

# Progetto Tesi

## Process Manager

**A cura di:**

Bonetti Mattia (10910831),  
Morelli Alessio (10828391)

**Corso:** Ingegneria del Software

**Professore:** Salnitri Mattia

**Consegna:** 11 settembre 2025

<b>Traccia:</b>	<b>3</b>
<b>Valutazione dei Progetti:</b>	<b>5</b>
<b>1. Prima Sezione:</b>	<b>6</b>
1.1 CONTESTO:	6
1.2 ABSTRACT:	6
1.3 REQUISITI:	7
1.4 RUOLI e PERMESSI:	9
<b>2. Seconda Sezione:</b>	<b>10</b>
2.1 TECNOLOGIE USATE:	10
2.2 PATTERN MVC e LAYERED ARCHITECTURE:	13
2.3 DESIGN PATTERN:	14
2.4 USE CASE DIAGRAM:	17
2.5 PACKAGE DIAGRAM:	20
2.6 CLASS DIAGRAM BACKEND:	21
2.7 CLASS DIAGRAMS FRONTEND:	28
2.8 OTHER DIAGRAMS:	33
2.9 MULTITHREADING:	35
<b>3. Terza Sezione:</b>	<b>37</b>
3.1 PIANO DI TEST:	37
3.2 RAPPORTO COVERAGE:	42
3.3 RISULTATI TEST:	43
<b>4. Conclusioni:</b>	<b>47</b>
<b>Collegamenti:</b>	<b>48</b>

# Traccia:

La documentazione dovrà descrivere:

1. I requisiti per il sistema da sviluppare
2. Il progetto del sistema (comprendendo diagrammi UML)
3. Il piano di test per il sistema (e.g., test di unità, test di integrazione)
4. Il rapporto sulla copertura dei test

Nella stesura della documentazione ci si aspetta che facciate riferimento a quanto imparerete nelle parti del corso che riguardano gli aspetti associati alle parti richieste. Per ottenere una buona valutazione è necessario (ma non sufficiente) produrre **documentazione di buona qualità**, considerando tutti gli aspetti sopra elencati. La qualità della documentazione non è necessariamente legata alla sua lunghezza. In caso di domande/dubbi in corso d'opera, potete contattare il docente via mail.

La documentazione del progetto avrà una lunghezza massima in base al numero di componenti del gruppo:

- 1 persona: 40 pagine.
- 2 persone: 50 pagine.
- 3 persone: 60 pagine.

Il testo oltre il limite massimo di pagine verrà ignorato, yerrà messa una penalità al punteggio finale se il limite viene superato.

La documentazione del progetto dovrà contenere:

- Informazioni della persona o del gruppo che propone il progetto
- Un link all'archivio (versioning system oppure cartella condivisa) contenente il software. È opportuno che questo archivio sia esterno a WeBeep visto che c'è un limite nella quantità di 250MB che potete caricare nel sistema.
- Descrizione dell'ambito:
  - Definizione dei requisiti: una descrizione dettagliata dei requisiti del vostro progetto. Ad esempio, il contesto dove opererà il progetto, quali sono gli obiettivi del progetto.
  - Descrizione delle funzionalità divise per utente, con nome del requisito ed una brevissima descrizione (un paio di righe al massimo)

- Descrizione della progettazione:
  - use case diagram con descrizione degli attori e funzionalità.
  - class diagram con descrizione dei packages.
  - class diagram con descrizione delle classi principali, per ogni package.
  - class diagram con descrizione pattern utilizzati.
  - sequence/activity/state diagram con alcuni casi rilevanti (opzionale).
- Descrizione dei test cases, siano essi fatti con Junit(o libreria similare), postman, o fatti manualmente per testare GUI.
  - Per il progetto sono obbligatori i test di unita' (unit test), altri tipi di test, come quelli di integrazione (integration tests) sono opzionali

I Diagrammi devono essere tutti discussi nella documentazione. Un diagramma (quindi un'immagine) senza discussione non verrà considerato.

# **Valutazione dei Progetti:**

La valutazione di ogni progetto durerà 30 minuti, e sarà organizzata nel seguente modo:

**Presentazione progetto** (max 15 minuti): Lo studente (o studenti in caso di sviluppo di gruppo) dovranno brevemente descrivere obiettivi, funzionalità, e architettura dell'applicazione realizzata. A tal fine, possono utilizzare la documentazione già consegnata o, se preferiscono, delle slide. Nel caso di gruppi composti da più persone, tutti i membri del gruppo dovranno intervenire nella presentazione.

**Demo progetto** (max 5 minuti): i membri del gruppo dovranno mostrare il corretto funzionamento dell'applicazione mediante una demo della medesima. La demo dovrà essere effettuata live condividendo lo schermo del proprio computer. Qualora l'applicazione comprendesse un app per smartphone, è consentito utilizzare un emulatore su computer, oppure è possibile condividere lo schermo del proprio smartphone (p.e., con TeamViewer).

**Domande** (max 10 minuti): il docente vi porrà alcune domande sulla progettazione, sull'implementazione e sul testing del progetto. I membri del gruppo dovranno quindi avere a portata di mano il codice sorgente del progetto, ed essere pronti a giustificare le proprie scelte progettuali.

Per la discussione progetti, verrà utilizzata l'aula Webex del corso. Affinché possiate partecipare alla discussione dei progetti, sarà necessario utilizzare l'applicazione Cisco Webex Meetings (e non l'interfaccia web) ed autenticarvi in modalità Single-Sign-On utilizzando le vostre credenziali di ateneo (codice persona). \*NON\* è invece sufficiente inserire semplicemente nome e cognome, in quanto il docente non avrebbe alcuna garanzia sull'identità del partecipante.

I membri di ogni gruppo dovranno inoltre collegarsi 30 minuti prima del proprio turno, e rimanere collegati per i 30 minuti successivi alla discussione del loro progetto, così da garantire la presenza di testimoni. **È altresì permesso agli studenti che non presenteranno il proprio progetto in questa sessione di assistervi.**

# 1. Prima Sezione:

## 1.1 CONTESTO:

---

L'azienda oggetto del progetto è una realtà di piccole/medie dimensioni situata a Cassina de Pecchi, operante nel settore della metalmeccanica, con competenze che spaziano dalla tornitura a freddo, all'imbutitura e alla tranceria.

La gestione della produzione avviene ad oggi in maniera informale: il know-how relativo ai modelli, alle lavorazioni e ai processi è detenuto principalmente dal titolare e da suo figlio, mentre il know-how relativo agli ordini e ai clienti è detenuto dalla moglie del titolare e da un'unica collega.

Tali informazioni non sono documentate in modo strutturato, ma trasmesse oralmente o annotato su lavagnette e fogli sparsi.

Questa modalità operativa comporta alcune criticità, tra cui:

- Rischio di perdita di informazioni in caso di assenza delle figure chiave.
- Difficoltà nella condivisione delle informazioni coi dipendenti.
- Gestione manuale e poco tracciabile di ordini, clienti, macchinari e lavorazioni.

A tutti gli effetti, senza il titolare o il figlio presenti, la lavorazione deve necessariamente interrompersi. Negli anni, con l'espandersi dell'azienda, la situazione è diventata insostenibile.

Il progetto nasce quindi dall'esigenza di digitalizzare e centralizzare, seppur in parte, la gestione delle lavorazioni, fornendo un'applicazione che possa diventare il punto di riferimento per la produzione e l'organizzazione interna.

## 1.2 ABSTRACT:

---

L'obiettivo dell'applicativo è quello di offrire uno strumento in grado di gestire in modo integrato i principali aspetti dell'attività produttiva e organizzativa.

In particolare, deve consentire:

- Gestione degli ordini, con relativi modelli e processi produttivi.
- Pianificazione e monitoraggio delle lavorazioni.
- Organizzazione delle risorse aziendali, quali macchinari, materie prime e personale.
- Gestione delle informazioni relative ai clienti.
- Pubblicazione di annunci e comunicazioni interne.

## **1.3 REQUISITI:**

---

### **Requisiti funzionali:**

L'applicazione deve offrire funzionalità CRUD (Create, Read, Update, Delete) per i seguenti elementi:

- Annunci (Notices): creazione e gestione di comunicazioni interne rivolte al personale.
- Ordini (Orders): inserimento di nuovi ordini, tracciamento dello stato di avanzamento, completamento e cancellazione.
- Modelli (Models): archiviazione dei modelli con relativi processi di lavorazione.
- Processi (Processes): gestione dei passaggi produttivi associati a ciascun modello.
- Macchinari (Machinery): catalogazione e monitoraggio delle risorse tecniche utilizzate nelle lavorazioni.
- Materie prime (Raw): gestione delle giacenze a magazzino e disponibilità per la produzione. Rientrano in questa categoria anche che i vari strumenti o stampi utilizzati.
- Clienti (Clients): anagrafica dei clienti con informazioni di contatto e storico ordini.

Inoltre, sono richieste funzionalità legate agli utenti e all'autenticazione:

- Registrazione e accesso protetto al sistema tramite email e password.
- Possibilità di consultare, modificare ed eventualmente cancellare i propri dati personali, quali nome, cognome e numero di telefono.

### **Requisiti di sicurezza:**

Essendoci informazioni protette, come i dati sui clienti, sono stati imposti dei minimi requisiti di sicurezza:

- L'accesso al sistema deve avvenire solo tramite autenticazione con credenziali (email e password).
- A seguito dell'autenticazione, viene generato un token JWT (JSON Web Tokens), necessario per ogni successiva richiesta.
- Gli endpoint backend devono essere protetti, in modo da poter interagire col database solo tramite le funzionalità dell'applicazione.
- Le password vengono gestite in modo sicuro, tramite hashing.

### **Requisiti di usabilità:**

Un aspetto fondamentale del progetto riguarda l'usabilità dell'applicazione, dal momento che gli utenti previsti appartengono ad una fascia d'età molto ampia (16-65 anni) e possono avere competenze informatiche eterogenee, fino ad includere persone totalmente inesperte.

Per questo motivo, l'interfaccia è stata progettata con l'obiettivo di essere il più semplice e intuitiva possibile, appiattendo al massimo la curva di apprendimento.

La maggior parte delle sezioni ha un formato tabellare: rappresentazione familiare e universalmente riconoscibile, che permette di visualizzare in modo ordinato e immediato i dati relativi ad annunci, ordini, clienti, macchinari e materie prime.

Sono previste solo due eccezioni, motivate dalla natura delle informazioni da gestire.

Nella sezione dei processi si è scelto di utilizzare un approccio WYSIWYG (what-you-see-is-what-you-get), rappresentando graficamente i vari step come delle box e le varie funzionalità tramite icone.

Nella sezione delle lavorazioni, la rappresentazione grafica tramite diagrammi di Gantt è essenziale per evidenziare la pianificazione temporale delle attività. L'interfaccia consente comunque interazioni semplici e familiari, come l'uso del tasto destro per accedere all'unico menu contestuale, oppure resize e drag-and-drop dei blocchi per la modifica delle durate e delle posizioni delle lavorazioni.

Queste scelte mirano a garantire che l'applicazione possa essere implementata e sfruttata a pieno immediatamente, senza stravolgere eccessivamente il normale iter lavorativo.

### **Requisiti di portabilità / compatibilità:**

L'applicazione è stata sviluppata con l'obiettivo principale di essere utilizzata su computer con sistema operativo Windows nelle versioni recenti, in quanto rappresenta la piattaforma maggiormente diffusa all'interno dell'azienda e quindi il contesto reale di utilizzo.

La scelta di adottare il linguaggio Java garantisce tuttavia un buon grado di portabilità intrinseca, aprendo la possibilità di estendere il sistema anche ad altre piattaforme in futuro, senza dover riscrivere da zero l'intera applicazione. In particolare, il progetto potrebbe essere adattato a dispositivi mobili, qualora emergesse la necessità di consentire l'accesso a determinate funzionalità anche da smartphone e tablet. Questa prospettiva, pur non essendo un requisito richiesto e quindi non sviluppato, rappresenta un'opportunità evolutiva resa possibile dalla tecnologia scelta.

## 1.4 RUOLI e PERMESSI:

Gli utenti del sistema sono suddivisi in tre categorie, ognuna caratterizzata da diversi livelli di accesso e responsabilità:

- **Manager:** ruolo con la totalità dei privilegi, responsabile della gestione globale di ogni aspetto.
- **Employee:** ruolo operativo, con accesso limitato alle funzionalità necessarie per svolgere le attività produttive.
- **Accountant:** ruolo focalizzato sulla gestione amministrativa e contabile, con possibilità di accedere principalmente ai dati economici e ai clienti.

Per esplorare i permessi, utilizziamo la notazione CRUD (Create, Read, Update, Delete) che indica rispettivamente la possibilità di creare, leggere, modificare ed eliminare elementi nelle varie sezioni.

Sections	Manager				Employee				Accountant			
	C	R	U	D	C	R	U	D	C	R	U	D
Notice Board	✓	✓	✗	✓	✓	✓	✗	✗	✓	✓	✗	✗
Profile	✗	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓
Employee List	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
Gantt	✗	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗
Clients	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	✓
Orders	✓	✓	✓	✓	✗	✓	✗	✗	✓	✓	✓	✓
Models	✓	✓	✓	✓	✗	✓	✗	✗	✗	✓	✗	✗
Processes	✓	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗
Machinery	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
Inventory	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓	✓

La sezione relativa ai diagrammi di Gantt presenta delle regole aggiuntive:

- I Manager sono gli unici che possono apportare modifiche effettive alla pianificazione, spostando e ridimensionando i blocchi, e assegnando i dipendenti alle lavorazioni.
- Gli Employee, pur potendo fisicamente interagire coi blocchi, non possono confermare tali modifiche. La loro unica azione specifica è la possibilità di segnalare un ritardo.

## 2. Seconda Sezione:

### 2.1 TECNOLOGIE USATE:

---

Il progetto è stato sviluppato con architettura a due livelli, composta da un server (backend) e un client (frontend).

#### Backend – Spring Framework:

La parte server dell'applicazione è stata realizzata utilizzando Spring Boot, un framework che semplifica lo sviluppo di applicazioni Java. Le principali tecnologie e componenti utilizzate sono:

- Spring MVC and API REST.
- Spring Data JPA with Hibernate.
- Spring Security.

La componente Spring MVC è stata utilizzata per realizzare un set di API REST, ossia una serie di interfacce che consentono ai client di comunicare col server tramite richieste HTTP standard (GET, POST, PUT, DELETE). L'iter è il seguente:

1. Il client invia richieste HTTP contenenti eventualmente dati in formato JSON.
2. Il server riceve le richieste e le instrada verso i metodi appropriati dei controller, grazie al sistema di mapping fornito da Spring (@GetMapping, @PostMapping, ecc.).
3. Il server elabora la richiesta, eventualmente interagisce col database e restituisce una risposta, anch'essa in formato JSON.

Questo approccio rende il sistema modulare e soprattutto indipendente dalla tecnologia del client: in futuro, un altro client, come un app mobile, potrebbe interagire con le stesse API senza modifiche sostanziali.

Per la gestione dei dati si è scelto di utilizzare JPA (Java Persistence API) insieme a Hibernate, che ne rappresenta l'implementazione più diffusa e sviluppata.

JPA è uno standard che definisce un insieme di annotazioni e interfacce per mappare le entità Java su un database relazionale.

Hibernate è un motore ORM (Object-Relational Mapping) che traduce automaticamente le operazioni sugli oggetti Java in query SQL verso il database.

In pratica, anziché scrivere manualmente query SQL, abbiamo definito delle classi Java annotate (usando `@Entity`, `@Table`, `@Column`, ecc.) che rappresentano le tabelle del database, e dei metodi che rappresentano le operazioni che vogliamo eseguire (come `save`, `findById`, `delete`, ecc.). Hibernate si occupa di generare le query SQL corrispondenti alle operazioni, gestire le relazioni tra le entità e in generale mantenere la consistenza dei dati usando transazioni.

Fanno eccezione solo query particolarmente complesse, che vengono manualmente espresse tramite l'annotazione `@Query`.

Per la parte di autenticazione e autorizzazione è stato utilizzato Spring Security. L'iter è il seguente:

1. L'utente inserisce le credenziali, in questo caso email e password.
2. Il backend verifica i dati confrontando la password inserita (dopo hashing) con quella memorizzata nel database.
3. Se le credenziali sono valide, il sistema genera un token di sessione, in questo caso JWT.
4. Ad ogni richiesta successiva il client deve includere il token nell'header (`Authorization: Bearer "token"`).
5. Spring riceve la richiesta e ne valida il token.
6. Oltre al token, è definito anche un ruolo, che indica se l'utente può o meno accedere all'endpoint richiesto.

Questo approccio garantisce che il database, e quindi l'interezza dei dati, sia accessibile solo tramite l'applicazione, in quanto unico mezzo che genera e assegna token e ruoli.

L'utilizzo di un token permette inoltre di mantenere un paradigma stateless, in cui ogni richiesta è indipendente dalle altre; non viene mantenuta alcuna sessione aperta lato server e ogni richiesta deve contenere tutte le informazioni necessarie per essere autenticata e autorizzata.

Questo approccio rende il server più sicuro, scalabile e flessibile.

Come database è stato scelto PostgreSQL, un DBMS relazionale open source particolarmente affidabile e comodo grazie alla GUI offerta con pgAdmin.

Postgre, oltre ad essere estremamente performante, offre una serie di tipi proprietari (come il tipo `Text` che indica una stringa di dimensione potenzialmente infinita e non definita a priori) e supporta oggetti di grandi dimensioni (LOB – Large Object), garantendo a noi sviluppatori la massima flessibilità.

## Frontend – JavaFX

La parte client è stata sviluppata interamente con JavaFX, la libreria ufficiale di Java per la realizzazione di interfacce grafiche moderne.

JavaFX consente di costruire interfacce interattive e responsive, basate su una gerarchia di nodi (Scene Graph).

I principali componenti usati nel progetto sono:

- Strutture tabellari (`TableView`): usate per la loro semplicità e familiarità.
- Layout flessibili (`GridPane`, `HBox`, `VBox`): container che permettono di adattarne le dimensioni e il contenuto dinamicamente.
- Binding dati: grazie all'annotazione `@FXML` e all'`fx:id` consente di collegare direttamente i componenti nella view ai controller.
- Integrazione col backend: la comunicazione col server avviene tramite `HttpURLConnection`, che consente di mantenere il paradigma REST.

Uno degli elementi più innovativi del progetto, e motivo per cui abbiamo preferito JavaFX a Swing, React o altri framework, è stato l'impiego della libreria FlexGanttFX.

FlexGanttFX è una libreria commerciale sviluppata specificatamente per JavaFX, pensata per la visualizzazione e la gestione di attività pianificate nel tempo tramite la generazione di diagrammi di Gantt. La sua integrazione ha permesso di fornire agli utenti uno strumento visivo immediato e potente per la gestione della produzione.

Un ringraziamento particolare va a *Dirk Lemmermann*, sviluppatore principale di FlexGanttFX, che ha gentilmente fornito una licenza gratuita e a tempo indeterminato per l'utilizzo della libreria all'interno del nostro progetto.

## Testing

Infine, per la parte di testing, sono stati adottati tre strumenti: JUnit, Mockito e TestFX.

JUnit è il framework di riferimento per il testing in Java, compreso nel pacchetto di Spring. È stato utilizzato per scrivere ed eseguire test unitari, verificando il corretto funzionamento dei singoli metodi e delle singole classi.

Mockito è il framework per il mocking (quindi la creazione di oggetti finti) integrato con JUnit. Viene impiegato per isolare i componenti da testare simulandone le loro dipendenze, come repository o servizi, in modo da verificare esclusivamente la logica del componente sotto esame.

TestFX è utilizzato per il testing del frontend in JavaFX, simulando una componente grafica alla volta per verificarne l'esperienza utente e la coerenza della GUI.

## **2.2 PATTERN MVC e LAYERED ARCHITECTURE:**

Una delle specifiche della tesi prevedeva l'adozione di un pattern architettonale, come MVC, MVVM o altre varianti. Abbiamo scelto di adottare la versione nativa di Spring, nota come Layered Architecture.

Tale architettura suddivide l'applicazione in 5 layer ben distinti, ciascuno con una responsabilità specifica, favorendo la separazione delle competenze e la scalabilità del codice.

- >  controller
- >  dto
- >  model
- >  repository
- >  security
- >  service
- >  view

Nel server, i controller si occupano esclusivamente delle richieste HTTP, esponendo gli endpoint REST e richiamando i services. Nel client invece, i controller si occupano della gestione della view e richiamano i services.

Il model include le entità del dominio applicativo, che rappresentano i dati del sistema. Rientrano in questo package anche le entità, che Hibernate utilizza per popolare il database.

Il package view è presente solo nel client, in quanto riguarda la parte del frontend. Al suo interno sono presenti le view statiche di JavaFX.

Nel server, il service contiene la logica di business, ovvero le regole e i processi applicativi che elaborano i dati. Nel client invece, i service si occupano della comunicazione tra client e server.

Il package repository è presente solo nel server, in quanto necessario a Hibernate per comunicare col database.

A questi package principali si sono successivamente aggiunti due ulteriori moduli: security e dto.

Il primo è dedicato interamente alla configurazione e alla gestione della sicurezza, necessario per implementare Spring Security.

Il secondo contiene le classi di tipo Data Transfer Object, usate per implementare l'omonimo design pattern.

## 2.3 DESIGN PATTERN:

### DTO – Data Transfer Object:

Il pattern DTO è stato ampiamente utilizzato per separare il modello interno del dominio applicativo dalle strutture inviate e ricevute tramite le API REST.

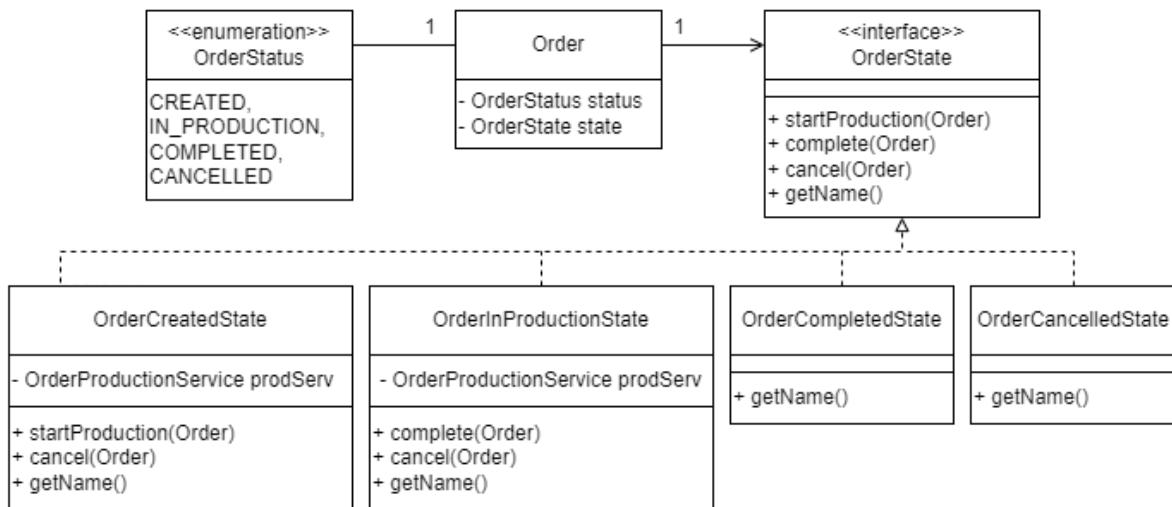
I DTO contengono solo le informazioni strettamente necessarie per la richiesta, evitando di esporre direttamente le entità JPA ed eventuali dati sensibili, come le password.

Tutti i DTO sono stati raccolti in un package dedicato.

### State Pattern:

Gli ordini possono trovarsi in diversi stati: Creato, In Produzione, Completato e Annullato. Per gestire i cambiamenti di stato è stato adottato uno state pattern; questo ha permesso di definire in modo chiaro quali operazioni fossero consentite in un determinato stato, evitando costrutti condizionali.

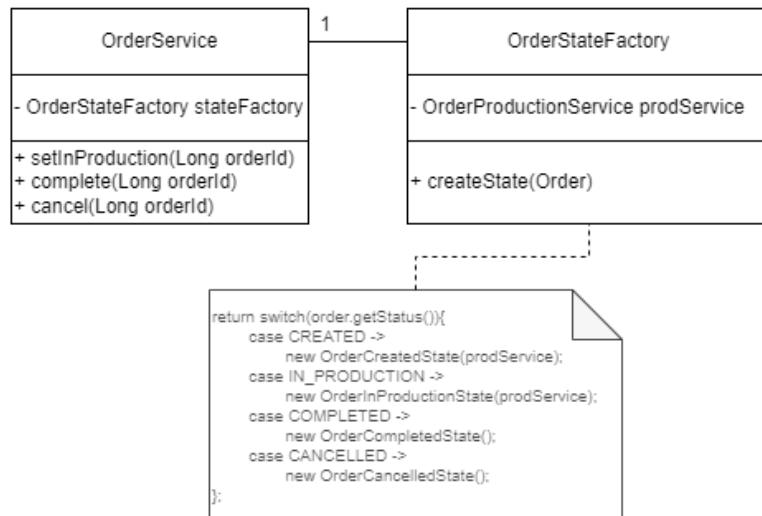
Ogni Order possiede un OrderStatus e un OrderState: il primo è usato solo per tenere traccia dello stato corrente a database tramite una stringa, mentre il secondo è usato per implementare effettivamente lo state pattern.



Nel caso una transizione non sia valida, viene sollevata un'eccezione. Tali transizioni, non essendo particolarmente significative, sono state omesse nel diagramma.

### Factory Method:

Per la gestione degli stati degli ordini è stato utilizzato una sorta di Factory Method. Tale pattern è stato essenziale per supportare il meccanismo di Dependency Injection di Spring, permettendo quindi l'iniezione automatica dei servizi necessari all'interno delle classi di stato.

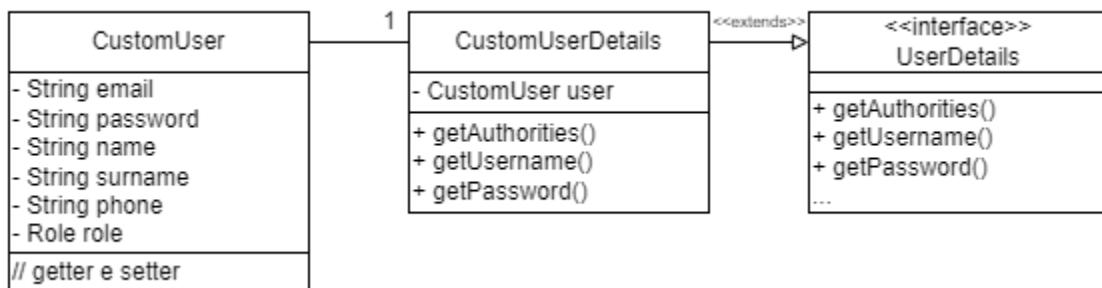


Senza questa soluzione, gli oggetti **OrderState** non avrebbero potuto accedere ai bean gestiti da Spring, come repository o services, e non avrebbero quindi potuto agire sul database.

### Adapter Pattern:

Nel progetto è stato necessario applicare un Adapter Pattern; la classe **CustomUserDetails** rappresenta l'adattatore tra l'entità utente del dominio **CustomUser** e l'interfaccia **UserDetails** richiesta da Spring Security.

In questo modo è stato possibile mantenere una rappresentazione e una modalità di accesso personalizzata.

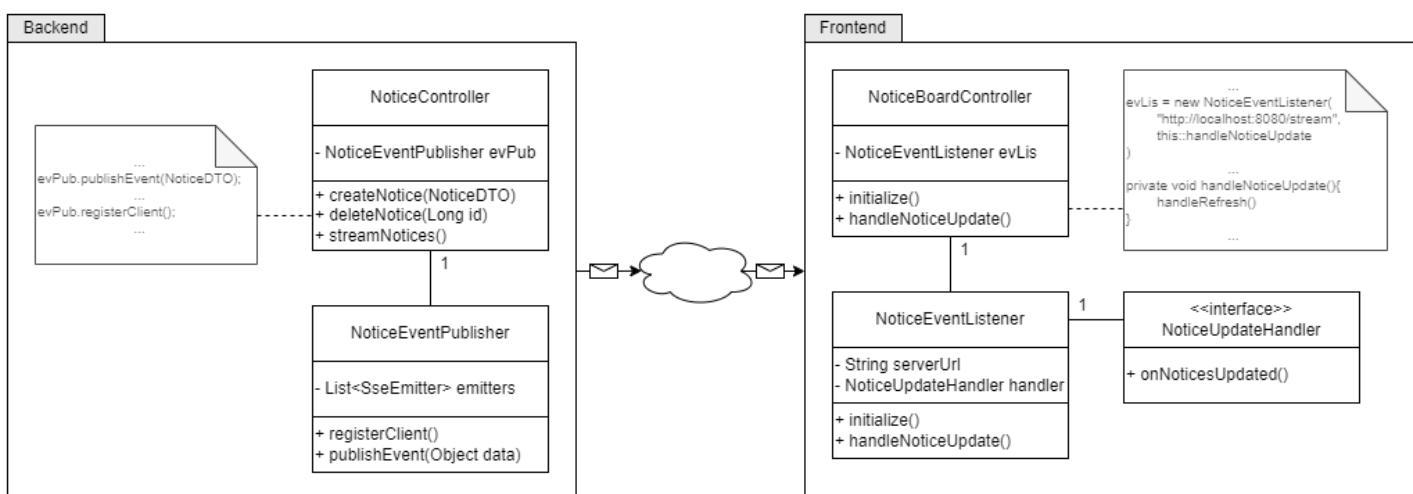


### Observer Pattern (distribuito):

Il pattern Observer è stato utilizzato per la gestione della Notice Board, cioè il sistema di annunci condiviso tra utenti. Anche in questo caso si tratta di una variante particolare poiché l'osservazione avviene tra backend e frontend.

Sul backend è stata utilizzata la logica di SSE (Server-Sent Events): il server mantiene un canale (stream) aperto e invia notifiche in tempo reale agli utenti connessi ogni volta un annuncio venga creato o eliminato.

Sul frontend è stato implementato un listener che si “sottoscrive” allo stream emesso dal backend e aggiorna dinamicamente l’interfaccia in base agli eventi che riceve.

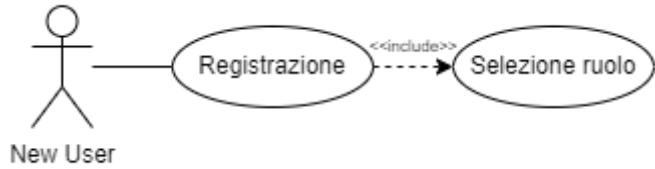


L’interfaccia `NoticeUpdateHandler` è stata aggiunta per permettere al controller stesso di definire il proprio handler.

## 2.4 USE CASE DIAGRAM:

Per questioni di chiarezza, semplicità e spazio, dividiamo gli use case per utente, in ordine di permessi.

Alla prima apertura dell'applicazione, l'utente non possiede ancora alcun ruolo o account, e l'unica funzionalità disponibile è quindi la registrazione.



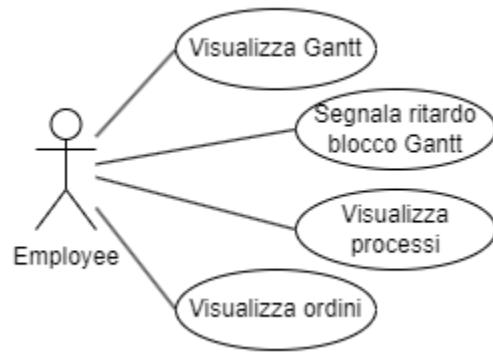
Alcune delle funzionalità sono comuni a tutti i ruoli. Nello specifico, ogni utente ha la possibilità di visualizzare la bacheca degli annunci, in quanto punto di incontro comune per tutta l'azienda. Allo stesso modo, ognuno ha il diritto di visualizzare il proprio account ed eventualmente modificarlo o eliminarlo.

Di seguito lo use case diagram relativo a qualsiasi utente che abbia un ruolo riconosciuto come valido (al momento, Manager, Employee o Accountant).



L'Employee è il ruolo col minor numero di permessi, in quanto può soltanto visualizzare lo stretto necessario per proseguire con la lavorazione, come la pagina dei Gantt o la pagina dei processi.

Di seguito, lo use case diagram coi permessi aggiuntivi degli Employee rispetto ad All Role.



L'Accountant è un ruolo prettamente amministrativo, che possiede però i primi permessi di modifica, in quanto deve poter gestire gli ordini e i clienti.



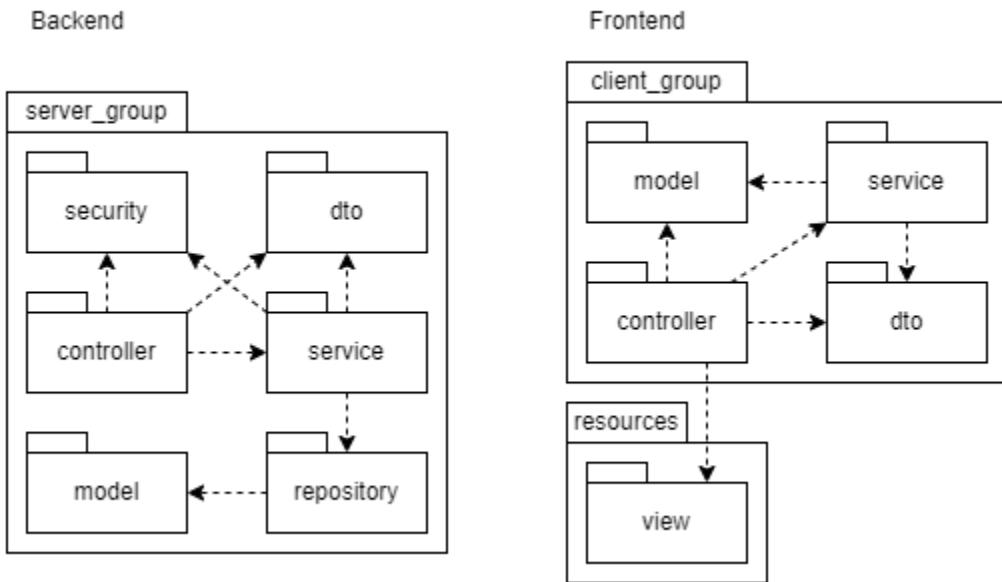
Il Manager infine, è il ruolo con la totalità dei permessi. Essendo la persona di riferimento dell'azienda, in questo caso il titolare, deve poter visualizzare e modificarne ogni aspetto, dalle lavorazioni agli ordini.

Di seguito, lo use case diagram coi permessi aggiuntivi dei Manager rispetto ad All Role.



## 2.5 PACKAGE DIAGRAM:

Il progetto è diviso in due macro-sezioni: backend e frontend. Entrambi rispettano la Layered Architecture discussa in precedenza, da cui derivano le dipendenze esemplificate dal seguente schema.



Nel frontend, il package `view` è in una cartella a parte in quanto contiene file XML e non Java. “Resources” è il nome di default assegnato da IntelliJ, nonché la cartella di riferimento per l’FXMLLoader di JavaFX.

Tutte le frecce indicano una dipendenza di tipo <<use>>.

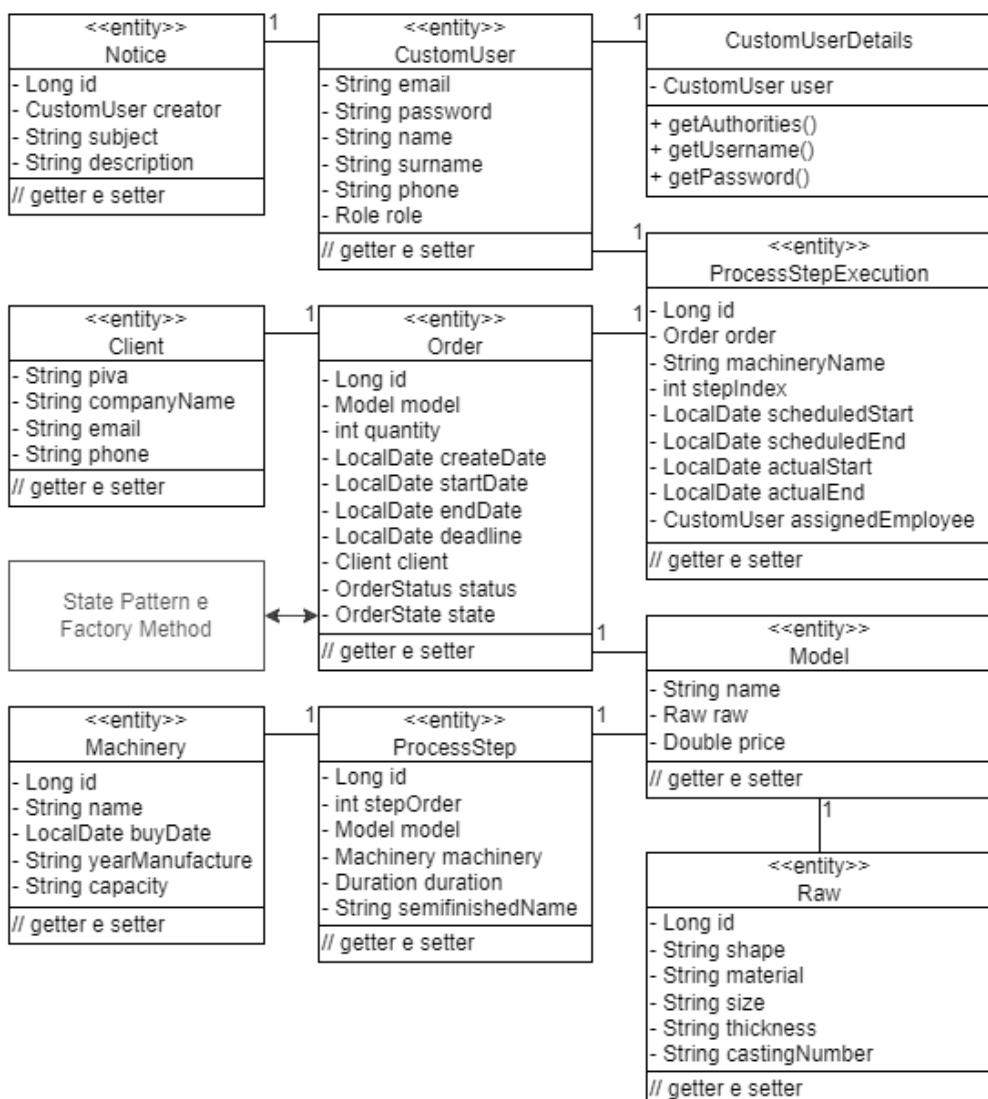
## 2.6 CLASS DIAGRAM BACKEND:

### Model:

Per via di Hibernate, la maggior parte delle classi in questo package corrispondono alle tabelle a database. Tali classi sono state indicate con un <<entity>> e non hanno nessuna logica applicativa, se non getter e setter per la loro creazione.

Fanno eccezione solo le classi riguardanti gli stati degli ordini e la classe `CustomUserDetails`, che sono però spiegate nel dettaglio nella sezione apposita, in quanto relative a dei design pattern.

`ProcessStep` e `ProcessStepExecution` potrebbero sembrare ripetuti ma rappresentano due concetti distinti: mentre i primi rappresentano lo step astratto di lavorazione, usato come template per definire una sequenza di lavorazioni, i secondi rappresentano le istanze concrete di esecuzione e sono generati nel momento in cui un ordine va in produzione.

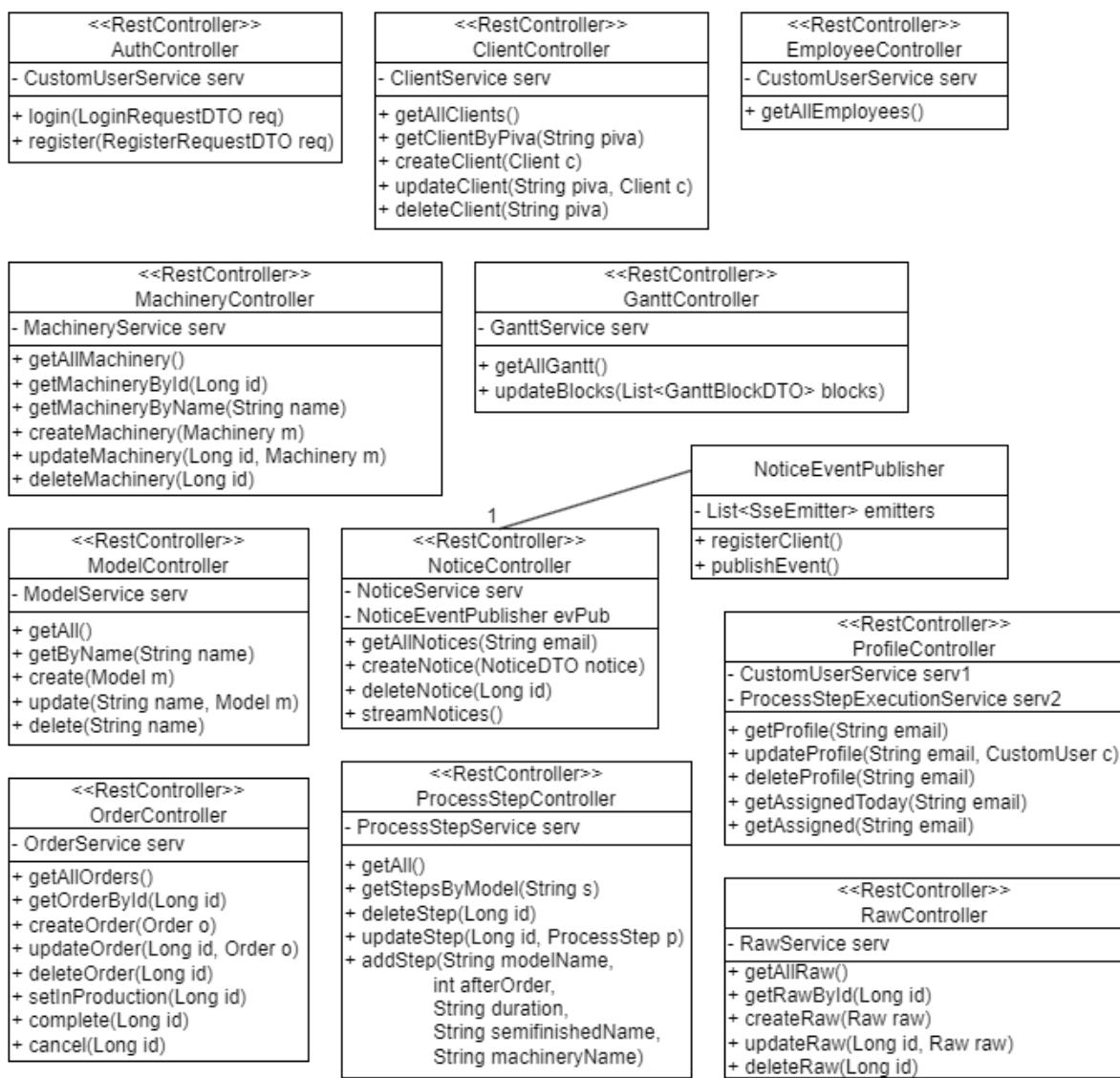


## Controller:

I controller si occupano solo ed esclusivamente di servire le richieste sui vari backend e sono quindi tutti annotati come <<RestController>>. Ogni controller serve i propri endpoint, relativi alla risorsa a cui è riferito.

Ogni metodo corrisponde ad un endpoint e gli unici attributi che hanno sono i relativi servizi.

Fa eccezione solo il **NoticeController**, che dovendo mantenere aperto lo stream degli annunci, viene supportato dal **NoticeEventPublisher**.



## DTO:

Sono stati aggiunti per rimuovere campi sensibili dai model oppure per facilitare la comunicazione di informazioni tra le varie funzioni, incapsulando tutti i dati in un unico oggetto (eventualmente serializzato come JSON).

`AssignedTaskDTO` è stato aggiunto per poter visualizzare solo le informazioni strettamente necessarie della lavorazione assegnata, come nome del macchinario e date.

`GanttBlockDTO` è stato aggiunto per semplicità, in quanto le informazioni al suo interno sono numerose e provengono da tabelle differenti.

`LoginRequestDTO` e `LoginResponseDTO` sono usati nella fase di autenticazione, per nascondere la password e incapsulare il token.

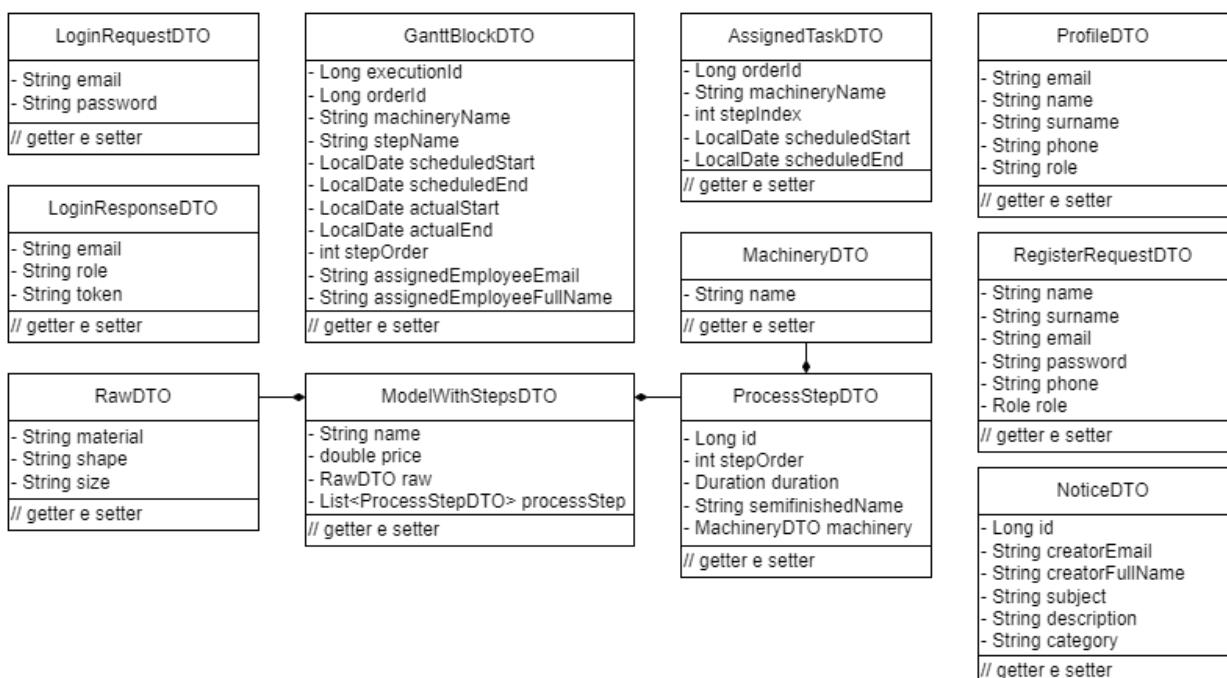
`ModelWithStepsDTO` rappresenta un modello di prodotto con la relativa sequenza di lavorazioni (`ProcessStepDTO`), le materie prime (`RawDTO`) e i macchinari necessari (`MachineryDTO`).

`NoticeDTO` è introdotto per informazioni come la categoria o il nome intero del creatore, che non sono salvati a database ma sono usati nel frontend.

`RegisterRequestDTO` è usato per forzare l'utente ad inserire dati validi nel momento della registrazione.

`ProfileDTO` è di nuovo usato per nascondere la password.

In alcuni casi, per semplicità, abbiamo utilizzato direttamente metodi come `List.of()`, l'approccio con stream e `.map()` o `.toList()` oppure l'approccio con `class.builder()`, in modo da evitare l'aggiunta di troppi DTO potenzialmente banali.

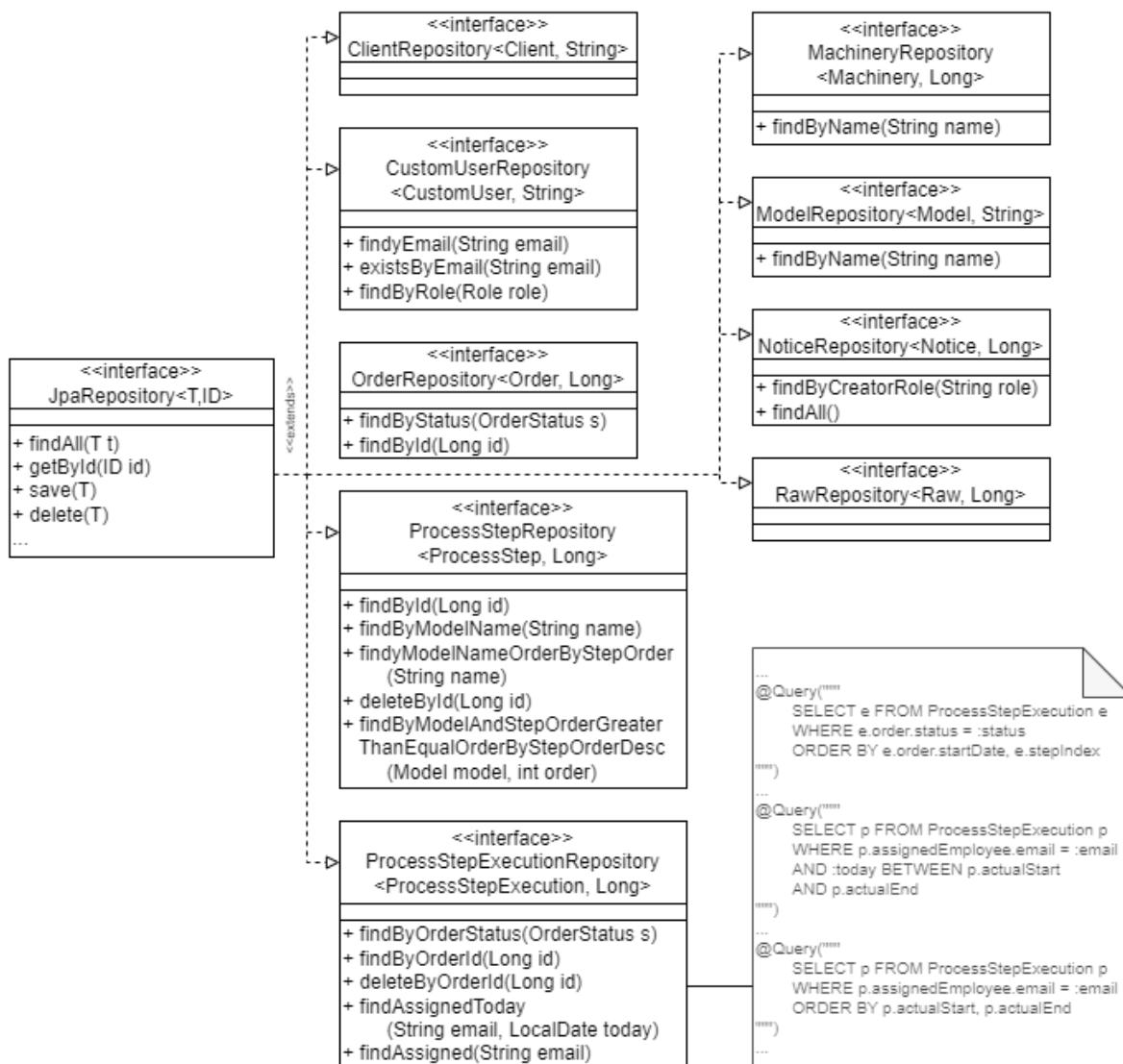


## Repository:

Tutti i file all'interno del repository package sono interfacce che estendono l'interfaccia `JpaRepository<T, ID>` di Hibernate.

La maggior parte dei metodi rappresentano query comuni, come `findById()`, `save()` o `delete()`, e sono già previste da Hibernate.

Fanno eccezione alcune delle query di `ProcessStepExecutionRepository`, che vista la loro complessità, sono state esplicitate con l'apposita annotation `@Query`.



## Security:

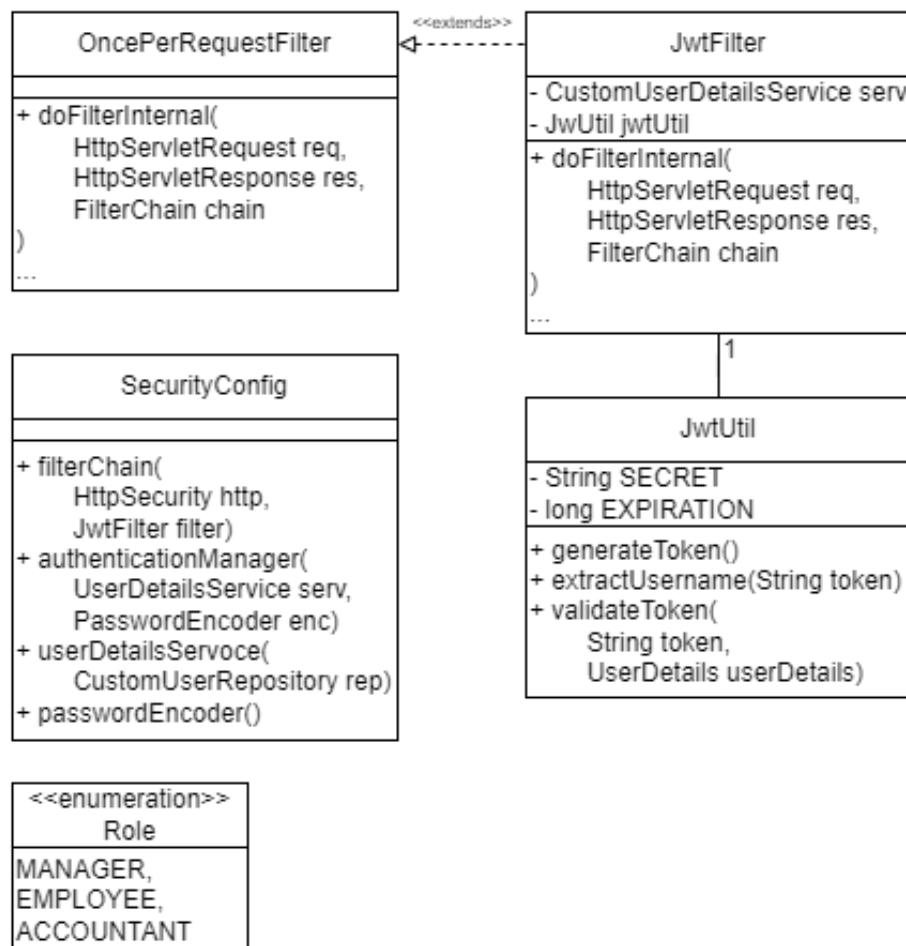
Il package security contiene tutte le classi responsabili della gestione dell'autenticazione e autorizzazione tramite Spring Security e JWT.

`JwtFilter`, che estende `OncePerRequestFilter`, ha il compito di intercettare ogni richiesta, estrarne il token JWT dall'header, verificarne la validità e in caso positivo associare le credenziali dell'utente al contesto di sicurezza di Spring.

`JwtUtil` si occupa di creare i token al momento del login, firmarli con una chiave segreta e verificare la correttezza, in termini di integrità e scadenza, di ogni richiesta.

`SecurityConfig` definisce le regole di sicurezza: nel nostro caso, quali endpoint sono pubblici, quali richiedono autenticazione e quali ruoli hanno accesso a quali risorse.

`Role` è una semplice enumerazione che abbiamo aggiunto per essere sicuri della coerenza nei nomi dei ruoli.



### **Service:**

Il package service contiene la logica applicativa dell'intero sistema. Al suo interno risiedono le classi che implementano le operazioni "di business" e coordinano l'interazione tra controller e repository.

La maggior parte dei servizi semplicemente implementano i metodi necessari per gestire la risorsa a cui si riferiscono, utilizzando il relativo repository.

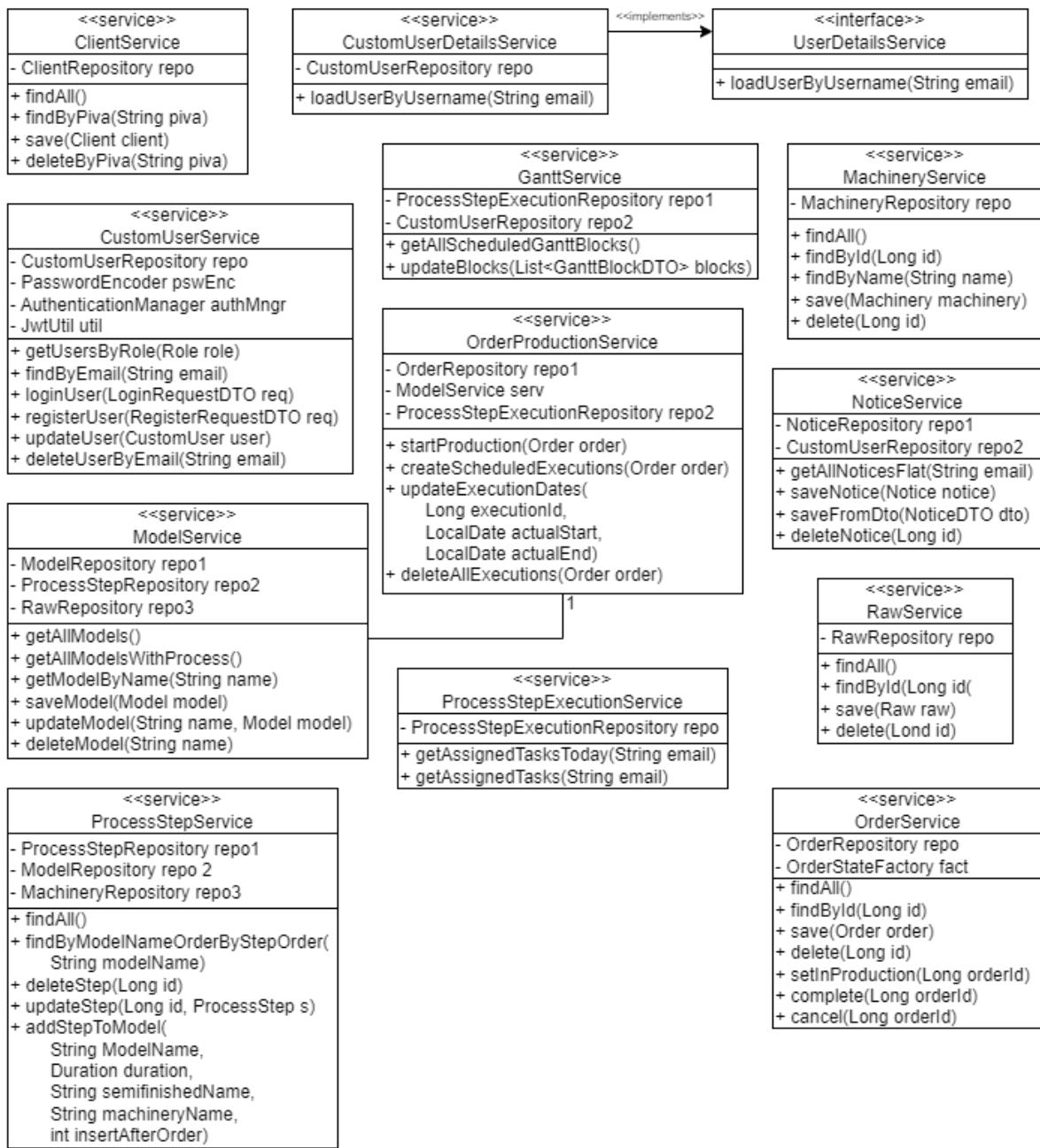
La classe OrderProductionService è il servizio più complesso, che si occupa della pianificazione dell'avvio della produzione degli ordini. Il suo compito non è semplicemente cambiare lo stato di un ordine, ma verificare che esso possa essere effettivamente portato a termine rispettando la deadline stabilita.

Quando un ordine viene mandato in produzione, il servizio:

1. Recupera la sequenza di lavorazioni (ProcessStep) associate a tale modello.
2. Trasforma gli step in esecuzioni vere e proprie (ProcessStepExecution), creando un piano temporale di produzione che rispetti l'ordine degli step.
3. Applica un algoritmo di scheduling simulando l'allocazione delle risorse: controlla quindi che i macchinari richiesti siano disponibili nei tempi previsi, verifica che non ci siano sovrapposizioni tra più ordini e stima i tempi di completamento totali sommando le durate dei vari step.
4. Confronta infine il tempo stimato con la deadline dell'ordine: se l'ordine può esser completato nei tempi richiesti, allora l'algoritmo conferma la produzione e l'ordine passa nello stato IN\_PRODUCTION; se invece la scadenza non può esser rispettata, viene lanciata un'eccezione, la produzione non viene avviata e l'ordine rimane nello stato corrente, in attesa di ripianificazione.

Il GanttService è il componente che gestisce la logica legata ai diagrammi di Gantt del frontend. A partire dai dati presenti nel database, quali ordini e relative lavorazioni, costruisce i blocchi visualizzati nella timeline. Nello specifico, ha il compito di tradurre le informazioni in oggetti compatibili con la libreria FlexGanttFX e gestirne le modifiche.

Lo UserService centralizza la gestione degli utenti del sistema. È in primis responsabile della creazione, modifica e cancellazione degli utenti, nonché della gestione dei ruoli. Attraverso l'integrazione con Spring Security, il servizio fornisce anche i metodi per recuperare gli utenti in fase di autenticazione e autorizzazione.



## 2.7 CLASS DIAGRAMS FRONTEND:

---

### Model:

La maggior parte delle classi in questo package sono uguali a quelle del package del backend, in quanto corrispondono alle entità del dominio applicativo.

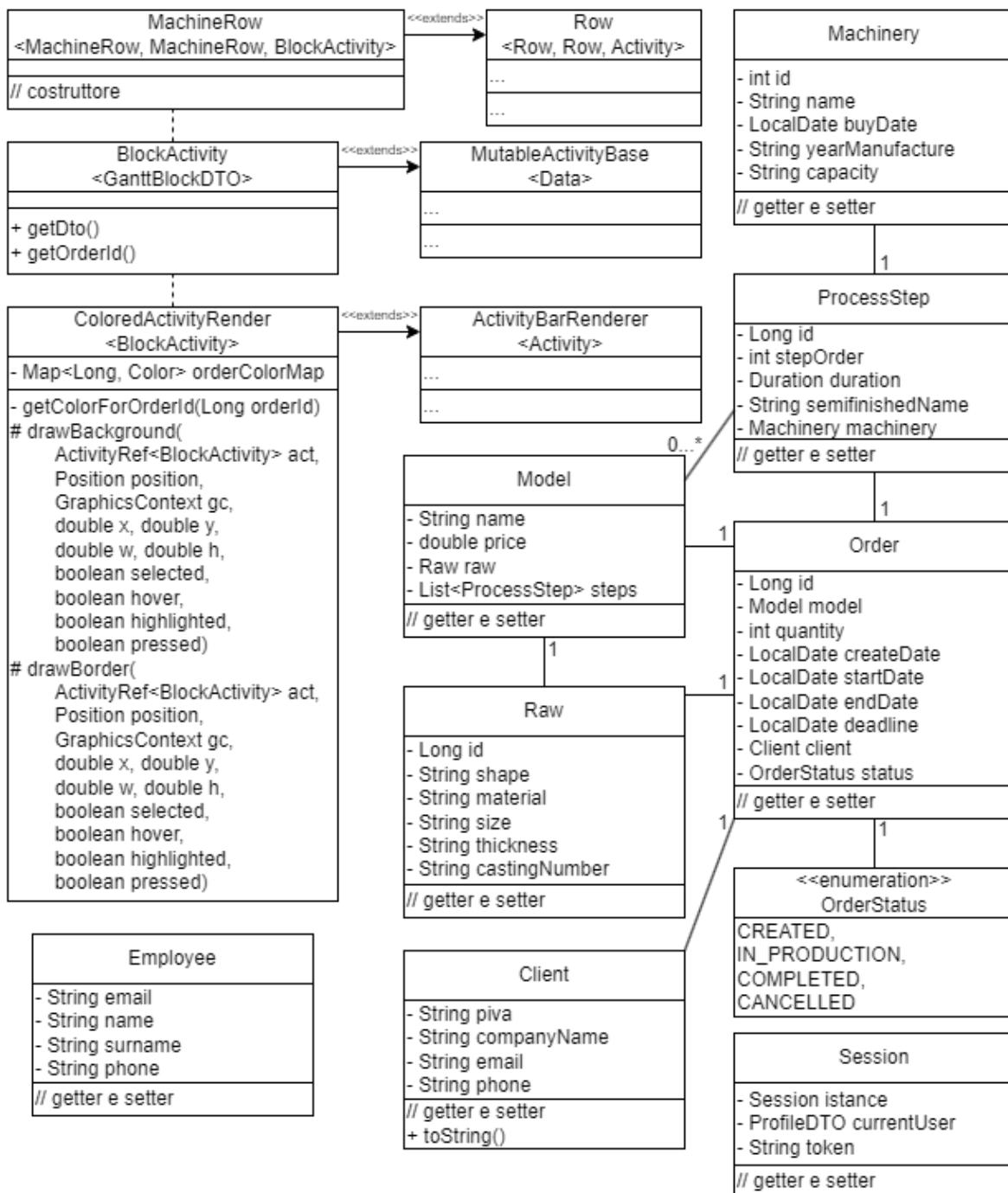
`MachineRow`, `BlockActivity` e `ColoredActivityRenderer` sono le classi necessarie per implementare FlexGanttFX.

`MachineRow`, che estende `flexganttfx.model.Row`, rappresenta una riga nel diagramma di Gantt, corrispondente nel nostro caso ad un macchinario. Ogni riga può contenere uno o più blocchi (`BlockActivity`), che indicano le lavorazioni assegnate al macchinario.

`BlockActivity`, che estende `flexganttfx.model.activity.MutableActivityBase`, rappresenta un'attività all'interno dei diagrammi. Ogni blocco ha un inizio e una fine, e può esser spostato o ridimensionato (da cui il nome `Mutable`). L'aggiunta automatica dei listener rappresenta il motivo per cui anche gli Employee possono modificare i blocchi, anche se la conferma è vietata, e quindi la modifica risulta solo visiva.

`ColoredActivityRenderer`, che estende `flexganttfx.view.graphics.renderer.ActivityBarRenderer` è utilizzata per personalizzare graficamente la rappresentazione dei blocchi. In questo modo, ogni ordine ha il proprio colore, assegnatogli ogni volta che la pagina viene caricata, e può esser facilmente distinto dagli altri, senza necessitare un campo apposito salvato a database.

`Session` è una classe di utilità introdotta per mantenere le informazioni relative all'utente attualmente autenticato. Dal momento che il backend espone soltanto API REST in modalità stateless (quindi senza il concetto di “sessione” lato server), questa classe è stata creata per comodità; consente di tenere in memoria locale i dati dell'utente loggato, come la mail e il token JWT assegnatogli, evitando la copia manuale in ogni richiesta.



## **Controller:**

I controller del frontend hanno il compito principale di gestire le view.

Ogni controller contiene come attributi i riferimenti alle componenti grafiche della view che deve controllare e, quando necessario, i servizi utili a recuperare i dati dal backend.

Ogni controller espone un unico metodo pubblico, `initialize()`, che viene automaticamente eseguito al momento dell'istanziazione del controller, cioè quando la relativa view viene caricata dal loader di JavaFX. Accanto a questo, ciascun controller può includere metodi privati di supporto, utilizzati per organizzare le logiche interne. I metodi di supporto più comuni sono `handleAdd()`, `handleRefresh()`, `handleEdit()` e `handleDelete()`, in quanto la maggior parte delle view ha una vista tabellare ed è gestita principalmente da quattro tasti.

Fanno eccezione tre controller: `HomeController`, `AuthController`, `DashboardController`.

`HomeController` gestisce la finestra iniziale, mostrata all'avvio dell'applicazione, dove l'utente può scegliere se registrarsi o accedere.

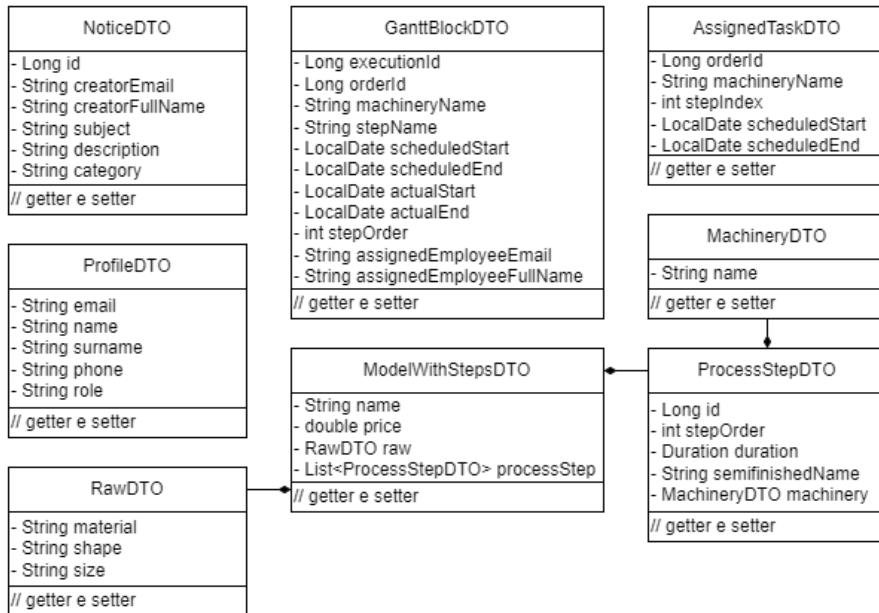
`AuthController` gestisce le finestre dedicate al login e alla registrazione.

`DashboardController` gestisce l'intera finestra principale dell'applicazione. La dashboard è infatti strutturata con una barra di navigazione laterale e un contenitore centrale che rimane vuoto fino al caricamento delle varie sezioni. Questo controller si occupa quindi della navigazione e del caricamento dinamico delle view (e quindi dei relativi controller) all'interno del contenitore.

<b>DashboardController</b>	<b>AuthController</b>	<b>ClientController</b>	<b>ProfileController</b>
<ul style="list-style-type: none"> <li>- AnchorPane contentPane</li> <li>- Button btnProfile</li> <li>- Button btnNoticeBoard</li> <li>- Button btnEmployees</li> <li>- Button btnGantt</li> <li>- Button btnClients</li> <li>- Button btnOrders</li> <li>- Button btnModels</li> <li>- Button btnProcesses</li> <li>- Button btnMachinery</li> <li>- Button btnInventory</li> </ul> <p>+ initialize()</p> <ul style="list-style-type: none"> <li>- handleNoticeBoard()</li> <li>- handleProfile()</li> <li>- handleEmployees()</li> <li>- handleGantt()</li> <li>- handleClients()</li> <li>- handleOrders()</li> <li>- handleModels()</li> <li>- handleProcesses()</li> <li>- handleMachinery()</li> <li>- handleInventory()</li> <li>- handleLogout()</li> <li>- loadContent(String Path)</li> <li>- showAlert(String title, String msg)</li> </ul>	<ul style="list-style-type: none"> <li>- TextField emailField</li> <li>- PasswordField pswField</li> <li>- TextField nameField</li> <li>- TextField surnameField</li> <li>- TextField phoneField</li> <li>- ComboBox&lt;String&gt; role</li> <li>- Label statusLabel</li> </ul> <p>- handleLogin()</p> <ul style="list-style-type: none"> <li>- loadDashboard(String fxmlPath)</li> <li>- handleBack()</li> <li>- handleRegister()</li> </ul>	<ul style="list-style-type: none"> <li>- TableView&lt;Client&gt; clientTable</li> <li>- TableColumn&lt;Client, String&gt; pivaCol</li> <li>- TableColumn&lt;Client, String&gt; compCol</li> <li>- TableColumn&lt;Client, String&gt; emailCol</li> <li>- TableColumn&lt;Client, String&gt; phoneCol</li> <li>- Label statusLabel</li> </ul> <p>+ initialize()</p> <ul style="list-style-type: none"> <li>- loadClientList()</li> <li>- handleRefresh()</li> <li>- handleDelete()</li> <li>- handleAdd()</li> <li>- handleEdit()</li> </ul>	<ul style="list-style-type: none"> <li>- Label emailLabel</li> <li>- Label nameLabel</li> <li>- Label surnameLabel</li> <li>- Label phoneLabel</li> <li>- Label statusLabel</li> <li>- Label welcomeLabel</li> <li>- Label assignedTaskLabel</li> <li>- TableView&lt;AssignedTaskDTO&gt; table</li> <ul style="list-style-type: none"> <li>- TableColumn&lt;AssignedTaskDTO, Long&gt; orderIdCol</li> <li>- TableColumn&lt;AssignedTaskDTO, String&gt; machCol</li> <li>- TableColumn&lt;AssignedTaskDTO, Integer&gt; stepCol</li> <li>- TableColumn&lt;AssignedTaskDTO, LocalDate&gt; sCol</li> <li>- TableColumn&lt;AssignedTaskDTO, LocalDate&gt; eCol</li> </ul> <li>- ProfileService serv</li> <li>- ProfileDTO currentProfile</li> </ul> <p>+ initialize()</p> <ul style="list-style-type: none"> <li>- handleEdit()</li> <li>- handleDelete()</li> <li>- logout()</li> <li>- showStatus(String msg, String color)</li> <li>- loadAssignedTasks()</li> </ul>
<b>HomeController</b>	<b>EmployeeListController</b>	<b>GanttController</b>	<b>RawController</b>
<ul style="list-style-type: none"> <li>- navigate()</li> <li>- onCustomUserLogin(ActionEvent evt)</li> <li>- onCustomUserRegister(ActionEvent evt)</li> </ul>	<ul style="list-style-type: none"> <li>- TableView&lt;Employee&gt; eTable</li> <li>- TableColumn&lt;Employee, String&gt; emailCol</li> <li>- TableColumn&lt;Employee, String&gt; nameCol</li> <li>- TableColumn&lt;Employee, String&gt; surnameCol</li> <li>- TableColumn&lt;Employee, String&gt; phoneCol</li> <li>- EmployeeListService serv</li> </ul> <p>+ initialize()</p> <ul style="list-style-type: none"> <li>- setupTable()</li> <li>- loadEmployeeData()</li> </ul>	<ul style="list-style-type: none"> <li>- AnchorPane contentPane</li> <li>- Button filterButton</li> <li>- ScrollPane ganttScrollPane</li> <li>- Button confirmButton</li> <li>- Button cancelButton</li> <li>- List&lt;BlockActivity&gt; modAct</li> <li>- Set&lt;Long&gt; selOrderIds</li> <li>- GanttService serv1</li> <li>- EmployeeListService serv2</li> <li>- NoticeService serv3</li> </ul> <p>+ initialize()</p> <ul style="list-style-type: none"> <li>- loadAndShowAllOrders()</li> <li>- buildChartFromData(List&lt;GanttBlockDTO&gt; data)</li> <li>- createDelayNotice( <ul style="list-style-type: none"> <li>BlockActivity b, Duration delay,</li> <li>long days, long hours, long minutes)</li> </ul> )</li> <li>- formatDelayHuman( <ul style="list-style-type: none"> <li>long days, long hours, long minutes)</li> </ul> )</li> <li>- createDelayInputFields( <ul style="list-style-type: none"> <li>int days, int hours, int minutes)</li> </ul> )</li> <li>- parseLongSafe(String s)</li> <li>- createNumericField(String initialValue)</li> <li>- onFilterButtonClick()</li> <li>- showGanttWithFilterButton( <ul style="list-style-type: none"> <li>GanttChart&lt;MachineRow&gt; chart</li> </ul> )</li> <li>- openFilterDialog()</li> <li>- handleActivityUpdate(ActivityEvent evt)</li> <li>- showConfirmationButtons()</li> <li>- hideConfirmationButtons()</li> <li>- onConfirmChanges()</li> <li>- onCancelChanges()</li> </ul>	<ul style="list-style-type: none"> <li>- TableView&lt;Raw&gt; table</li> <li>- TableColumn&lt;Raw, Long&gt; idCol</li> <li>- TableColumn&lt;Raw, String&gt; shapeCol</li> <li>- TableColumn&lt;Raw, String&gt; matCol</li> <li>- TableColumn&lt;Raw, String&gt; sizeCol</li> <li>- TableColumn&lt;Raw, String&gt; castNumCol</li> <li>- TableColumn&lt;Raw, String&gt; thickCol</li> <li>- Label statusLabel</li> <li>- RawService serv</li> </ul> <p>+ initialize()</p> <ul style="list-style-type: none"> <li>- validaForm(TextField shapeField,</li> <li>  TextField materialField,</li> <li>  TextField shape, TextField material,</li> <li>  TextField size, Node addButton)</li> <li>- loadRawList()</li> <li>- handleRefresh()</li> <li>- handleDelete()</li> <li>- handleAdd(ActionEvent evt)</li> <li>- handleEdit()</li> </ul>
<b>ModelController</b>	<b>MachineryController</b>	<b>NoticeBoardController</b>	<b>ProcessController</b>
<ul style="list-style-type: none"> <li>- TableView&lt;Model&gt; table</li> <li>- TableColumn&lt;Model, String&gt; nameCol</li> <li>- TableColumn&lt;Model, Long&gt; rawIdCol</li> <li>- Label statusLabel</li> <li>- ModelService serv1</li> <li>- RawService serv2</li> </ul> <p>+ initialize()</p> <ul style="list-style-type: none"> <li>- loadModelList()</li> <li>- handleRefresh()</li> <li>- handleDelete()</li> <li>- handleAdd()</li> <li>- handleEdit()</li> </ul> <p>- validateForm( <ul style="list-style-type: none"> <li>TextField nameField,</li> <li>TextField priceField,</li> <li>ComboBox&lt;Raw&gt; rawCombo,</li> <li>Node button)</li> </ul> )</p>	<ul style="list-style-type: none"> <li>- TableView&lt;Machinery&gt; mTable</li> <li>- TableColumn&lt;Machinery, Integer&gt; idCol</li> <li>- TableColumn&lt;Machinery, String&gt; nameCol</li> <li>- TableColumn&lt;Machinery, LocalDate&gt; buyDateCol</li> <li>- TableColumn&lt;Machinery, String&gt; yearManuCol</li> <li>- TableColumn&lt;Machinery, String&gt; capacityCol</li> <li>- Label statusLabel</li> <li>- MachineryService serv</li> </ul> <p>+ initialize()</p> <ul style="list-style-type: none"> <li>- loadMachineryList()</li> <li>- handleRefresh()</li> <li>- handleDelete()</li> <li>- handleAdd(ActionEvent evt)</li> <li>- handleEdit(ActionEvent evt)</li> <li>- validateForm( <ul style="list-style-type: none"> <li>TextField nameField, DatePicker datePicker,</li> <li>TextField yearManuField,</li> <li>TextField capacityField, Node addButton)</li> </ul> )</li> </ul>	<ul style="list-style-type: none"> <li>- GridPane personalGrid</li> <li>- GridPane delayGrid</li> <li>- GridPane employeeGrid</li> <li>- GridPane managerGrid</li> <li>- GridPane accountantGrid</li> <li>- GridPane otherGrid</li> <li>- Button deleteButton</li> <li>- Label statusLabel</li> <li>- NoticeService serv</li> <li>- NoticeDTO selectedNotice</li> <li>- VBox selectedBox</li> <li>- NoticeEventListener evList</li> </ul> <p>+ initialize()</p> <ul style="list-style-type: none"> <li>- handleNoticeUpdate()</li> <li>- createNoticeBox(NoticeDTO notice,</li> <li>  String userRole, String userEmail)</li> <li>- updateDeleteButton(String userRole,</li> <li>  String userEmail, NoticeDTO not)</li> <li>- addBoxToGrid(GridPane grid, VBox box)</li> <li>- handleRefresh()</li> <li>- handleAdd()</li> <li>- handleDelete()</li> <li>- clearAllGrids()</li> </ul>	<ul style="list-style-type: none"> <li>- TableView&lt;Order&gt; orderTable</li> <li>- TableColumn&lt;Order, Long&gt; idCol</li> <li>- TableColumn&lt;Order, LocalDate&gt; deadlineCol</li> <li>- TableColumn&lt;Order, Integer&gt; quantityCol</li> <li>- TableColumn&lt;Order, String&gt; clientPivaCol</li> <li>- TableColumn&lt;Order, String&gt; modelNameCol</li> <li>- TableColumn&lt;Order, LocalDate&gt; createDateCol</li> <li>- TableColumn&lt;Order, LocalDate&gt; endDateCol</li> <li>- TableColumn&lt;Order, LocalDateTime&gt; startDateTimeCol</li> <li>- TableColumn&lt;Order, String&gt; statusCol</li> <li>- Label statusLabel</li> <li>- Button addButton</li> <li>- Button editButton</li> <li>- Button refreshButton</li> <li>- Button inProductionButton</li> <li>- Button completeButton</li> <li>- Button cancelButton</li> <li>- OrderService serv</li> <li>- ObservableList&lt;Order&gt; orderList</li> </ul> <p>+ initialize()</p> <ul style="list-style-type: none"> <li>- loadOrderList()</li> <li>- handleAdd()</li> <li>- handleEdit()</li> <li>- handleRefresh()</li> <li>- handleInProduction()</li> <li>- handleComplete()</li> <li>- handleCancel()</li> <li>- showAlert(String title, String msg)</li> </ul>

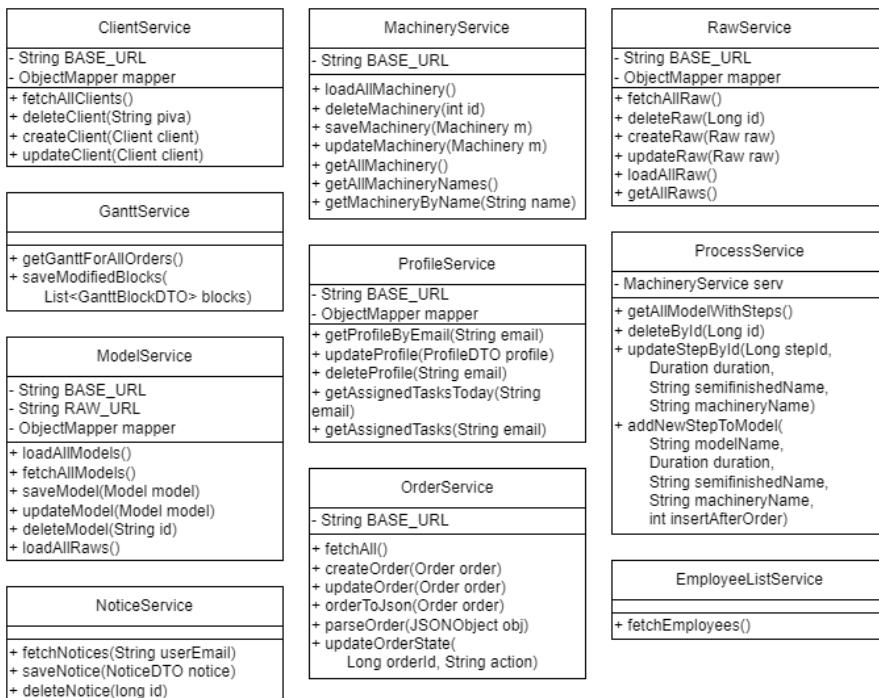
## DTO:

Il package dto del frontend contiene parzialmente le stesse classi presenti nel package omonimo del backend. Non implementano nessuna logica aggiuntiva.



## Service:

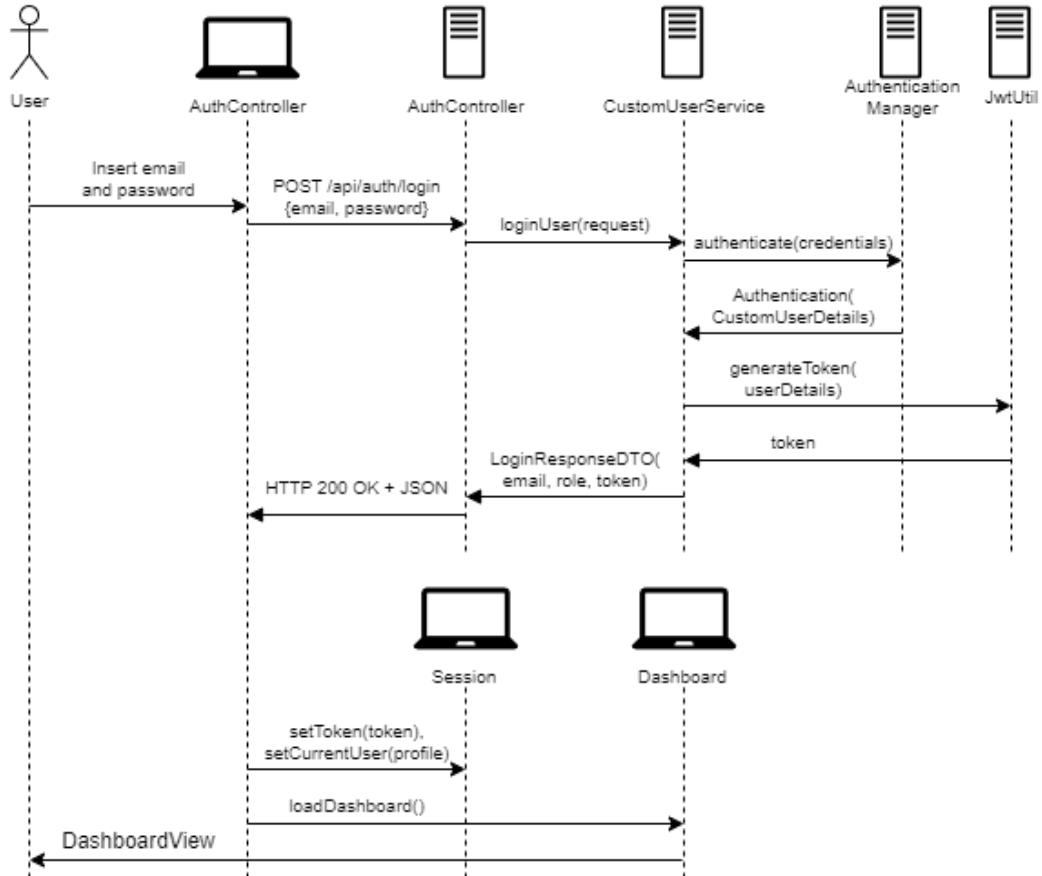
I service nel frontend servono da mediatori col backend, invocandone le API per recuperare o inviare i dati, elaborarli e trasformali eventualmente in formati o oggetti utili per la logica dell'interfaccia.



## 2.8 OTHER DIAGRAMS:

### Sequence Diagram:

Per chiarire meglio la questione dell'autenticazione, presentiamo il relativo Sequence Diagram. Per chiarezza, i moduli frontend sono indicati con un laptop, mentre i moduli backend con un server.



1. L'utente inserisce email e password nella view.
2. Il relativo controller del frontend crea una richiesta HTTP POST all'endpoint corretto inviando un JSON con email e password.
3. La richiesta arriva al backend e viene intercettata dal controller adibito.
4. Il controller richiama il servizio per la gestione degli utenti.
5. Il servizio utilizza l'authentication manager per delegare a Spring Security la verifica delle credenziali.
6. Se corrette, Spring Security risponde coi dati dell'utente.
7. A questo punto il servizio può generare un token JWT per l'utente.
8. Email, ruolo e token vengono incapsulati in un DTO e reinviati al frontend.
9. Il frontend riceve la risposta e salva i dati.

Tutte le richieste successive dovranno avere `Authorization: Bearer "token"` tra gli header.

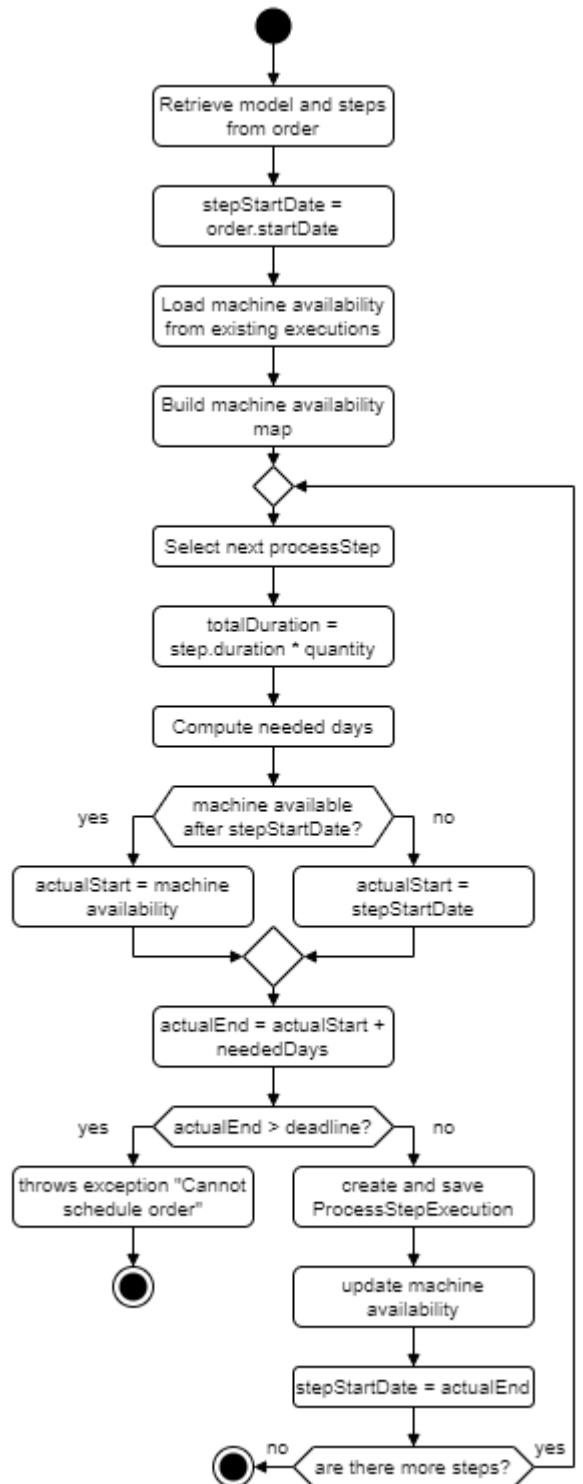
### Activity Diagram:

Per comprendere meglio come viene gestita la messa in produzione, presentiamo l'Activity Diagram dell'algoritmo di scheduling.

L'algoritmo di scheduling si occupa di pianificare l'esecuzione di un ordine in base al suo modello e ai relativi passi. Per ciascun passo produttivo:

1. Recupera il modello e i relativi step.
2. Inizializza una mappa con le disponibilità delle macchine, tenendo conto delle esecuzioni già pianificate.
3. Per ogni step del modello:
  - a. Calcola la durata totale dello step in funzione della quantità di elementi dell'ordine.
  - b. Determina la prima data utile per l'avvio confrontando la disponibilità della prima macchina e la data prevista.
  - c. Verifica che la fine dello step non superi la deadline dell'ordine.
  - d. Crea un nuovo oggetto che rappresenta la produzione.
  - e. Aggiorna le disponibilità della macchina e la data di partenza dello step successivo.
4. Se la deadline non può esser rispettata, l'algoritmo solleva un'eccezione.

In questo modo si garantisce che gli ordini siano schedulati senza conflitti e rispettando i vincoli temporali.



## 2.9 MULTITHREADING:

Il multithreading, in quanto requisito chiave del progetto, è stato ampiamente usato, sia direttamente che indirettamente.

Spring, grazie al suo container servlet incorporato Tomcat, supporta di default il multithreading, gestendo in modo automatico la concorrenza delle richieste HTTP. In pratica, ogni richiesta viene assegnata ad un thread separato proveniente da un pool.

Per limitazioni dell'IDE utilizzato, IntelliJ, non è stato possibile avviare contemporaneamente due istanze dell'app. Prima dell'aggiunta di Spring Security, quindi prima di bloccare le rotte al di fuori di quelle dell'applicazione, è stato possibile verificarlo con strumenti come `cURL` o `Invoke-RestMethod`, eseguendo comandi come il seguente, confermando il funzionamento del multithreading nel contesto reale:

```
PowerShell      notice insert and delete tryout

1 $body = @{
2     creatorEmail = "m"
3     Subject = "Test inserimento da PowerShell"
4 } | ConvertTo-Json
5
6 $headers = @{
7     "Content-Type" = "application/json"
8 }
9
10 Invoke-RestMethod -Uri
11     http://localhost:8080/api/notice/add?creatorEmail=m
12     -Method Post
13     -Body $body
14     -Headers $headers
15
16 Invoke-RestMethod -Uri http://localhost:8080/api/notice/delete/19 -
    Method Delete
```

Nel frontend, il multithreading è stato utilizzato in maniera esplicita per mantenere l'interfaccia reattiva. Le chiamate al backend vengono eseguite in thread separati rispetto al JavaFX Application Thread, che è responsabile dell'aggiornamento della UI. In questo modo, eventuali errori o rallentamenti nelle richieste, non bloccano l'interfaccia.

Un altro aspetto importante è l'uso di `Platform.runLater()`. Questo metodo non crea un nuovo thread, ma permette di accodare operazioni da eseguire in sicurezza sul JavaFX Application Thread. È stato ampiamente utilizzato per aggiornare i componenti grafici dopo che un thread separato ha completato un'elaborazione o ha ricevuto una risposta dal server.

Non si tratta di multithreading in senso stretto, ma comunque gestione e sincronizzazione dei processi, per evitare conflitti e garantire un'interfaccia stabile.

Il seguente metodo, usato nel `ModelController`, mostra in pratica come viene gestito il multithreading nel frontend.

```
Java    private void loadModelList()  
  
...  
46     new Thread(() -> {  
47         try{  
48             List<Model> models = modelService.loadAllModels();  
49             Platform.runLater(() -> {  
50                 modelTable.getItems().addAll(models);  
51                 statusLabel.setText("Loaded " + models.size() + " models");  
52                 statusLabel.setStyle("-fx-text-fill: green;");  
53             });  
54         }catch(Exception e){  
55             e.printStackTrace();  
56             Platform.runLater(() -> {  
57                 statusLabel.setText("Error while loading");  
58                 statusLabel.setStyle("-fx-text-fill: red;");  
59             });  
60         }  
61     }).start();  
...
```

1. La chiamata `new Thread(() -> {...}).start();` crea e avvia un thread per eseguire il metodo del servizio e interrogare il backend per ottenere la lista di modelli.
2. Una volta ottenuti i dati, non è possibile aggiornare direttamente la UI in un thread diverso. Per questo motivo, viene usato `Platform.runLater()`, che accoda il codice di modifica dell'interfaccia all'interno del `JavaFX Application Thread`. In questo caso, si popola la tabella e si aggiorna l'etichetta che mostra i messaggi in fondo alla pagina.
3. Se si verifica un'eccezione, come per esempio la mancata risposta del server, essa viene catturata nel blocco `catch` e l'utente viene notificato usando nuovamente `Platform.runLater()` per mostrare un messaggio di errore.

# 3. Terza Sezione:

## 3.1 PIANO DI TEST:

---

Il testing rappresenta una fase fondamentale nello sviluppo di un qualsiasi software, nonché una delle specifiche fornite a inizio progetto, in quanto consente di verificare la correttezza delle funzionalità implementate e di garantire un adeguato livello di qualità e affidabilità del sistema.

Per il testing sono stati usati tre strumenti principali: JUnit, Mockito, e TestFX.

La strategia di testing adottata si è incentrata principalmente sui test di unità (grazie a JUnit), col supporto di test di integrazione (grazie a Mockito) e a test di interfaccia (grazie a TestFX). Questa combinazione ha permesso di coprire i diversi aspetti del sistema, mantenendo un focus sulla logica di business.

### Backend:

Per la parte di backend è stata effettuata una copertura completa dei package `controller` e `service`, in modo da garantire che tutte le funzionalità principali del sistema fossero testate.

I test dei controller verificano il corretto funzionamento degli endpoint REST esposti: ogni test simula una chiamata HTTP (`GET`, `POST`, `PUT` o `DELETE`) verso l'endpoint corrispondente, controllando che la risposta abbia lo status code corretto e che i dati restituiti siano coerenti con quelli attesi. In questa fase, Mockito è stato usato per simulare i service sottostanti e isolare il comportamento dei controller.

I test dei service verificano invece la logica applicativa interna del sistema: per ogni risorsa, sono stati implementati test unitari per le operazioni di creazione, modifica, eliminazione e ricerca (CRUD), controllando che venissero rispettati tutti i vincoli e i controlli. In questa fase Mockito è stato utilizzato per simulare i repository e le dipendenze, in modo da non dover interagire col database reale.

Tutti i test che riguardano metodi CRUD standard hanno il seguente schema:

- Creazione di un oggetto in input.
- Mock della risposta del service o del repository.
- Verifica del risultato ottenuto col risultato atteso.

Poiché questo schema si ripete nella maggior parte dei file, riportiamo il test case del `ClientController` come esempio.

<b>Operazione</b>	<b>Scenario</b>	<b>Input</b>	<b>Output atteso</b>
Get All Clients	Successo	Lista di clienti mockata	200 OK + lista clienti
Get Client by PIVA	Cliente trovato	PIVA esistente "123"	200 OK + cliente trovato
Get Client by PIVA	Cliente non trovato	PIVA non esistente "123"	404 not found + messaggio "Client not Found!"
Create Client	Successo	Nuovo cliente valido	201 created + cliente creato
Create Client	Fallimento	Nuovo cliente con valore non valido	400 bad request + messaggio "'Failed to create client: DB error"
Update Client	Cliente trovato	PIVA "123" e nuovo client modificato	200 OK + salvataggio cliente aggiornato
Update Client	Cliente non trovato	PIVA non esistente "123"	404 not found + messaggio "Client not found!"
Delete Client	Cliente trovato	PIVA "123"	200 OK + messaggio "Client deleted successfully!"
Delete Client	Cliente non trovato	PIVA non esistente "123"	404 not found + messaggio "Client not found!"

Di seguito riportiamo invece il test case condotto per `OrderProductionService`, che non si limita a verificare la corretta esecuzione di operazioni CRUD ma testa la gestione della produzione degli ordini, nello specifico verifica il passaggio tra stati e le relative operazioni di creazione, aggiornamento o eliminazione delle esecuzioni.

Operazione	Scenario	Input	Output atteso
Start Production	Successo	Ordine con stato CREATED e modello valido	Stato aggiornato a IN_PRODUCTION, startDate impostata, salvataggio di ordine ed esecuzioni
Start Production	Ordine già in produzione	Ordine con stato IN_PRODUCTION	Throw IllegalStateException
Start Production	Ordine completato	Ordine con stato COMPLETED	Throw IllegalStateException
Start Production	Ordine cancellato	Ordine con stato CANCELLED	Throw IllegalStateException
Update Execution Dates	Successo	ID esecuzione esistente e nuove date	Aggiornamento di actualStart e actualEnd, salvataggio con repository
Delete All Executions	Successo	Ordine con stato IN_PRODUCTION	endDate impostata, cancellazione esecuzioni legate all'ordine, salvataggio ordine

I package `dto` e `model` contengono classi molto semplici: la maggior parte possiede solo metodi getter e setter, e non vale quindi la pena testarli. Hanno inoltre una coverage particolarmente alta di default.

Il package `security` è principalmente gestito da Spring e risulta quindi particolarmente complicato da testare.

Il package `repository`, infine, essendo solo un'interfaccia per Hibernate, non possiede implementazioni, e non è quindi possibile da testare. Viene automaticamente considerato col 100% di coverage.

## Frontend:

Anche per la parte di frontend sono stati realizzati test esclusivamente sui package controller e service, in quanto gli unici a contenere logica applicativa.

I controller gestiscono l'interazione tra l'interfaccia grafica e i dati recuperati dai servizi dal backend e sono quindi testati principalmente con TestFX, mentre i servizi encapsulano le chiamate HTTP alle API REST del server e sono stati controllati principalmente con JUnit e Mockito.

La maggior parte delle finestre dell'applicazione è di tipo CRUD (Create, Read, Update, Delete) e di conseguenza i relativi test sono molto simili tra di loro. In particolare, la maggior dei controlli ha metodi come `handleRefresh()`, `handleAdd()`, `handleEdit()` e `handleDelete()`, che richiedono controlli ripetitivi sulla corretta esecuzione delle chiamate e sulla gestione di eventuali errori. Per questi casi, i test hanno sempre lo stesso schema:

- Mock delle dipendenze (service e HttpURLConnection).
- Invocazione del metodo sotto test.
- Verifica del risultato atteso.

Poiché questo schema si ripete nella maggior parte dei file, riportiamo il test case del `RawController` come esempio.

Operazione	Scenario	Input	Output atteso
Refresh	Successo	/	La label di stato mostra “Refreshing...” e successivamente quanti elementi ha caricato
Load Raw List	Successo	Due elementi fintizi	La tabella contiene 2 elementi, la label di stato mostra “Loaded 2 items” in verde
Delete Raw	Successo	Un elemento fintizio	L'elemento viene rimosso dalla tabella, la label di stato mostra “Deleted item id=1” in verde
Delete Raw	No Selection	/	La label di stato mostra “Select an item to delete” in rosso
Add	Successo	Nuovo elemento fintizio	La tabella contiene il nuovo Raw, la label di stato mostra “Added Material with ID=100”
Edit	Successo	Nuovo elemento fintizio	L'elemento viene aggiornato in tabella, la label di stato mostra “Edit Material with ID=1”

Un esempio di testing un po' più particolare è rappresentato dal testing sull'`AuthController`, che si occupa di verificare la logica di autenticazione e registrazione degli utenti. A differenza degli altri controller, si interfaccia direttamente col backend, scambiando dati in formato JSON e aggiornando la UI di conseguenza.

Questo rende il test più articolato in quanto è necessario simulare anche la connessione HTTP, gestire i flussi di input e output, testare scenari di errore come server non raggiungibile o credenziali invalide, e simulare anche la navigazione interna a seguito di un'autenticazione con successo.

Operazione	Scenario	Input	Output atteso
Login	Empty Fields	Mancate credenziali	Mostra messaggio "All fields are required"
Login	Failure Response	Credenziali errate	Mostra messaggio "Invalid credentials" in rosso
Login	Eccezione	Errore durante la richiesta	Mostra messaggio "Server error" in rosso
Login	Missing token	Login di risposta priva di token	Mostra messaggio "Server error" in rosso
Register	Successo	Nuovo utente con dati validi	Mostra messaggio "Registration Successful!" in verde
Register	Empty Fields	Interfaccia con campi vuoti	Mostra messaggio "All fields are required"
Register	Failure Response	Registrazione con richiesta non valida	400 Bad Request + Mostra messaggio "Bad request" in rosso
Register	Eccezione	Eccezione durante la richiesta	Mostra messaggio "Internal error during registration" in rosso
Register	Null Stream	Credenziali valide	500 Internal Error + Mostra messaggio "Generic error!" in rosso
Back	Successo	/	Nessuna eccezione sollevata
Back	Invalid Fxml	FXML errato o inesistente	Nessuna eccezione sollevata
Load Dashboard	Successo	FXML valido	Nessuna eccezione sollevata
Load DashBoard	Invalid FXML	FXML errato o inesistente	Mostra "Error while loading dashboard" in rosso

### **3.2 RAPPORTO COVERAGE:**

---

Il rapporto sulla coverage mostra risultati soddisfacenti, con una percentuale media complessiva di copertura delle linee attorno al 76%, con differenze sostanziali tra i package.

**Controller** e **Service** hanno le coverage più alte, in quanto manualmente testati.

**Repository** ha 100% di coverage essendo composto solo da interfacce per Hibernate.

**View** non è testabile, trattandosi di file XML statici, ed è quindi escluso dal conteggio.

**DTO** e **Model**, esclusi dai test per banalità, mantengono comunque percentuali elevate di coverage di default.

**Security** nel backend è un package escluso totalmente dal testing a causa della complessità.

**GanttController** nel frontend è l'unico file escluso totalmente dal testing a causa della complessità della libreria esterna.

Di seguito la coverage del backend:

Element ^	Class, %	Method, %	Line, %	Branch, %
server_group	85% (49/57)	79% (308/387)	83% (710/855)	73% (89/121)
> controller	100% (12/12)	96% (60/62)	97% (191/196)	100% (36/36)
> dto	100% (11/11)	69% (76/110)	68% (106/155)	44% (4/9)
> model	70% (12/17)	73% (101/138)	65% (106/162)	0% (0/4)
> repository	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
> security	50% (2/4)	58% (7/12)	39% (20/51)	0% (0/10)
> service	100% (12/12)	100% (64/64)	98% (287/290)	79% (49/62)
<b>Server</b>	0% (0/1)	0% (0/1)	0% (0/1)	100% (0/0)

Di seguito la coverage del frontend:

Element ^	Class, %	Method, %	Line, %	Branch, %
client_group	86% (50/58)	76% (415/540)	73% (2022/2738)	49% (236/478)
> controller	94% (17/18)	64% (170/264)	64% (1154/1789)	34% (98/284)
> dto	87% (7/8)	93% (80/86)	93% (92/98)	100% (0/0)
> model	71% (10/14)	79% (87/109)	70% (107/151)	20% (4/20)
> service	94% (16/17)	98% (78/79)	96% (669/693)	77% (134/174)
<b>Main</b>	0% (0/1)	0% (0/2)	0% (0/7)	100% (0/0)

### 3.3 RISULTATI TEST:

#### Backend:

<b>OrderProductionServiceTest</b>	1.40 s	<b>ClientServiceTest</b>	37 ms
startProduction_alreadyInProduction_throws()	passed 1.33 s	save_client_success()	passed 37 ms
deleteAllExecutions_success()	passed 44 ms	findByPiva_notFound()	passed
startProduction_alreadyCompleted_throws()	passed 2 ms	findAll_returnsList()	passed
startProduction_alreadyCancelled_throws()	passed 2 ms	findByPiva_found()	passed
updateExecutionDates_success()	passed 15 ms	deleteByPiva_callsRepository()	passed
startProduction_success()	passed 9 ms		
<b>ModelControllerTest</b>	35 ms	<b>GanttServiceTest</b>	5 ms
delete_successful()	passed 23 ms	updateBlocks_removeAssignment_success()	passed 1 ms
getByName_notFound()	passed 3 ms	updateBlocks_executionNotFound_throws()	passed 1 ms
getModelsWithProcesses_successful()	passed 2 ms	updateBlocks_assignEmployee_success()	passed 1 ms
update_found()	passed 2 ms	updateBlocks_userNotFound_throws()	passed
create_successful()	passed 1 ms	updateBlocks_userNotEmployee_throws()	passed 1 ms
getByName_found()	passed 1 ms	getAllScheduledGanttBlocks_success()	passed 1 ms
delete_notFound()	passed 1 ms		
update_notFound()	passed 1 ms	<b>EmployeeControllerTest</b>	28 ms
getAll_successful()	passed 1 ms	getAllEmployees_successful()	passed 28 ms
<b>MachineryControllerTest</b>	43 ms	<b>ProvaSpringApplicationTests</b>	13 ms
getAllMachinery_successful()	passed 26 ms	contextLoads()	passed 13 ms
deleteMachinery_found()	passed 3 ms	<b>AuthControllerTest</b>	1 ms
createMachinery_successful()	passed 1 ms	login_invalidCredentials()	passed 1 ms
updateMachinery_notFound()	passed 1 ms	register_emailAlreadyExists()	passed
getMachineryByName_notFound()	passed 2 ms	login_successful()	passed
updateMachinery_found()	passed 2 ms	register_successful()	passed
getMachineryByBId_notFound()	passed 1 ms		
getMachineryByBId_found()	passed 1 ms	<b>CustomUserServiceTest</b>	138 ms
getMachineryByName_found()	passed 1 ms	findByEmail_found()	passed 96 ms
deleteMachinery_notFound()	passed 1 ms	getUsersByRole_returnsList()	passed 1 ms
createMachinery_failure()	passed 4 ms	registerUser_success()	passed 1 ms
<b>RawServiceTest</b>	55 ms	loginUser_badCredentials()	passed 1 ms
delete_success()	passed 47 ms	deleteUserByEmail_callsRepository()	passed 1 ms
findById_notFound()	passed 2 ms	updateUser_savesUser()	passed
save_success()	passed 3 ms	loginUser_success()	passed 36 ms
findById_found()	passed 2 ms	findByEmail_notFound()	passed 1 ms
findAll_success()	passed 1 ms	registerUser_emailExists()	passed 1 ms
<b>MachineryServiceTest</b>	46 ms		
delete_success()	passed 45 ms	<b>NoticeControllerTest</b>	64 ms
findByName_found()	passed 1 ms	deleteNotice_successful()	passed 59 ms
findById_notFound()	passed	streamNotices_successful()	passed 3 ms
findByName_notFound()	passed	deleteNotice_notFound()	passed 1 ms
save_success()	passed	createNotice_successful()	passed 1 ms
findById_found()	passed	getAllNotices_successful()	passed
findAll_success()	passed		
<b>ProcessStepControllerTest</b>	1.54 s	<b>OrderControllerTest</b>	155 ms
testGetStepsByModel()	passed 1.35 s	testDeleteOrder_NotFound()	passed 48 ms
testAddStep()	passed 23 ms	testGetOrderById_Found()	passed 12 ms
testUpdateStep_Success()	passed 122 ms	testSetInProduction()	passed 7 ms
testDeleteStep()	passed 12 ms	testUpdateOrder_Found()	passed 18 ms
testGetAll()	passed 15 ms	testGetOrderById_NotFound()	passed 7 ms
testUpdateStep_Exception()	passed 16 ms	testCreateOrder_Success()	passed 13 ms
		testDeleteOrder_Found()	passed 7 ms
		testComplete()	passed 7 ms
		testCreateOrder_Failure()	passed 12 ms
		testGetAllOrders()	passed 9 ms
		testCancel()	passed 5 ms
		testUpdateOrder_NotFound()	passed 10 ms

<b>RawControllerTest</b>	136 ms	<b>ProfileControllerTest</b>	32 ms
testDeleteRaw_Found()	passed 40 ms	deleteProfile_successful()	passed 29 ms
testGetRawById_Found()	passed 17 ms	getProfile_successful()	passed 1 ms
testUpdateRaw_Found()	passed 14 ms	getProfile_notFound()	passed
testCreateRaw_Success()	passed 11 ms	updateProfile_successful()	passed 1 ms
testDeleteRaw_NotFound()	passed 9 ms	deleteProfile_notFound()	passed 1 ms
testCreateRaw_Failure()	passed 18 ms	updateProfile_notFound()	passed
testGetRawById_NotFound()	passed 8 ms		
testUpdateRaw_NotFound()	passed 10 ms		
testGetAllRaw()	passed 9 ms		
<b>GanttControllerTest</b>	21 ms	<b>NoticeEventPublisherTest</b>	4 ms
getAllGantt_successful()	passed 19 ms	registerClient_shouldAddEmitter()	passed 2 ms
updateBlocks_failure()	passed 1 ms	publishEvent_withActiveEmitter_shouldSendEvent()	passed 2 ms
updateBlocks_successful()	passed 1 ms		
<b>ModelServiceTest</b>	88 ms	<b>ClientControllerTest</b>	32 ms
deleteModel_notExists_false()	passed 78 ms	updateClient_found()	passed 32 ms
deleteModel_exists_true()	passed 1 ms	getClientByPiva_notFound()	passed
saveModel_withoutRawId_savesDirectly()	passed 1 ms	deleteClient_notFound()	passed
getAllModels_success()	passed 1 ms	createClient_failure()	passed
getAllModelsWithProcess_success()	passed 1 ms	updateClient_notFound()	passed
getModelByName_notFound()	passed 1 ms	getClientByPiva_found()	passed
getModelByName_found()	passed	createClient_successful()	passed
updateModel_rawNotFound_throws()	passed 1 ms	getAllClients_successful()	passed
saveModel_withRawId_fetchesRaw()	passed 1 ms	deleteClient_found()	passed
saveModel_withRawId_notFound_throws()	passed 1 ms		
updateModel_notFound_returnsEmpty()	passed 1 ms		
updateModel_success_withRaw()	passed 1 ms		
<b>ProcessStepExecutionServiceTest</b>	4 ms	<b>OrderServiceTest</b>	66 ms
getAssignedTasks_empty()	passed 1 ms	setInProduction_success()	passed 59 ms
getAssignedTasksToday_empty()	passed 1 ms	delete_success()	passed
getAssignedTasks_success()	passed 1 ms	cancel_success()	passed 1 ms
assignedTaskDTO_gettersAndSetters()	passed	complete_orderNotFound_throws()	passed 1 ms
getAssignedTasksToday_success()	passed 1 ms	findById_success()	passed 1 ms
<b>NoticeServiceTest</b>	76 ms	complete_success()	passed 1 ms
saveFromDto_success()	passed 70 ms	setInProduction_orderNotFound_throws()	passed 1 ms
saveFromDto_missingCreatorEmail_throws()	passed 1 ms	save_success()	passed 1 ms
saveFromDto_userNotFound_throws()	passed 1 ms	cancel_orderNotFound_throws()	passed
saveNotice_success()	passed 1 ms	findAll_success()	passed 1 ms
deleteNotice_notExists_false()	passed 1 ms		
getAllNoticesFlat_success()	passed 1 ms		
deleteNotice_exists_true()	passed 1 ms		
<b>ProcessStepServiceTest</b>	5 ms	<b>CustomUserDetailsServiceTest</b>	2 ms
deleteStep_notFound()	passed	loadUserByUsername_found()	passed 1 ms
updateStep_notFound()	passed	loadUserByUsername_notFound()	passed 1 ms
deleteStep_success_withReorder()	passed		
addStepToModel_machineryNotFound()	passed		
addStepToModel_modelNotFound()	passed		
findByNameOrderByStepOrder_success()	passed		
addStepToModel_success_withShift()	passed		
updateStep_success()	passed		
findAll_success()	passed		

## Frontend:

<b>ModelControllerAddEditTestFX</b>	8.75 s	<b>ClientControllerTest</b>	6.80 s
testHandleAddConfirm()	passed 5.20 s	testHandleEditSuccess()	passed 579 ms
testHandleRefresh()	passed 220 ms	testHandleEditFailure()	passed 540 ms
testHandleDeleteSuccess()	passed 325 ms	testHandleDeleteWithoutSelection()	passed 540 ms
testHandleDeleteNoSelection()	passed 217 ms	testInitializeHandlesException()	passed 538 ms
testHandleEditConfirm()	passed 2.78 s	testHandleRefreshUpdatesStatus()	passed 541 ms
<b>ProfileServiceTest</b>	347 ms	testHandleAddSuccess()	passed 543 ms
testGetAssignedTasksToday_success()	passed 298 ms	testHandleDeleteWithSelection()	passed 1.04 s
testGetAssignedTasks_success()	passed 5 ms	testHandleEditDialogCancelled()	passed 541 ms
testDeleteProfile_success()	passed 2 ms	testHandleEditNoSelection()	passed 542 ms
testUpdateProfile_success()	passed 39 ms	testInitializeLoadsClients()	passed 544 ms
testGetProfileByEmail_success()	passed 3 ms	testHandleAddFailure()	passed 846 ms
<b>ModelServiceTest</b>	29 ms	<b>GanttServiceTest</b>	9 ms
testDeleteModel_success()	passed 1 ms	testSaveModifiedBlocksSuccess()	passed 3 ms
testUpdateModel_success()	passed 7 ms	testSaveModifiedBlocksOtherException()	passed
testLoadAllModels_success()	passed 9 ms	testSaveModifiedBlocksHttpError()	passed 2 ms
testSaveModel_success()	passed 6 ms	testGetGanttForAllOrdersSuccess()	passed 4 ms
testFetchAllModels_exception()	passed 3 ms	<b>OrderControllerTestFX</b>	6.37 s
testLoadAllRaws_success()	passed 2 ms	testHandleAddConfirm()	passed 395 ms
testSaveModel_failure()	passed 1 ms	testLoadOrderListSuccess()	passed 718 ms
<b>MachineryControllerTestFX</b>	11.21 s	testInitializeSetsColumns()	passed 718 ms
testLoadMachineryListSuccess()	passed 740 ms	testHandleAddFailure()	passed 725 ms
testHandleAddDialog()	passed 3.89 s	testHandleEditDialog()	passed 3.81 s
testHandleDeleteSuccess()	passed 921 ms	<b>NoticeServiceTest</b>	7 ms
testHandleDeleteNoSelection()	passed 422 ms	testDeleteNoticeException()	passed 1 ms
testHandleEditDialog()	passed 5.24 s	testFetchNoticesException()	passed
<b>ProcessControllerTest</b>	156 ms	testSaveNoticeSuccess()	passed 2 ms
testHandleAddFirstStepConfirm()	passed 111 ms	testFetchNoticesCallsDisconnect()	passed 1 ms
testHandleEditStepCancel()	passed 6 ms	testFetchNoticesSuccess()	passed 1 ms
testInitializeWithSteps()	passed 9 ms	testFetchNoticesFailure()	passed
testInitializeWithNoSteps()	passed 2 ms	testSaveNoticeException()	passed 1 ms
testHandleAddStepCancel()	passed 5 ms	testSaveNoticeFailureNon2xxResponse()	passed 1 ms
testInitializeExceptionHandling()	passed 1 ms	testDeleteNoticeSuccess()	passed
testHandleAddStepConfirm()	passed 8 ms	testDeleteNoticeFailure()	passed
testHandleDeleteStepConfirm()	passed 14 ms	testDefaultConstructor()	passed

<b>OrderServiceTest</b>	16 ms	<b>DashboardControllerTestFX</b>	5.02 s
testFetchAll_failure()	passed 2 ms	testHandleProfile()	passed 343 ms
testCreateOrder_success()	passed 5 ms	testInitializeAccountant()	passed 337 ms
testCreateOrder_failure()	passed 1 ms	testInitializeEmployee()	passed 341 ms
testUpdateOrder_success()	passed 1 ms	testInitializeManager()	passed 342 ms
testUpdateOrderState_success()	passed 1 ms	testInitializeUnknownRole()	passed 340 ms
testUpdateOrder_failure()	passed 1 ms	testHandleInventory()	passed 341 ms
testFetchAll_success()	passed 4 ms	testHandleClients()	passed 344 ms
testUpdateOrderState_failure()	passed 1 ms	testHandleEmployees()	passed 340 ms
<b>ClientServiceTest</b>	13 ms	testHandleMachinery()	passed 344 ms
testCreateClientError()	passed 2 ms	testHandleModels()	passed 339 ms
testCreateClientSuccess()	passed 3 ms	testHandleOrders()	passed 347 ms
testFetchAllClientsSuccess()	passed 3 ms	testHandleProcesses()	passed 342 ms
testDeleteClientSuccess204()	passed 1 ms	testHandleGantt()	passed 344 ms
testUpdateClientError()	passed 1 ms	testInitializeEmployeeRole()	passed 241 ms
testUpdateClientSuccess()	passed 2 ms	testHandleNoticeBoard()	passed 339 ms
testDeleteClientFailure()	passed		
testFetchAllClientsErrorCode()	passed 1 ms		
<b>NoticeEventListenerTest</b>	29 ms		
testRealConnectionFallback()	passed 26 ms	<b>RawControllerTestFX</b>	19.30 s
testMultipleLinesSomeValidSomeInvalid()	passed 1 ms	testHandleRefresh()	passed 216 ms
testHandlesExceptionGracefully()	passed	testHandleAddDialog()	passed 5.64 s
testStopInterruptsThread()	passed	testHandleDeleteSuccess()	passed 716 ms
testStartAndHandlerCalledWithValidEvent()	passed 2 ms	testHandleDeleteNoSelection()	passed 415 ms
<b>MachineryServiceTest</b>	12 ms	testLoadRawListSuccess()	passed 719 ms
testDeleteMachinery_success()	passed 1 ms	testHandleEditDialog()	passed 11.59 s
testDeleteMachinery_failure()	passed		
testGetAllMachineryNames_success()	passed 2 ms		
testSaveMachinery_success()	passed 1 ms	<b>ProfileControllerTestFX</b>	8.52 s
testLoadAllMachinery_success()	passed 1 ms	testInitializeAsNonEmployeeHidesTasks()	passed 718 ms
testGetMachineryByName_found()	passed 2 ms	testHandleDeleteNoProfile()	passed 719 ms
testUpdateMachinery_failureNon200()	passed 1 ms	testInitializeLoadsProfileAndTasks()	passed 715 ms
testLoadAllMachinery_failure()	passed	testHandleEditConfirm()	passed 5.65 s
testGetMachineryByName_notFound()	passed 2 ms	testInitializeWithServiceException()	passed 720 ms
testSaveMachinery_failureNon201()	passed		
test GetAllMachinery_success()	passed 1 ms	<b>EmployeeListServiceTest</b>	1 ms
testUpdateMachinery_success()	passed 1 ms	testFetchEmployeesHttpError()	passed 1 ms
testGetAllMachinery_failure_io()	passed	testFetchEmployeesSuccess()	passed
<b>ProcessServiceTest</b>	12 ms	testFetchEmployeesExceptionHandled()	passed
testAddNewStepToModelFailure()	passed 1 ms		
testDeleteByIdSuccess()	passed	<b>RawServiceTest</b>	4 ms
testDeleteByIdFailure()	passed	testCreateRaw_success()	passed 1 ms
testUpdateStepByIdSuccess()	passed 8 ms	testDeleteRaw_failure()	passed
testAddNewStepToModelSuccess()	passed 3 ms	testLoadAllRaw_success()	passed
<b>AuthControllerTest</b>	3.26 s	testGetAllRaws_success()	passed 2 ms
testHandleBack_safe()	passed 333 ms	testCreateRaw_failure()	passed
testLoadDashboard_safe()	passed 254 ms	testLoadAllRaw_failure()	passed
testHandleLogin_exception()	passed 241 ms	testGetAllRaws_failure()	passed
testHandleRegister_success()	passed 244 ms	testDeleteRaw_exception()	passed
testHandleBack_invalidFXML()	passed 241 ms	testFetchAllRaw_success()	passed
testLoadDashboard_invalidFXML()	passed 239 ms	testUpdateRaw_exception()	passed
testHandleRegister_failureResponse()	passed 246 ms	testFetchAllRaw_failure()	passed
testHandleRegister_missingFields()	passed 245 ms	testUpdateRaw_success()	passed 1 ms
testHandleLogin_missingTokenInResponse()	passed 239 ms	testUpdateRaw_failure()	passed
testHandleRegister_exception()	passed 246 ms	testCreateRaw_exception()	passed
testHandleLogin_emptyFields()	passed 246 ms	testDeleteRaw_success()	passed
testHandleRegister_nullStream()	passed 239 ms	testFetchAllRaw_exception()	passed
testHandleLogin_failureResponse()	passed 243 ms		

## **4. Conclusioni:**

Il lavoro svolto ha permesso di progettare e realizzare un sistema completo per la gestione della produzione, arricchito dalla visualizzazione dei processi tramite diagrammi di Gantt.

Gli obiettivi iniziali sono stati raggiunti: la piattaforma consente di gestire in maniera efficace ordini, risorse e processi, col giusto compromesso di completezza e flessibilità.

L'attività di testing ha confermato l'affidabilità delle componenti principali, assicurando una buona stabilità complessiva del software, e quindi un possibile deployment reale nel contesto applicativo.

In prospettiva, il sistema potrebbe esser ulteriormente ampliato, con funzionalità di scheduling più avanzate, strumenti di analisi e statistiche, e un'interfaccia più piacevole.

Pur trattandosi di un progetto sviluppato al solo scopo accademico, e pur essendo consapevoli che non sia perfetto e presenti ampi margini di miglioramento, siamo soddisfatti del risultato ottenuto. Questo lavoro rappresenta per noi un traguardo significativo, in quanto prima esperienza concreta di sviluppo software completo, dalla progettazione alla realizzazione vera e proprio.

# Collegamenti:

Repository GitHub del progetto: <https://github.com/JoJoJoJonny/ProgettoTesi>

Presentazione Canva: [Link](#)

Framework Spring (09 settembre 2025): <https://spring.io/>

Framework JavaFX (09 settembre 2025): <https://openjfx.io/>

ORM Hibernate (09 settembre 2025): <https://hibernate.org/>

Database PostgreSQL (09 settembre 2025): <https://www.postgresql.org/>

Project Manager Apache Maven (09 settembre 2025): <https://maven.apache.org/>

Repository Librerie Maven (09 settembre 2025): <https://mvnrepository.com/>

IDE IntelliJ (09 settembre 2025): <https://www.jetbrains.com/idea/>

Libreria FlexGanttFX (09 settembre 2025): <https://www.flexganttfx.com/>

Framework JUnit (09 settembre 2025): <https://junit.org/>

Framework Mockito (09 settembre 2025): <https://site.mockito.org/>

Framework TestFX (09 settembre 2025): <https://github.com/TestFX/TestFX>