e "1234" come password.



Effettuato il login la prima schermata che verrà visualizzata è la seguente.

È molto importante come prima operazione aggiungere aziende con cui si collabora.

Cliccare sul tasto "Edit" e puntare il cursore su "Azienda", per poi cliccare su "Aggiungi Azienda".



The dialog box is titled 'Aggiungi un'azienda' and has a close button (X) in the top right corner. It contains two input fields: 'Nome Azienda:' and 'Partita IVA:'. At the bottom, there are two buttons: 'OK' and 'Annulla'.

Cliccare su "Aggiungi Azienda" farà apparire questa finestra in cui sarà possibile aggiungere un'azienda all'interno della nostra applicazione inserendo i suoi dati, in particolare il nome e la sua partita iva.

Ipotizziamo di scrivere “Samsung” come nome dell’azienda e “78345” come partita iva. Una volta cliccato sul tasto OK, il tutto sarà visualizzato nella tabella:

Show Edit Logout	
Nome Azienda	Partita IVA
Samsung	78345

Inoltre, una volta creata un’azienda verranno creati dei Veicoli associati all’azienda appena creata (questi serviranno per l’algoritmo).

Show Edit Logout				
Tipo Veicolo	Capienza Veicolo	Codice	Azienda Associata	Non Disponibile
AUTO	50.0	0	Samsung	false
FURGONE	100.0	1	Samsung	false
CAMION	150.0	2	Samsung	false
CAMION	150.0	3	Samsung	false
CAMION	150.0	4	Samsung	false

La colonna “Non disponibile” con “false” indica che il veicolo in questione è disponibile per contenere pacchi.

Creare l’azienda è il primo passaggio da svolgere. Quello successivo è registrarsi come Cliente per poter iniziare a generare un ordine:

Admin

Username

Corriere

Password

Cliente

Log-in

Traccia Pacco

Registrati

Cliccato sul bottone “Cliente” si potrà premere sul bottone “Registrati”. Premendo su quel bottone, sarà possibile visualizzare la seguente finestra, in cui è possibile registrare un account “Cliente”.

Registra un Account

×

Registra un account

Nome

Cognome

Username

Password

Email

Indirizzo

Codice fiscale

Numero di Telefono

OK

Annulla



Creato un Cliente e inseriti i dati per accedere, se il login va a buon fine, ciò che verrà visualizzata è la seguente schermata:

The screenshot shows a web application window titled "Gaetano Ippolito". It has a navigation bar with two buttons: "Crea Nuovo Ordine" (highlighted with a blue border) and "Visualizza Ordini". A "Logout" button is in the top right corner. Below the navigation bar is a table with the following headers: "Mittente", "Destinatario", "Stato Ordine", "In arrivo il..", "Corriere", "Azienda", and "Codice Pacco". The table body is empty, and a message "Nessun contenuto nella tabella" is displayed in the center.

Qui il cliente potrà visualizzare gli ordini da lui effettuati e creare nuovi ordini presso azienda che collaborano con la nostra applicazione.

Premendo sul tasto "Visualizza Ordini" in questo momento non mostrerà niente. Per visualizzare un ordine bisogna crearlo cliccando su "Crea Nuovo Ordine":

The screenshot shows a modal window titled "Crea un ordine". It contains a dropdown menu labeled "Ordine presso azienda..." with the selected option "Samsung - Partita IVA: 78345". Below the dropdown are four input fields: "Cognome", "Indirizzo", "Codice fiscale", and "Numero di Telefono". At the bottom of the modal are two buttons: "OK" and "Annulla".

Le aziende che verranno visualizzate saranno solo quelle che sono state create da all'interno di MyDelivery. Ciò che bisogna fare in questa schermata è creare un ordine con le informazioni del destinatario dell'ordine.

Compilato il tutto e una volta premuto sul tasto OK verrà visualizzato l'ordine:

Mittente	Destinatario	Stato Ordine	In arrivo il..	Corriere	Azienda	Codice Pacco
Gaetano Ippolito	Marco P	IN_PREPARAZIONE	2021-01-22	null	Samsung	42623

Come si può notare, nell'informazione del corriere abbiamo la scritta "null". Questo accade siccome non abbiamo registrato un Corriere e non abbiamo ancora utilizzato l'algoritmo NextFit.

Come terzo passaggio, procedere alla creazione del Corriere:

The main application window contains a sidebar with buttons: Admin, Corriere (highlighted), Cliente, Traccia Pacco, Log-in, and Registrati. The main area has input fields for Nome, Cognome, and ID.

A dialog box titled "Registra un corriere" is open. It contains input fields for Nome, Cognome, and ID. There is a dropdown menu labeled "Seleziona l'azienda presso cui lavora" with "Samsung - Partita IVA: 78345" selected. At the bottom are "OK" and "Annulla" buttons.

In questa finestra di dialogo sarà possibile far Registrare un corriere. Da notare che si può utilizzare l'azienda presso cui lavora e che può effettuare il login tramite "Nome", "Cognome" e "ID".

Una volta compilati è possibile effettuare l'accesso con il Corriere.

The main application window shows the "Visualizza Ordini" button highlighted. Below it is a table with columns: Codice ordine, Destinazione, Scadenza, Veicolo, Peso Container, and Stato Ordine. The table is currently empty.

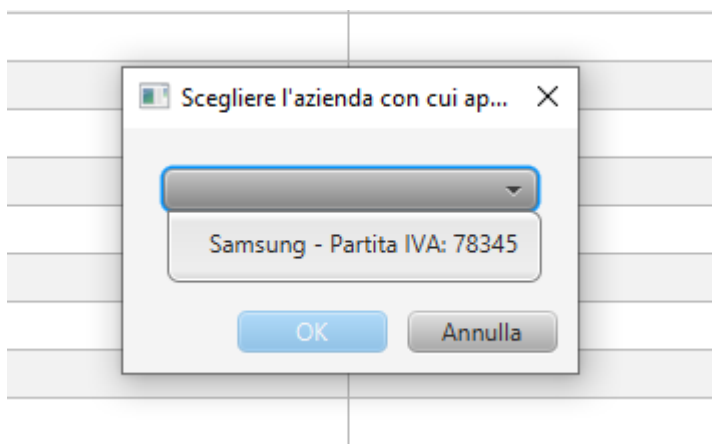
Nessun contenuto nella tabella

Questa è ciò che viene mostrato quando si logga con un Corriere. Al momento non verrà visualizzato nulla se si preme sul tasto "Visualizza Ordini". Mentre se si prova a cambiare lo stato dell'ordine non accadrà nulla. Questo accade siccome l'ordine creato dal cliente "Gaetano" ancora deve essere preso in carico e quindi ancora deve essere associato al corriere "Paolo".

Come prossimo passaggio bisogna avviare l'algoritmo NextFit. Quindi bisogna ritornare alla schermata dell'Admin. I pacchi che verranno generati dagli ordini potranno essere controllati tramite l'interfaccia Admin, infatti (ipotizzando di aver creato due ordini):

Gaetano Ippolito		— □ ×	
Show Edit Logout			
Codice Pacco		Peso Pacco	
42623		0.1	
33176		13.1	

Una volta che i pacchi sono stati creati e si hanno abbastanza corrieri è finalmente possibile utilizzare l'algoritmo:



Cliccare sul bottone "Edit" e premere su "NextFit". Questo farà apparire la seguente schermata.

Qui bisogna scegliere l'azienda presso cui generare l'algoritmo nextFit.

L'algoritmo funzionerà se saranno disponibili abbastanza Corrieri – Veicoli ed esisteranno Ordini presso quell'azienda. Cliccando sul

bottone OK, si potranno visualizzare i cambiamenti che ha apportato l'algoritmo:

Show Edit Logout				
Tipo Veicolo	Capienza Veicolo	Codice	Azienda Associata	Non Disponibile
FURGONE	100.0	1	Samsung	false
CAMION	150.0	2	Samsung	false
CAMION	150.0	3	Samsung	false
CAMION	150.0	4	Samsung	false
AUTO	50.0	0	Samsung	true

Il primo Veicolo ha abbastanza capienza per contenere entrambi i pacchi creati dal cliente "Gaetano". Da notare che non è più disponibile, per cui, per i prossimi ordini, non potrà essere utilizzato per essere riempito.

Crea Nuovo Ordine		Visualizza Ordini				Logout
Mittente	Destinatario	Stato Ordine	In arrivo il..	Corriere	Azienda	Codice Pacco
Gaetano Ippolito	Marco P	IN_PREPARAZIONE	2021-01-22	Paolo D	Samsung	42623
Gaetano Ippolito	Daniele D	IN_PREPARAZIONE	2021-02-03	Paolo D	Samsung	33176

Ora il cliente avrà le seguenti informazioni. Il Corriere non è più null, ma verrà visualizzato il corriere del suo ordine, ovvero colui che abbiamo creato precedentemente che lavora presso l'azienda Samsung.

Paolo D

Visualizza Ordini

Cambia stato ordine

Logout

Codice ordine	Destinazione	Scadenza	Veicolo	Peso Container	Stato Ordine
42623	Via Q, 4	2021-01-22	0: AUTO	13.2	IN_PREPARAZIONE: Deposito
33176	Via O, 3	2021-02-03	0: AUTO	13.2	IN_PREPARAZIONE: Deposito

Il corriere ora potrà visualizzare gli ordini a lui associati. Nella sezione "Veicolo" verrà visualizzato il Veicolo che dovrà guidare, mentre "Peso Container" indica di quanto è stato riempito il suo Veicolo. Ora sarà possibile cambiare lo stato degli ordini:

2021-02-03	0: AUTO	13.2
------------	---------	------

Cambia stato dell'ordine

Centro di smistamento

1 - Napoli

1 - Napoli
2 - Ancona
3 - Milano
4 - Roma
5 - Messina
6 - Palermo

In questa schermata bisogna scegliere il centro di smistamento che il corriere ha raggiunto.

Una volta scelto il centro di smistamento, l'ordine cambierà il suo stato:

Paolo D

Visualizza Ordini

Cambia stato ordine

Logout

Codice ordine	Destinazione	Scadenza	Veicolo	Peso Container	Stato Ordine
42623	Via Q, 4	2021-01-22	0: AUTO	13.2	IN_TRANSITO: Milano
33176	Via O, 3	2021-02-03	0: AUTO	13.2	IN_TRANSITO: Milano

Gaetano Ippolito

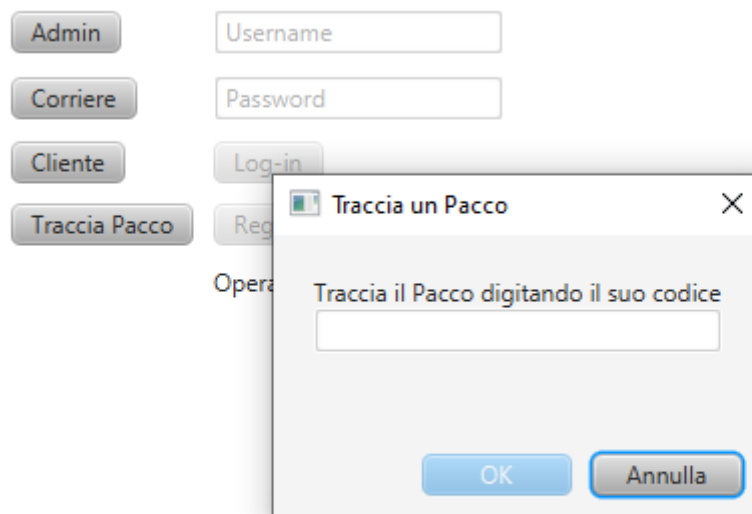
Crea Nuovo Ordine

Visualizza Ordini

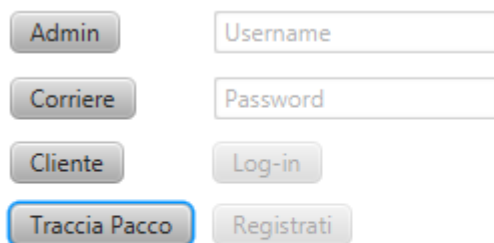
Logout

Mittente	Destinatario	Stato Ordine	In arrivo il..	Corriere	Azienda	Codice Pacco
Gaetano Ippolito	Marco P	IN_TRANSITO	2021-01-22	Paolo D	Samsung	42623
Gaetano Ippolito	Daniele D	IN_TRANSITO	2021-02-03	Paolo D	Samsung	33176

I pacchi potranno essere tracciati grazie al bottone “Traccia Pacco”:



Inserito il codice giusto si potranno visualizzare le informazioni dell’ordine. Ipotizziamo di inserire il codice del primo pacco “42623”:



```
Mittente: Gaetano Ippolito
Destinatario: Marco P
Data di consegna: 2021-01-22
Stato pacco: IN_TRANSITO
Ultima posizione: Milano
Codice pacco: 42623
Fragile: true
```

In fondo a destra appariranno tutte le informazioni che sull’ordine.